Description of STM32H5 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeH5 for STM32H5 Series)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as Azure® RTOS ThreadX, USBX, and secure manager API.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors. The HAL drivers are feature-oriented instead of IP-oriented. For example, the timer APIs are split into several categories following the IP functions, such as basic timer, capture and pulse width modulation (PWM). The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities, and provide atomic operations that must be called by following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers. All operations are performed by changing the content of the associated peripheral registers. Unlike the HAL, LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or a complex upper-level stack.

The HAL and LL are complementary and cover a wide range of application requirements:

- The HAL offers high-level and feature-oriented APIs with a high-portability level. These hide the MCU and peripheral complexity from the end-user.
- The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.

The HAL- and LL-driver source code is developed in Strict ANSI-C, which makes it independent of the development tools. It is checked with the CodeSonar® static analysis tool. It is fully documented.

It is compliant with MISRA C®:2012 standard.

This user manual is structured as follows:

- Overview of HAL drivers
- Overview of low-layer drivers
- Cohabiting of HAL and LL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application

1 General information

The STM32CubeH5 MCU Package runs on STM32H5 32-bit microcontrollers based on the Arm® Cortex®-M processor with Arm® TrustZone® technology.

Note: Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Acronyms and definitions

Table 1. Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
DAC	Digital to analog converter
DCMI	Digital camera interface
DMA	Direct memory access
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMAC	Filtering mathematical calculation unit
FMC	Flexible memory controller
GPIO	General purpose I/Os
GTZC	Global TrustZone® controller
GTZC-MPCBB	GTZC block-based memory protection controller
GTZC-MPCWM	GTZC watermark memory protection controller
GTZC-TZIC	TrustZone® illegal access controller
GTZC-TZSC	TrustZone® security controller
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
I2C	Inter-integrated circuit
I3C	Improved inter-integrated circuit
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
LCD	Liquid crystal display controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter

Acronym	Definition
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SAES	Secure advanced encryption standard
SBS	System configuration, boot and security
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SRAM	SRAM external memory
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C® and Power Delivery interface
USART	Universal synchronous receiver/transmitter
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
USB	Universal serial bus
XSPI	Extended SPI interface

3 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (such as USART1 or USART2)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

3.1 HAL and user-application files

3.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32h5xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32h5xx_hal_adc.c, stm32h5xx_hal_irda.c.*
stm32h5xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32h5xx_hal_adc.h,stm32h5xx_hal_irda.h.*
stm32h5xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32h5xx_hal_adc_ex.c,stm32h5xx_hal_flash_ex.c.*
stm32h5xx_hal_ppp_ex.h	Header file of the extension C file It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32h5xx_hal_adc_ex.h,stm32h5xx_hal_flash_ex.h.*
stm32h5xx_hal.c	This file is used for HAL initialization and contains VREFBUF, SBS, DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32h5xx_hal.h	stm32h5xx_hal.c header file
stm32h5xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32h5xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application
stm32h5xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros

3.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3. User-application files

File	Description
system_stm32h5xx.c	When TrustZone® security is disabled, this file contains SystemInit() that is called at startup just after reset and before branching to the main program. It allows relocating the vector table in internal Flash memory or internal SRAM.
system_stm32h5xx_ns.c	When TrustZone® security is enabled, this file is part of the non-secure application. It contains SystemInit() that is called by the secure application before branching to the non-secure main program. It allows relocating the vector table in internal Flash memory or internal SRAM.
startup_stm32h5xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32h5xx_flash.icf/ stm32h5xx_flash.icf	When TrustZone® security is disabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the application requirements.

File	Description
stm32h5xx_flash_s.icf/ stm32h5xx_flash_s.icf	When TrustZone® security is enabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the secure application requirements.
stm32h5xx_flash_ns.icf/ stm32h5xx_flash_ns.icf	When TrustZone® security is enabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the non-secure application requirements.
stm32h5xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32h5xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32h5xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32h5xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

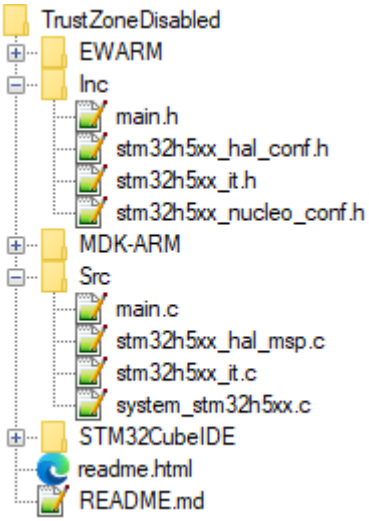
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_GetTick()
  - System clock configured with the selected device frequency.

Note: If an existing project is copied to another location, then include paths must be updated.

Figure 1. Example of project template



3.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

3.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that enables working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
  Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
   - *Reentrant code does not hold any static (or global) non-constant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (for example serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.*
   - *Reentrant code does not modify its own code.*
2. *When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.*
3. *For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:*
   - *GPIO*
   - *SYSTICK*
   - *NVIC*
   - *PWR*
   - *RCC*
   - *FLASH*

3.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
  uint32_t BaudRate;       /*!< This member configures the Usart communication baud rate.
                                The baud rate is computed using the following formula:
                                Baud Rate Register[15:4] = ((2 * fclk_pres) /
                                ((huart->Init.BaudRate)))[15:4]
                                Baud Rate Register[3]    = 0
                                Baud Rate Register[2:0]  =  (((2 * fclk_pres) /
                                ((huart->Init.BaudRate)))[3:0]) >> 1
                                where fclk_pres is the USART input clock frequency (fclk)
                                divided by a prescaler.
                                @note  Oversampling by 8 is systematically applied to
                                achieve high baud rates. */
  uint32_t WordLength;          /*!< Specifies the number of data bits
                                transmitted or received in a frame.This
              parameter can be a value of                                                USAR
TEx_Word_Length. */
  uint32_t StopBits;            /*!< Specifies the number of stop bits transmitted.
                                This parameter can be a value of USART_Stop_Bits. */
  uint32_t Parity;              /*!< Specifies the parity mode.
                                This parameter can be a value of USART_Parity
                                @note When parity is enabled, the computed parity is inserted
                                at the MSB position of the transmitted data (9th bit when
                                the word length is set to 9 data bits; 8th bit when the
                                word length is set to 8 data bits). */
  uint32_t Mode;                /*!< Specifies whether the Receive or
              Transmit mode is enabled or disabled.
                                This parameter can be a value of
              USART_Mode. */
  uint32_t CLKPolarity;         /*!< Specifies the steady state of the serial clock.
                                This parameter can be a value of USART_Clock_Polarity. */
  uint32_t CLKPhase;            /*!< Specifies the clock transition on
              which the bit capture is made.
                                This parameter can be a value of USART_Clock_Phase. */
  uint32_t CLKLastBit;          /*!< Specifies whether the clock pulse
              corresponding to the last transmitted data                                 bit
(MSB) has to be output on the SCLK pin                                  in synchronous mode.
                                This parameter can be a value of USART_Last_Bit. */
  uint32_t ClockPrescaler;      /*!< Specifies the prescaler value used to
              divide the USART clock source.
                                This parameter can be a value of USART_ClockPrescaler. */
} USART_InitTypeDef;
```

Note: The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, const ADC_ChannelConfTypeDef *pConfig)
```

3.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

3.3 API classification

The HAL APIs are classified into the following categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:**
  This set of API is divided into two sub-categories :
  - **Family specific APIs**: APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t Si
ngleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDi
ff);
```

  - **Device part number specific APIs:**These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.

Note: The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4. API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs	-	X
Device specific APIs	-	X

Note: Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

The IRQ handlers are used for common and family specific processes.

3.4 Devices supported by HAL/LL drivers

Table 5. List of devices supported by HAL/LL drivers

IP/Module	STM32H573xx	STM32H563xx	STM32H562xx	STM32H503xx
stm32h5xx_hal.c	YES	YES	YES	YES
stm32h5xx_hal_adc.c	YES	YES	YES	YES
stm32h5xx_hal_adc_ex.c	YES	YES	YES	YES
stm32h5xx_hal_cec.c	YES	YES	YES	NO
stm32h5xx_hal_comp.c	NO	NO	NO	YES
stm32h5xx_hal_cordic.c	YES	YES	YES	NO
stm32h5xx_hal_cortex.c	YES	YES	YES	YES
stm32h5xx_hal_crc.c	YES	YES	YES	YES
stm32h5xx_hal_crc_ex.c	YES	YES	YES	YES
stm32h5xx_hal_cryp.c	YES	NO	NO	NO
stm32h5xx_hal_cryp_ex.c	YES	NO	NO	NO
stm32h5xx_hal_dac.c	YES	YES	YES	YES
stm32h5xx_hal_dac_ex.c	YES	YES	YES	YES
stm32h5xx_hal_dcache.c	YES	YES	YES	NO
stm32h5xx_hal_dcmi.c	YES	YES	YES	NO
stm32h5xx_hal_dma.c	YES	YES	YES	YES
stm32h5xx_hal_dma_ex.c	YES	YES	YES	YES
stm32h5xx_hal_dts.c	YES	YES	YES	YES
stm32h5xx_hal_eth.c	YES	YES	NO	NO
stm32h5xx_hal_eth_ex.c	YES	YES	NO	NO
stm32h5xx_hal_exti.c	YES	YES	YES	YES
stm32h5xx_hal_fdcan.c	YES	YES	YES	YES
stm32h5xx_hal_flash.c	YES	YES	YES	YES
stm32h5xx_hal_flash_ex.c	YES	YES	YES	YES
stm32h5xx_hal_fmac.c	YES	YES	YES	NO
stm32h5xx_hal_gpio.c	YES	YES	YES	YES
stm32h5xx_hal_gtzc.c	YES	YES	YES	YES
stm32h5xx_hal_hash.c	YES	YES	YES	YES
stm32h5xx_hal_hcd.c	YES	YES	YES	YES
stm32h5xx_hal_i2c.c	YES	YES	YES	YES
stm32h5xx_hal_i2c_ex.c	YES	YES	YES	YES
stm32h5xx_hal_i2s.c	YES	YES	YES	YES
stm32h5xx_hal_i2s_ex.c	YES	YES	YES	YES
stm32h5xx_hal_i3c.c	YES	YES	YES	YES
stm32h5xx_hal_icache.c	YES	YES	YES	YES
stm32h5xx_hal_irda.c	YES	YES	YES	YES
stm32h5xx_hal_iwdg.c	YES	YES	YES	YES
stm32h5xx_hal_lptim.c	YES	YES	YES	YES

IP/Module	STM32H573xx	STM32H563xx	STM32H562xx	STM32H503xx
stm32h5xx_hal_mmc.c	YES	YES	YES	NO
stm32h5xx_hal_mmc_ex.c	YES	YES	YES	NO
stm32h5xx_hal_msp_template.c	YES	YES	YES	YES
stm32h5xx_hal_nand.c	YES	YES	YES	NO
stm32h5xx_hal_nor.c	YES	YES	YES	NO
stm32h5xx_hal_opamp.c	NO	NO	NO	YES
stm32h5xx_hal_opamp_ex.c	NO	NO	NO	YES
stm32h5xx_hal_otfdec.c	YES	NO	NO	NO
stm32h5xx_hal_pcd.c	YES	YES	YES	YES
stm32h5xx_hal_pcd_ex.c	YES	YES	YES	YES
stm32h5xx_hal_pka.c	YES	NO	NO	NO
stm32h5xx_hal_pssi.c	YES	YES	YES	NO
stm32h5xx_hal_pwr.c	YES	YES	YES	YES
stm32h5xx_hal_pwr_ex.c	YES	YES	YES	YES
stm32h5xx_hal_ramcfg.c	YES	YES	YES	YES
stm32h5xx_hal_rcc.c	YES	YES	YES	YES
stm32h5xx_hal_rcc_ex.c	YES	YES	YES	YES
stm32h5xx_hal_rng.c	YES	YES	YES	YES
stm32h5xx_hal_rng_ex.c	YES	YES	YES	YES
stm32h5xx_hal_rtc.c	YES	YES	YES	YES
stm32h5xx_hal_rtc_ex.c	YES	YES	YES	YES
stm32h5xx_hal_sai.c	YES	YES	YES	NO
stm32h5xx_hal_sai_ex.c	YES	YES	YES	NO
stm32h5xx_hal_sd.c	YES	YES	YES	NO
stm32h5xx_hal_sdram.c	YES	YES	YES	NO
stm32h5xx_hal_sd_ex.c	YES	YES	YES	NO
stm32h5xx_hal_smartcard.c	YES	YES	YES	YES
stm32h5xx_hal_smartcard_ex.c	YES	YES	YES	YES
stm32h5xx_hal_smbus.c	YES	YES	YES	YES
stm32h5xx_hal_smbus_ex.c	YES	YES	YES	YES
stm32h5xx_hal_spi.c	YES	YES	YES	YES
stm32h5xx_hal_spi_ex.c	YES	YES	YES	YES
stm32h5xx_hal_sram.c	YES	YES	YES	NO
stm32h5xx_hal_tim.c	YES	YES	YES	YES
stm32h5xx_hal_timebase_rtc_alarm_template.c	YES	YES	YES	YES
stm32h5xx_hal_timebase_rtc_wakeup_template.c	YES	YES	YES	YES
stm32h5xx_hal_timebase_tim_template.c	YES	YES	YES	YES
stm32h5xx_hal_tim_ex.c	YES	YES	YES	YES
stm32h5xx_hal_uart.c	YES	YES	YES	NO
stm32h5xx_hal_uart_ex.c	YES	YES	YES	NO
stm32h5xx_hal_usart.c	YES	YES	YES	YES

IP/Module	STM32H573xx	STM32H563xx	STM32H562xx	STM32H503xx
stm32h5xx_hal_usart_ex.c	YES	YES	YES	YES
stm32h5xx_hal_wwdg.c	YES	YES	YES	YES
stm32h5xx_hal_xspi.c	YES	YES	YES	NO
stm32h5xx_ll_adc.c	YES	YES	YES	YES
stm32h5xx_ll_comp.c	NO	NO	NO	YES
stm32h5xx_ll_cordic.c	YES	YES	YES	NO
stm32h5xx_ll_crc.c	YES	YES	YES	YES
stm32h5xx_ll_crs.c	YES	YES	YES	YES
stm32h5xx_ll_dac.c	YES	YES	YES	YES
stm32h5xx_ll_dlyb.c	YES	YES	YES	NO
stm32h5xx_ll_dma.c	YES	YES	YES	YES
stm32h5xx_ll_exti.c	YES	YES	YES	YES
stm32h5xx_ll_fmac.c	YES	YES	YES	NO
stm32h5xx_ll_fmc.c	YES	YES	YES	NO
stm32h5xx_ll_gpio.c	YES	YES	YES	YES
stm32h5xx_ll_i2c.c	YES	YES	YES	YES
stm32h5xx_ll_i3c.c	YES	YES	YES	YES
stm32h5xx_ll_icache.c	YES	YES	YES	YES
stm32h5xx_ll_lptim.c	YES	YES	YES	YES
stm32h5xx_ll_lpuart.c	YES	YES	YES	YES
stm32h5xx_ll_opamp.c	NO	NO	NO	YES
stm32h5xx_ll_pka.c	YES	NO	NO	NO
stm32h5xx_ll_pwr.c	YES	YES	YES	YES
stm32h5xx_ll_rcc.c	YES	YES	YES	YES
stm32h5xx_ll_rng.c	YES	YES	YES	YES
stm32h5xx_ll_rtc.c	YES	YES	YES	YES
stm32h5xx_ll_sdmmc.c	YES	YES	YES	NO
stm32h5xx_ll_spi.c	YES	YES	YES	YES
stm32h5xx_ll_tim.c	YES	YES	YES	YES
stm32h5xx_ll_ucpd.c	YES	YES	YES	NO
stm32h5xx_ll_usart.c	YES	YES	YES	YES
stm32h5xx_ll_usb.c	YES	YES	YES	YES
stm32h5xx_ll_utils.c	YES	YES	YES	YES
stm32h5xx_util_i3c.c	YES	YES	YES	YES

3.5 HAL driver rules

3.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. HAL API naming rules

	Generic	Family specific APIs	Device specific APIs
File names	*stm32h5xx_hal_ppp (c/h)*	*stm32h5xx_hal_ppp_ex (c/h)*	*stm32h5xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32H5 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in the stm32h5xxx.h header file:
stm32h5xxx.h corresponds to stm32h573xx, stm32h563xx, stm32h562xx and stm32h503xx.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (for example HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (for example ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (for example ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (for example HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (for example HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called:
HAL_PPP_Function_DMA().
- The **Feature** prefix should refer to the new feature.
Example: *HAL_ADCEx_InjectedStart()*() refers to the injection mode.

3.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32h5xx_hal_cortex.c file.
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
 HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
 {
 return HAL_ERROR;
 }
```

- The macros defined below are used:
  - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

  - Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32h5xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Example: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Example: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

3.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (such as PWM, OC and IC).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()* function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in run time the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9. HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in run time the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in run time the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32h5xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32h5xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

Function group	Common API name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration
HAL_ADCEx_InjectedStart	This function is used to start the conversion of injected groups.

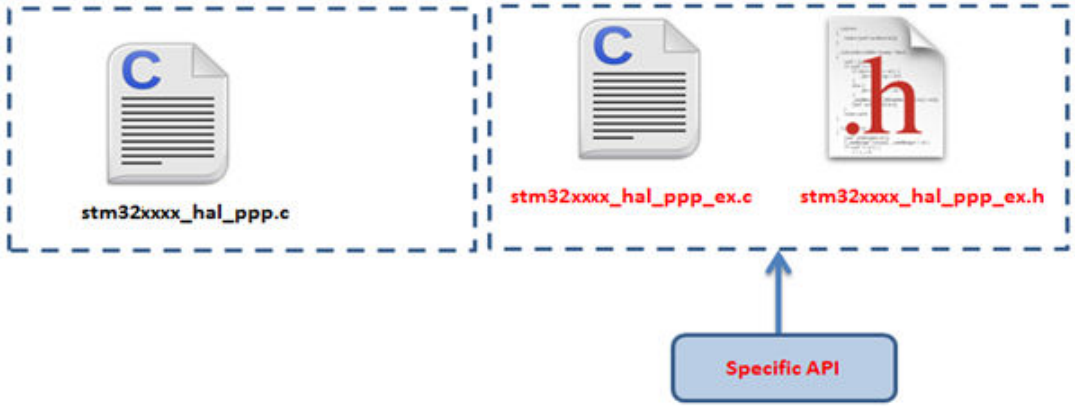
3.7.2 HAL extension model cases

The specific peripheral features can be handled by the HAL drivers in five different ways. They are described below.

Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32h5xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

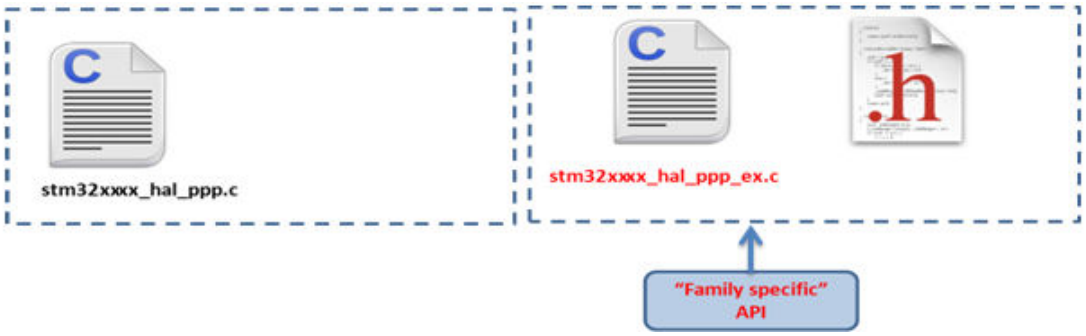
Figure 2. Adding device-specific functions



Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 3. Adding family-specific functions

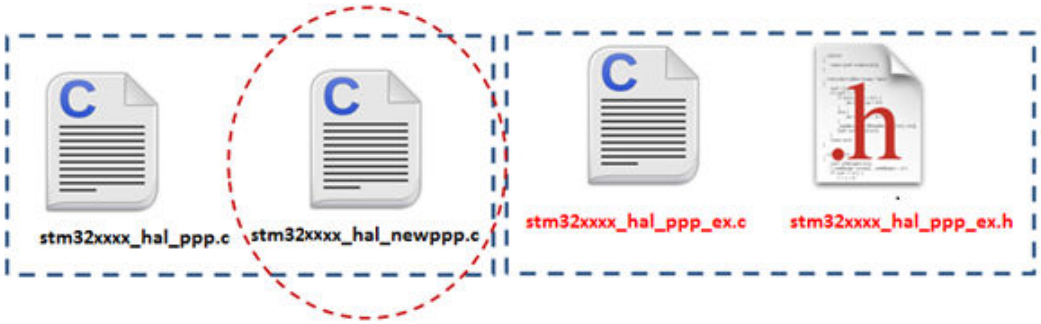


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32h5xx_hal_newppp.c. However the inclusion of this file is selected in the stm32h5xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. **Adding new peripherals**

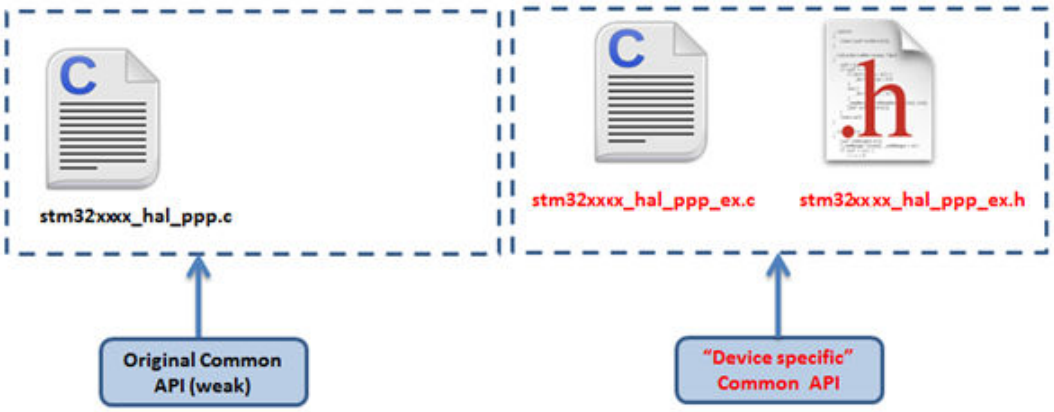


Example: stm32h5xx_hal_adc.c/h

Updating existing common APIs

In this case, the routines are defined with the same names in the stm32h5xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler overwrites the original routine by the new defined function.

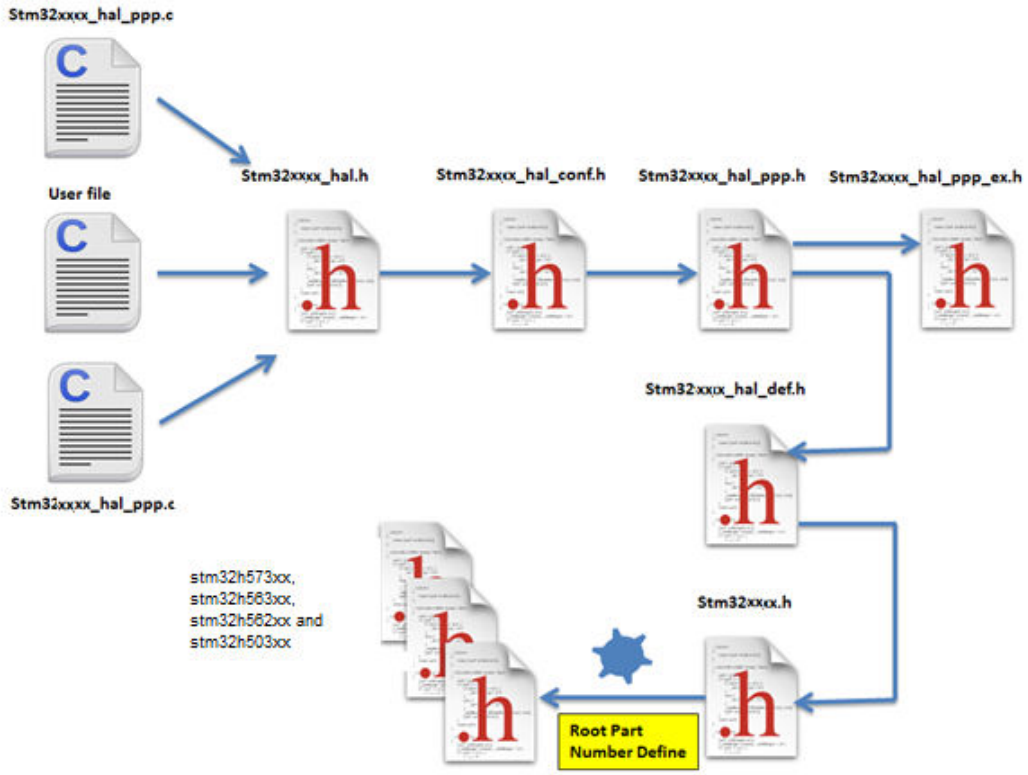
Figure 5. **Updating existing APIs**



3.8 File inclusion model

The header of the common HAL driver file (stm32h5xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6. File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/**********************************************************************
* @file stm32h5xx_hal_conf.h
* @author MCD Application Team
* @brief   HAL configuration file.
**********************************************************************
(…)
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
(…)
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32h5xx_hal_def.h*.The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status**
  The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{
HAL_OK = 0x00,
HAL_ERROR = 0x01,
HAL_BUSY = 0x02,
HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked**
  The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32h5xx_hal_def.h file calls the stm32h5xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:
  - Declarations of peripheral registers and bits definition.
  - Macros to access peripheral registers hardware (such as Write register or Read register).
- **Common macros**
  - Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  - Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
  (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
  (__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.10 HAL configuration

The configuration file, stm32h5xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	100 ms
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	64 000 000 Hz
HSI48_VALUE	Value of the internal high speed oscillator for USB FS/SDMMC/RNG expressed in Hz. The real value may vary depending on manufacturing process.	48000000 Hz
CSI_VALUE	Defines the default value of the low-power internal oscillator (CSI) expressed in Hz.	4 000 000 Hz

Configuration item	Description	Default Value
LSI_VALUE	Defines the default value of the Low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSI_STARTUP_TIMEOUT	Timeout for LSI startup, expressed in ms	130 ms
LSE_VALUE	Defines the value of the external oscillator (LSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000 Hz
VDD_VALUE	V_{DD} value	3300 mV
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch	FALSE

Note: The stm32h5xx_hal_conf_template.h file is located in the HAL drivers Inc folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32h5xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

3.11 HAL system peripheral handling

This section gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

3.11.1 Clocks

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, HSI48, CSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - selects the system clock source
  - configures APB1, APB2, and APB3 clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in stm32h5xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (such as system clock, HCLK, PCLK1, PCLK2, or PCLK3)
- MCO and CSS configuration functions

A set of macros are defined in stm32h5xx_hal_rcc.h and stm32h5xx_hal_rcc_ex.h. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __HAL_PPP_CLK_ENABLE/__HAL_PPP_CLK_DISABLE to enable/disable the peripheral clock.
- __HAL_PPP_FORCE_RESET/__HAL_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_PPP_CLK_SLEEP_ENABLE/__HAL_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during Sleep mode.
- __HAL_PPP_IS_CLK_ENABLED/__HAL_PPP_IS_CLK_DISABLED to query about the enabled/disabled status of the peripheral clock.
- __HAL_PPP_IS_CLK_SLEEP_ENABLED/__HAL_PPP_IS_CLK_SLEEP_DISABLED to query about the enabled/disabled status of the peripheral clock during Sleep mode.

3.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().
- HAL_GPIO_LockPin()/HAL_GPIO_WritePin()
- HAL_GPIO_EnableHighSPeedLowVoltage()/HAL_GPIO_DisableHighSPeedLowVoltage()

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32h5xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode – GPIO_MODE_ANALOG_ADC_CONTROL: ADC analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

3.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32h5xx_hal_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()
- HAL_MPU_Enable()
- HAL_MPU_Enable_NS()
- HAL_MPU_Disable()
- HAL_MPU_Disable_NS()
- HAL_MPU_ConfigRegion()
- HAL_MPU_ConfigRegion_NS()
- HAL_MPU_ConfigMemoryAttributes()
- HAL_MPU_ConfigMemoryAttributes_NS()
- MPU_ConfigRegion()
- MPU_ConfigMemoryAttributes()

3.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWR_PVD_IRQHandler()
  - HAL_PWR_PVDCallback()
- AVD configuration, enabling/disabling and interrupt handling:
  - HAL_PWREx_ConfigAVD()
  - HAL_PWREx_EnableAVD() / HAL_PWREx_DisableAVD()
  - HAL_PWREx_AVD_PVD_IRQHandler()
  - HAL_PWREx_PVD_AVD_Rising_Callback()
  - HAL_PWREx_PVD_AVD_Falling_Callback()

- Wakeup pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTOPMode() (kept for compatibility with other families)
  - HAL_PWR_EnterSTANDBYMode()
- STM32H5 low-power management features:
  - HAL_PWR_EnableSleepOnExit() / HAL_PWR_DisableSleepOnExit: (low-power management Sleep-on-entry)
  - HAL_PWR_EnableSEVOnPend() / HAL_PWR_DisableSEVOnPend (low-power management Cortex-SEV-on-pend)

3.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs. In addition, each peripheral HAL driver implements the associated EXTI configuration and function as macros in its header file.

The first 16 EXTI lines connected to the GPIOs, are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD/AVD, RTC, USB, and Ethernet, are configured within the HAL drivers of these peripheral through the macros given in the table below.

The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13. Description of EXTI configuration macros

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_PWR_PVD_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_PWR_PVD_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Falling edge

Macros	Description
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32h5xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

3.11.6 DMA

DMA transfer modes are divided into two main categories:

- Normal transfers (as for legacy Series)
- Linked-list transfers

In addition to linked-list capability, all the advanced DMA features are managed and configured via the extended API as extensions to Normal mode. The advanced features are repeated block, trigger and data handling.

For a given channel, enable and configure the peripheral to be connected to the DMA channel (except for internal SRAM and Flash memories for which no initialization is required). Refer to the device reference manuals for details on the connection between peripherals and DMA requests.

The Normal transfer mode is initialized via the standard API while the Linked-list mode is configured via the extended API.

Alternatively to the Normal programming mode, a DMA channel can be programmed by a list of transfers, also known as linked-list (list of node items), each node being defined by its data structure and specifying a standalone DMA channel. When enabled, the DMA channel fetches the first linked-list node from SRAM (also known as head node). When the execution is complete, it fetches and executes the next linked-list node. This operation is repeated until the end of the linked-list queue. The linked-list can be linear or circular:

- Linear linked-list:
  The last linked-list queue node is not linked to another queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node (head node) to the last (tail node). When the last node execution is complete, the DMA channel remains in idle state and another transfer can be launched.
- Circular linked-list:
  The last linked-list node can be linked to any linked-list queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node to the last. When the last node is executed, the DMA channel fetches again the first circular node and repeats the same sequence in an infinite loop (circular transfer). An abort operation is required to stop the DMA channel. The Circular linked-list mode replaces the legacy Circular transfer mode.

To reduce linked-list queue execution time and power consumption, the DMA channel supports dynamic linked-list execution in two types of formats, static and dynamic:

- Static linked-list format: this format refers to the full linked-list node where all DMA channel parameters are fetched and executed independently of the redundancy of information.
- Dynamic linked-list: this format refers to the customized linked-list node where only the necessary DMA channel parameters are fetched and executed (for example, no update is required if data size = 20 on previous node, and data size = 20 on the current node).

For linked-list transfers, the DMA channel can execute the linked-list queue node by node. This feature is named link-step mode. When activated, enabling the DMA channel for the first time fetches the head node from memory and stops. Another DMA channel enable is then required to execute the node. After that, the DMA channel must remain enabled to execute each node until the end of the linked-list queue. When the linked-list queue is circular, the DMA channel must be enabled in an infinite loop to keep the DMA channel running. This feature is useful for debug purposes or for executing queue nodes asynchronously.

Each DMA channel transfer (normal or linked-list) can be configured depending on the DMA channel instance embedded in the devices. The configurations are the following:

- Repeated block configuration:
  This feature is also named two-dimension addressing transfer. If it is supported, the DMA channel can perform repeated transfers of programmed blocks (the size of the block transfer is the legacy data size). In addition to the block repeat count, the DMA channel addresses can jump at burst or block level. The jump length is a programmable parameter defined in the user application:
  - Jump at burst level: the DMA channel keeps an empty area between two consecutive transmitted bursts.
  - Jump at block level; the DMA channel keeps an empty area between two consecutive transmitted blocks .
- Trigger configuration:
  The DMA channel transfers can be triggered on the rising or falling edge of a given signal (hardware triggers). The trigger condition can be applied at single/burst level, block level, repeated block level or node level.
  The DMA channel can report a trigger overrun when more than two trigger signal events are detected before executing the current transfer.
- Data handling:
  The data handling feature is based on a FIFO capability that can be:
  - Padding pattern: the padding pattern (zero padding or sign extension) is selected when the source data width is smaller than the destination data width.
  - Truncation: the source data is truncated when the source data width is bigger than the destination data width.
  - Pack/unpack: a set of data can be packed when the source data width is smaller than the destination data width or unpacked when the source data width is bigger than the destination data width.
  - Exchange: data can be exchanged at byte and half-word level on the destination, and byte level on the source.

Each ongoing DMA channel transfer (normal or linked-list) can be suspended and resumed at run time by the application. When trying to suspend an ongoing transfer, the DMA channel is not suspended instantly. Instead, the ongoing single/burst transfer completes and then stops.

When the DMA channel is suspended, the current transfer can be resumed instantly.

The DMA channel that supports the FIFO can report in real time the number of beats remaining on the destination FIFO ((output).

Two operating modes are available:

- Polling mode I/O operation
  1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  2. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority().
  2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ().
  3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine.
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (that is a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer.

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel

- __HAL_DMA_DISABLE: disables the specified DMA channel
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not

To correctly use the DMA linked-list feature, apply the following call sequences:

- Linear transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()
- Circular transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_SetCircularModeConfig()/HAL_DMAEx_List_SetCircularMode()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMA_Abort() / HAL_DMA_Abort_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()

Note: When a peripheral is used in DMA mode, the DMA initialization must be done in the HAL_PPP_MspInit() callback. In addition, the user application must associate the DMA handle to the PPP handle (refer to section "HAL IO operation functions").

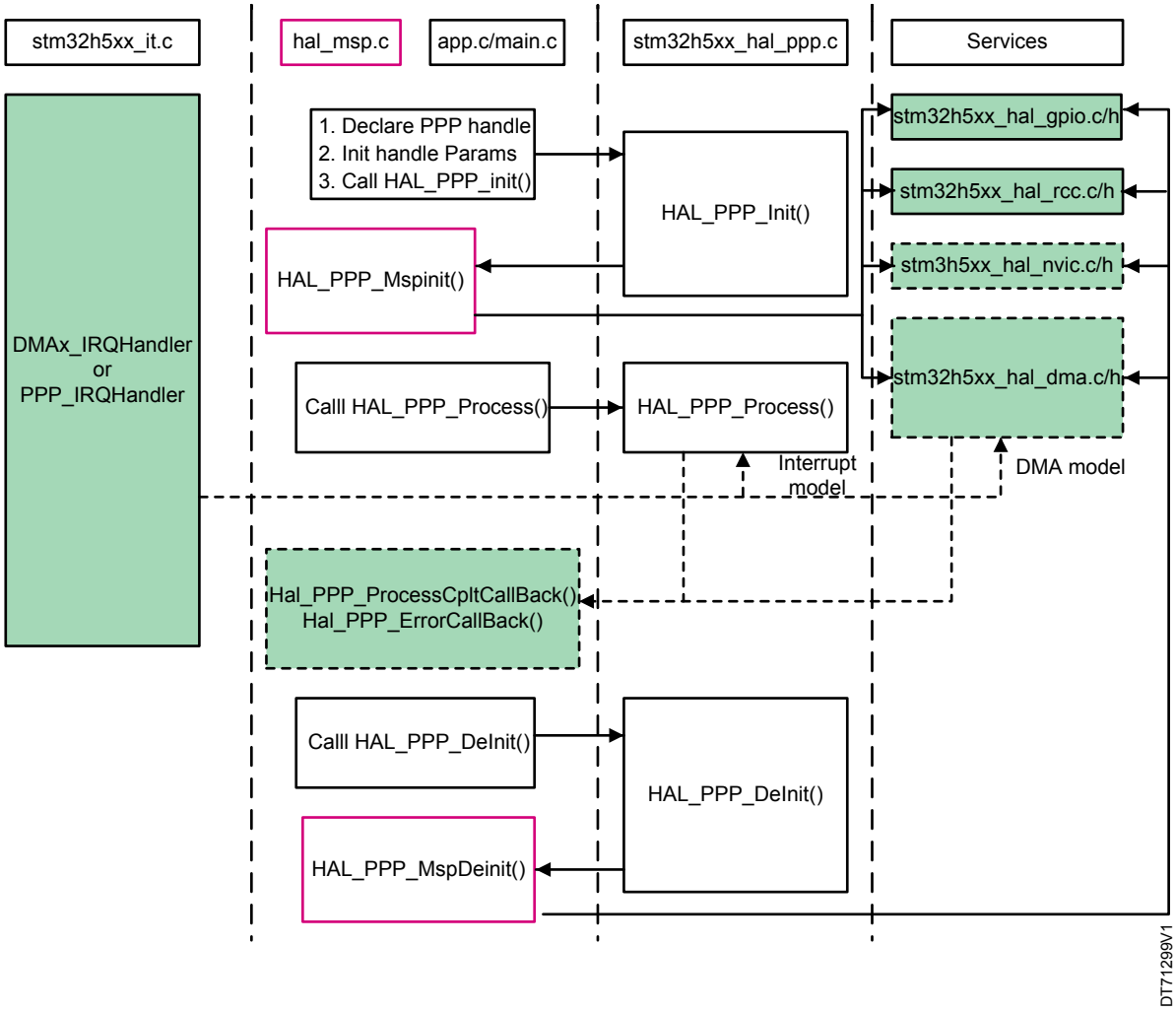
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

3.12 How to use HAL drivers

3.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 7. **HAL driver model**



Note: The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Non-dotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

3.12.2 HAL initialization

3.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32h5xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.

- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which is a user callback function to do system level De-Initalizations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR is blocked.

3.12.2.2 *System clock initialization*

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code.

Below the typical clock configuration sequence to reach the maximum clock frequency of 250 MHz based on the HSE clock.

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE0);

  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

  /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    * Configure PLLs :
    * PLL configuration: PLLCLK = (HSE/PLLM * PLLN) / PLLR = (8/4 * 250) / 2 = 250 MHz
    * Enable HSE and activate PLL with HSE as source
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS_DIGITAL;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 250;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLLVCIRANGE_1;
  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLFRACN = 0;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
                              |RCC_CLOCKTYPE_PCLK3;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
```

```
  }
}
```

3.12.2.3 *HAL MSP initialization process*

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

The MspInit callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32h5xx_hal_msp.c* file in the user folders. An *stm32h5xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32h5xx_hal_msp.c file contains the following functions:

Table 14. MSP functions

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit()*. In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

3.12.3 HAL I/O operation process

The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode

- DMA mode

3.12.3.1 *Polling mode*

In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical Polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HAL_OK; }
```

3.12.3.2 *Interrupt mode*

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In Interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launches the process
- *HAL_PPP_IRQHandler()*: global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: callback relative to the process Error.

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32h5xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

*stm32h5xx_it.c*file:

3.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32h5xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
```

```
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

stm32h5xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

3.12.4 Timeout and error management

3.12.4.1 *Timeout management*

The timeout is often used for the APIs that operate in Polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel, ui
nt32_t Timeout)
```

The timeout possible values are the following:

Table 15. Timeout values

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) (1)	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

1. *HAL_MAX_DELAY is defined in the stm32h5xx_hal_def.h as 0xFFFFFFFF*

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef *hppp)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if ((HAL_GetTick() - tickstart) > LOCAL_PROCESS_TIMEOUT)
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_StatusTypeDef HAL_PPP_Poll(PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if (((HAL_GetTick() - tickstart) > Timeout) || (Timeout == 0))
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

3.12.4.2 *Error management*

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32 Size
)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
if (hppp == NULL) //the handle should be already allocated
{
return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{
timeout = HAL_GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

3.12.4.3 *Run-time checking*

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *stm32h5xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)
```

```
/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32h5xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions ---------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
}
}
```

Attention: ***Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.***

4 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

4.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

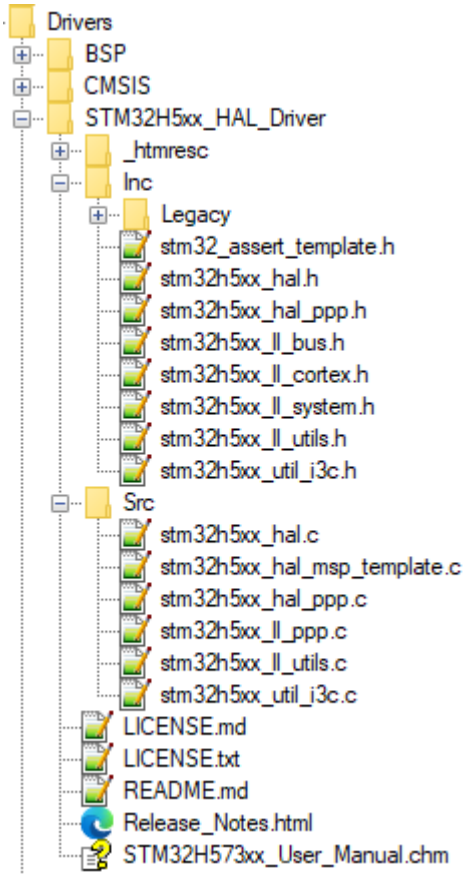
Table 16. LL driver files

File	Description
stm32h5xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32h5xx_ll_ppp.h/.c	stm32h5xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32h5xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32h5xx_ll_ppp.h file.
stm32h5xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32h5xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32h5xx_ll_system.h	System related operations. Example: LL_SBS_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

Note: There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

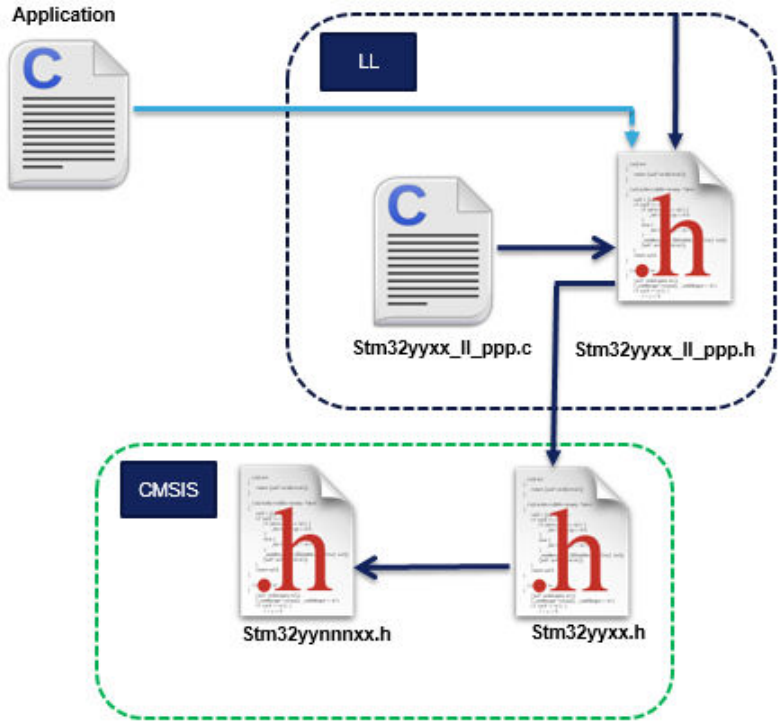
Figure 8. **Low-layer driver folders**



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 9. **Low-layer driver CMSIS files**



Application files have to include only the used low-layer driver header files.

4.2 Overview of low-layer APIs and naming rules

4.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in stm32h5xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17. Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to Table 18. Optional peripheral initialization functions).

Table 18. Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each *PPP_CommonInitStruct* member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

Functions	Return Type	Parameters	Examples
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

4.2.1.1 *Run-time checking*

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details, refer to Section 3.12.4.3 Run-time checking.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Note: Run-time checking is not available for LL inline functions.

4.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19. Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20. Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

Note: BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21. Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA| LL_AHB2_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

Note: 'x' corresponds to the group index and refers to the index of the modified register on a given bus. 'bus' corresponds to the bus name.

- ***Peripheral activation/deactivation management*** : Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22. Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management*** : Set/get a peripheral configuration settings

Table 23. Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management*** : Write/read the content of a register/retrun DMA relative register address

Table 24. Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

Note: The Propriety is a variable used to identify the DMA transfer direction or the data register type.

5 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

5.1 Low-layer driver used in Standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32h5xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeH5 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.

Note: When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

5.2 Mixed use of low-layer APIs and HAL drivers

In this case the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, Flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32h5 firmware package (refer to Examples_MIX projects).

Note:

1. *When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

6 HAL System Driver

6.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

6.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

6.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initializes the Flash interface the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- De-Initializes common part of the HAL.
- Configure The time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

6.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier

This section contains the following APIs:

- ***HAL_IncTick()***
- ***HAL_GetTick()***

- *HAL_GetTickPrio()*
- *HAL_SetTickFreq()*
- *HAL_GetTickFreq()*
- *HAL_Delay()*
- *HAL_SuspendTick()*
- *HAL_ResumeTick()*
- *HAL_GetHalVersion()*
- *HAL_GetREVID()*
- *HAL_GetDEVID()*
- *HAL_GetUIDw0()*
- *HAL_GetUIDw1()*
- *HAL_GetUIDw2()*

6.1.4 HAL Debug functions

This section provides functions allowing to:

- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- *HAL_DBGMCU_EnableDBGStopMode()*
- *HAL_DBGMCU_DisableDBGStopMode()*
- *HAL_DBGMCU_EnableDBGStandbyMode()*
- *HAL_DBGMCU_DisableDBGStandbyMode()*

6.1.5 HAL VREFBUF Control functions

This section provides functions allowing to:

- Configure the Voltage reference buffer
- Enable/Disable the Voltage reference buffer

This section contains the following APIs:

- *HAL_VREFBUF_VoltageScalingConfig()*
- *HAL_VREFBUF_HighImpedanceConfig()*
- *HAL_VREFBUF_TrimmingConfig()*
- *HAL_EnableVREFBUF()*
- *HAL_DisableVREFBUF()*

6.1.6 HAL SBS configuration functions

This section provides functions allowing to:

- Select the Ethernet PHY Interface
- Enable/Disable the VDD I/Os Compensation Cell
- Code selection/configuration for the VDD I/O Compensation cell
- Get ready flag status of VDD I/Os Compensation cell
- Get PMOS/NMOS compensation value of the I/Os supplied by VDD
- Enable/Disable the NMI in case of double ECC error in FLASH Interface

This section contains the following APIs:

- *HAL_SBS_ETHInterfaceSelect()*
- *HAL_SBS_EnableVddIO1CompensationCell()*
- *HAL_SBS_DisableVddIO1CompensationCell()*
- *HAL_SBS_EnableVddIO2CompensationCell()*
- *HAL_SBS_DisableVddIO2CompensationCell()*
- *HAL_SBS_VDDCompensationCodeSelect()*

- *HAL_SBS_VDDIOCompensationCodeSelect()*
- *HAL_SBS_GetVddIO1CompensationCellReadyFlag()*
- *HAL_SBS_GetVddIO2CompensationCellReadyFlag()*
- *HAL_SBS_VDDCompensationCodeConfig()*
- *HAL_SBS_VDDIOCompensationCodeConfig()*
- *HAL_SBS_GetNMOSVddCompensationValue()*
- *HAL_SBS_GetPMOSVddCompensationValue()*
- *HAL_SBS_GetNMOSVddIO2CompensationValue()*
- *HAL_SBS_GetPMOSVddIO2CompensationValue()*
- *HAL_SBS_FLASH_DisableECCNMI()*
- *HAL_SBS_FLASH_EnableECCNMI()*
- *HAL_SBS_FLASH_ECCNMI_IsDisabled()*

6.1.7 HAL SBS Boot control functions

This section provides functions allowing to:

- Increment the HDPL value
- Get the HDPL value

This section contains the following APIs:

- *HAL_SBS_IncrementHDPLValue()*
- *HAL_SBS_GetHDPLValue()*

6.1.8 HAL SBS Hardware secure storage control functions

This section provides functions allowing to:

- Select EPOCH security sent to SAES IP
- Set/Get EPOCH security selection
- Set/Get the OBK-HDPL Value

This section contains the following APIs:

- *HAL_SBS_EPOCHSelection()*
- *HAL_SBS_GetEPOCHSelection()*
- *HAL_SBS_SetOBKHDPL()*
- *HAL_SBS_GetOBKHDPL()*

6.1.9 SBS Debug control functions

This section provides functions allowing to:

- Open the device access port
- Open the debug
- Configure the authenticated debug HDPL
- Get the current value of the hide protection level
- Lock the access to the debug control register
- Configure/Get the authenticated debug security access

This section contains the following APIs:

- *HAL_SBS_OpenAccessPort()*
- *HAL_SBS_OpenDebug()*
- *HAL_SBS_ConfigDebugLevel()*
- *HAL_SBS_GetDebugLevel()*
- *HAL_SBS_LockDebugConfig()*
- *HAL_SBS_ConfigDebugSecurity()*
- *HAL_SBS_GetDebugSecurity()*

6.1.10 SBS lock functions

This section contains the following APIs:

- *HAL_SBS_Lock()*
- *HAL_SBS_GetLock()*

6.1.11 SBS attributes functions

This section contains the following APIs:

- *HAL_SBS_ConfigAttributes()*
- *HAL_SBS_GetConfigAttributes()*

6.1.12 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

Configure the Flash prefetch, the time base source, NVIC and any required global low level hardware by calling the HAL_MspInit() callback function to be optionally defined in user file stm32h5xx_hal_msp.c.

Return values

- **HAL:** status

Notes

- HAL_Init() function is called at the beginning of program after reset and before the clock configuration.
- In the default implementation the System Timer (Systick) is used as source of time base. The Systick configuration is based on HSI clock, as HSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4. Once done, time base tick starts incrementing: the tick variable counter is incremented each 1ms in the SysTick_Handler() interrupt handler.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-Initializes common part of the HAL and stops the systick.

Return values

- **HAL:** status

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initializes the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().
- In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name

void HAL_IncTick (void)

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in Systick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

HAL_StatusTypeDef HAL_SetTickFreq (HAL_TickFreqTypeDef Freq)

Function description

Set new tick Freq.

Return values

- **HAL:** status

HAL_GetTickFreq

Function name

HAL_TickFreqTypeDef HAL_GetTickFreq (void)

Function description

Return tick frequency.

Return values

- **Tick:** frequency. Value of HAL_TickFreqTypeDef.

HAL_SuspendTick

Function name

void HAL_SuspendTick (void)

Function description

Suspend Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

void HAL_ResumeTick (void)

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

uint32_t HAL_GetHalVersion (void)

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

uint32_t HAL_GetREVID (void)

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

HAL_GetDEVID

Function name

uint32_t HAL_GetDEVID (void)

Function description

Returns the device identifier.

Return values

- **Device:** identifier

HAL_GetUIDw0

Function name

uint32_t HAL_GetUIDw0 (void)

Function description

Return the first word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw1

Function name

uint32_t HAL_GetUIDw1 (void)

Function description

Return the second word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw2

Function name

uint32_t HAL_GetUIDw2 (void)

Function description

Return the third word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_DBGMCU_EnableDBGStopMode

Function name

void HAL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name

void HAL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name

void HAL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_VREFBUF_VoltageScalingConfig

Function name

void HAL_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)

Function description

Configure the internal voltage reference buffer voltage scale.

Parameters

- **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values:
  - VREFBUF_VOLTAGE_SCALE0: VREF_OUT1 around 2.5 V. This requires VDDA equal to or higher than 2.8 V.
  - VREFBUF_VOLTAGE_SCALE1: VREF_OUT2 around 2.048 V. This requires VDDA equal to or higher than 2.4 V.
  - VREFBUF_VOLTAGE_SCALE2: VREF_OUT3 around 1.8 V. This requires VDDA equal to or higher than 2.1 V.
  - VREFBUF_VOLTAGE_SCALE3: VREF_OUT4 around 1.5 V. This requires VDDA equal to or higher than 1.8 V.

Return values

- **None:**

HAL_VREFBUF_HighImpedanceConfig

Function name

void HAL_VREFBUF_HighImpedanceConfig (uint32_t Mode)

Function description

Configure the internal voltage reference buffer high impedance mode.

Parameters

- **Mode:** specifies the high impedance mode This parameter can be one of the following values:
  - VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output.
  - VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.

Return values

- **None:**

HAL_VREFBUF_TrimmingConfig

Function name

void HAL_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)

Function description

Tune the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_EnableVREFBUF

Function name

HAL_StatusTypeDef HAL_EnableVREFBUF (void)

Function description

Enable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **HAL_OK/HAL_TIMEOUT:**

HAL_DisableVREFBUF

Function name

void HAL_DisableVREFBUF (void)

Function description

Disable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SBS_ETHInterfaceSelect

Function name

void HAL_SBS_ETHInterfaceSelect (uint32_t SBS_ETHInterface)

Function description

Ethernet PHY Interface Selection either MII or RMII.

Parameters

- **SBS_ETHInterface:** Selects the Ethernet PHY interface This parameter can be one of the following values:
  - SBS_ETH_MII : Select the Media Independent Interface
  - SBS_ETH_RMII: Select the Reduced Media Independent Interface

Return values

- **None:**

HAL_SBS_EnableVddIO1CompensationCell

Function name

void HAL_SBS_EnableVddIO1CompensationCell (void)

Function description

Enables the VDD I/Os Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_SBS_DisableVddIO1CompensationCell

Function name

void HAL_SBS_DisableVddIO1CompensationCell (void)

Function description

Power-down the VDD I/Os Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_SBS_EnableVddIO2CompensationCell

Function name

void HAL_SBS_EnableVddIO2CompensationCell (void)

Function description

Enables the VDDIO2 I/Os Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_SBS_DisableVddIO2CompensationCell

Function name

void HAL_SBS_DisableVddIO2CompensationCell (void)

Function description

Power-down the VDDIO2 I/Os Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_SBS_VDDCompensationCodeSelect

Function name

void HAL_SBS_VDDCompensationCodeSelect (uint32_t SBS_CompCode)

Function description

Code selection for the VDD I/O Compensation cell.

Parameters

- **SBS_CompCode:** Selects the code to be applied for the I/O compensation cell This parameter can be one of the following values:
  - SBS_VDD_CELL_CODE : Select Code from the cell (available in the SBS_CCVALR)
  - SBS_VDD_REGISTER_CODE: Select Code from the SBS compensation cell code register (SBS_CCSWCR)

Return values

- **None:**

HAL_SBS_VDDIOCompensationCodeSelect

Function name

void HAL_SBS_VDDIOCompensationCodeSelect (uint32_t SBS_CompCode)

Function description

Code selection for the VDDIO I/O Compensation cell.

Parameters

- **SBS_CompCode:** Selects the code to be applied for the I/O compensation cell This parameter can be one of the following values:
  - SBS_VDDIO_CELL_CODE : Select Code from the cell (available in the SBS_CCVALR)
  - SBS_VDDIO_REGISTER_CODE: Select Code from the SBS compensation cell code register (SBS_CCSWCR)

Return values

- **None:**

HAL_SBS_GetVddIO1CompensationCellReadyFlag

Function name

uint32_t HAL_SBS_GetVddIO1CompensationCellReadyFlag (void)

Function description

VDDIO1 I/O Compensation cell get ready flag status.

Return values

- **State:** of bit (1 or 0).

HAL_SBS_GetVddIO2CompensationCellReadyFlag

Function name

uint32_t HAL_SBS_GetVddIO2CompensationCellReadyFlag (void)

Function description

VDDIO2 I/O Compensation cell get ready flag status.

Return values

- **State:** of bit (1 or 0).

HAL_SBS_VDDCompensationCodeConfig

Function name

void HAL_SBS_VDDCompensationCodeConfig (uint32_t SBS_PMOSCode, uint32_t SBS_NMOSCode)

Function description

Code configuration for the VDD I/O Compensation cell.

Parameters

- **SBS_PMOSCode:** PMOS compensation code This code is applied to the VDD I/O compensation cell when the CS1 bit of the SBS_CCSR is set
- **SBS_NMOSCode:** NMOS compensation code This code is applied to the VDD I/O compensation cell when the CS1 bit of the SBS_CCSR is set

Return values

- **None:**

HAL_SBS_VDDIOCompensationCodeConfig

Function name

void HAL_SBS_VDDIOCompensationCodeConfig (uint32_t SBS_PMOSCode, uint32_t SBS_NMOSCode)

Function description

Code configuration for the VDDIO I/O Compensation cell.

Parameters

- **SBS_PMOSCode:** PMOS compensation code This code is applied to the VDDIO I/O compensation cell when the CS2 bit of the SBS_CCSR is set
- **SBS_NMOSCode:** NMOS compensation code This code is applied to the VDDIO I/O compensation cell when the CS2 bit of the SBS_CCSR is set

Return values

- **None:**

HAL_SBS_GetNMOSVddCompensationValue

Function name

uint32_t HAL_SBS_GetNMOSVddCompensationValue (void)

Function description

Get NMOS compensation value of the I/Os supplied by VDD.

Return values

- **None:**

HAL_SBS_GetPMOSVddCompensationValue

Function name

uint32_t HAL_SBS_GetPMOSVddCompensationValue (void)

Function description

Get PMOS compensation value of the I/Os supplied by VDD.

Return values

- **None:**

HAL_SBS_GetNMOSVddIO2CompensationValue

Function name

uint32_t HAL_SBS_GetNMOSVddIO2CompensationValue (void)

Function description

Get NMOS compensation value of the I/Os supplied by VDDIO2.

Return values

- **None:**

HAL_SBS_GetPMOSVddIO2CompensationValue

Function name

uint32_t HAL_SBS_GetPMOSVddIO2CompensationValue (void)

Function description

Get PMOS compensation value of the I/Os supplied by VDDIO2.

Return values

- **None:**

HAL_SBS_FLASH_EnableECCNMI

Function name

void HAL_SBS_FLASH_EnableECCNMI (void)

Function description

Enable the NMI in case of double ECC error in FLASH Interface.

Return values

- **None:**

HAL_SBS_FLASH_DisableECCNMI

Function name

void HAL_SBS_FLASH_DisableECCNMI (void)

Function description

Disable the NMI in case of double ECC error in FLASH Interface.

Return values

- **None:**

HAL_SBS_FLASH_ECCNMI_IsDisabled

Function name

uint32_t HAL_SBS_FLASH_ECCNMI_IsDisabled (void)

Function description

Check if the NMI is Enabled in case of double ECC error in FLASH Interface.

Return values

- **State:** of bit (1 or 0).

HAL_SBS_IncrementHDPLValue

Function name

void HAL_SBS_IncrementHDPLValue (void)

Function description

Increment by 1 the HDPL value.

Return values

- **None:**

HAL_SBS_GetHDPLValue

Function name

uint32_t HAL_SBS_GetHDPLValue (void)

Function description

Get the HDPL Value.

Return values

- **Returns:** the HDPL value This return value can be one of the following values:
  - SBS_HDPL_VALUE_0: HDPL0
  - SBS_HDPL_VALUE_1: HDPL1
  - SBS_HDPL_VALUE_2: HDPL2
  - SBS_HDPL_VALUE_3: HDPL3

HAL_SBS_EPOCHSelection

Function name

void HAL_SBS_EPOCHSelection (uint32_t Epoch_Selection)

Function description

Select EPOCH security sent to SAES IP to encrypt/decrypt keys.

Parameters

- **Epoch_Selection:** Select EPOCH security This parameter can be one of the following values:
  - SBS_EPOCH_SEL_SECURE : EPOCH secure selected.
  - SBS_EPOCH_SEL_NONSECURE : EPOCH non secure selected.
  - SBS_EPOCH_SEL_PUFCHECK : EPOCH all zeros for PUF integrity check.

Return values

- **None:**

HAL_SBS_GetEPOCHSelection

Function name

uint32_t HAL_SBS_GetEPOCHSelection (void)

Function description

Get EPOCH security selection.

Return values

- **Returned:** value can be one of the following values:
  - SBS_EPOCH_SEL_SECURE : EPOCH secure selected.
  - SBS_EPOCH_SEL_NONSECURE : EPOCH non secure selected.
  - SBS_EPOCH_SEL_PUFCHECK : EPOCH all zeros for PUF integrity check.

HAL_SBS_SetOBKHDPL

Function name

void HAL_SBS_SetOBKHDPL (uint32_t OBKHDPL_Value)

Function description

Set the OBK-HDPL Value.

Parameters

- **OBKHDPL_Value:** Value of the increment to add to HDPL value to generate the OBK-HDPL. This parameter can be one of the following values:
  - SBS_OBKHDPL_INCR_0 : HDPL
  - SBS_OBKHDPL_INCR_1 : HDPL + 1
  - SBS_OBKHDPL_INCR_2 : HDPL + 2
  - SBS_OBKHDPL_INCR_3 : HDPL + 3

Return values

- **None:**

HAL_SBS_GetOBKHDPL

Function name

uint32_t HAL_SBS_GetOBKHDPL (void)

Function description

Get the OBK-HDPL Value.

Return values

- **Returns:** the incremement to add to HDPL value to generate OBK-HDPL This return value can be one of the following values:
  - SBS_OBKHDPL_INCR_0: HDPL
  - SBS_OBKHDPL_INCR_1: HDPL + 1
  - SBS_OBKHDPL_INCR_2: HDPL + 2
  - SBS_OBKHDPL_INCR_3: HDPL + 3

HAL_SBS_OpenAccessPort

Function name

void HAL_SBS_OpenAccessPort (void)

Function description

Open the device access port.

Return values

- **None:**

Notes

- This function can be only used when device state is Closed.

HAL_SBS_OpenDebug

Function name

void HAL_SBS_OpenDebug (void)

Function description

Open the debug when the hide protection level is authorized.

Return values

- **None:**

Notes

- This function can be only used when device state is Closed.

HAL_SBS_ConfigDebugLevel

Function name

HAL_StatusTypeDef HAL_SBS_ConfigDebugLevel (uint32_t Level)

Function description

Configure the authenticated debug hide protection level.

Parameters

- **Level:** Hide protection level where the authenticated debug opens This value is one of HDPL Value (except SBS_HDPL_VALUE_0)

Return values

- **HAL_OK:** if parameter is correct HAL_ERROR otherwise

Notes

- This function can be only used when device state is Closed.

HAL_SBS_GetDebugLevel

Function name

uint32_t HAL_SBS_GetDebugLevel (void)

Function description

Get the current value of the hide protection level.

Return values

- **Current:** hide protection level This value is one of HDPL Value

Notes

- This function can be only used when device state is Closed.

HAL_SBS_LockDebugConfig

Function name

void HAL_SBS_LockDebugConfig (void)

Function description

Lock the access to the debug control register.

Return values

- **None:**

Notes

- This function can be only used when device state is Closed.
- locking the current debug configuration is released only by a reset.

HAL_SBS_ConfigDebugSecurity

Function name

void HAL_SBS_ConfigDebugSecurity (uint32_t Security)

Function description

Configure the authenticated debug security access.

Parameters

- **Control:** debug opening secure/non-secure or non-secure only This parameter can be one of the following values:
  - SBS_DEBUG_SEC_NSEC: debug opening for secure and non-secure.
  - SBS_DEBUG_NSEC: debug opening for non-secure only.

Return values

- **None:**

HAL_SBS_GetDebugSecurity

Function name

uint32_t HAL_SBS_GetDebugSecurity (void)

Function description

Get the current value of the hide protection level.

Return values

- **Returned:** value can be one of the following values:
  - SBS_DEBUG_SEC_NSEC: debug opening for secure and non-secure.
  - SBS_DEBUG_NSEC: debug opening for non-secure only.

Notes

- This function can be only used when device state is Closed.

HAL_SBS_Lock

Function name

void HAL_SBS_Lock (uint32_t Item)

Function description

Lock the SBS item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be a combination of SBS Lock items

Return values

- **None:**

Notes

- Setting lock(s) depends on privilege mode in secure/non-secure code Lock(s) cleared only at system reset

HAL_SBS_GetLock

Function name

HAL_StatusTypeDef HAL_SBS_GetLock (uint32_t * pItem)

Function description

Get the lock state of SBS items.

Parameters

- **pItem:** pointer to return locked items the return value can be a combination of SBS Lock items

Return values

- **HAL:** status

Notes

- Getting lock(s) depends on privilege mode in secure/non-secure code

HAL_SBS_ConfigAttributes

Function name

void HAL_SBS_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the SBS item attribute(s).

Parameters

- **Item:** Item(s) to set attributes on. This parameter can be a one or a combination of SBS Attributes items
- **Attributes:** specifies the secure/non-secure attributes.

Return values

- **None:**

Notes

- Available attributes are to secure SBS items, so this function is only available in secure. SBS_FPU item attribute is only configurable through PRIVILEGE transaction.

HAL_SBS_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_SBS_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get the attribute of a SBS items.

Parameters

- **Item:** Single item to get secure/non-secure attribute from.
- **pAttributes:** pointer to return the attribute.

Return values

- **HAL:** status

Notes

- Available attributes have read restrictions, so this function is only available in secure

6.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

6.2.1 HAL

HAL

DBGMCU Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_TIM12

__HAL_DBGMCU_UNFREEZE_TIM12

__HAL_DBGMCU_FREEZE_TIM13

__HAL_DBGMCU_UNFREEZE_TIM13

__HAL_DBGMCU_FREEZE_TIM14

__HAL_DBGMCU_UNFREEZE_TIM14

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_I2C1

__HAL_DBGMCU_UNFREEZE_I2C1

__HAL_DBGMCU_FREEZE_I2C2

__HAL_DBGMCU_UNFREEZE_I2C2

__HAL_DBGMCU_FREEZE_I3C1

__HAL_DBGMCU_UNFREEZE_I3C1

__HAL_DBGMCU_FREEZE_LPTIM2

__HAL_DBGMCU_UNFREEZE_LPTIM2

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM8

__HAL_DBGMCU_UNFREEZE_TIM8

__HAL_DBGMCU_FREEZE_TIM15

__HAL_DBGMCU_UNFREEZE_TIM15

__HAL_DBGMCU_FREEZE_TIM16

__HAL_DBGMCU_UNFREEZE_TIM16

__HAL_DBGMCU_FREEZE_TIM17

__HAL_DBGMCU_UNFREEZE_TIM17

__HAL_DBGMCU_FREEZE_I2C3

__HAL_DBGMCU_UNFREEZE_I2C3

__HAL_DBGMCU_FREEZE_I2C4

__HAL_DBGMCU_UNFREEZE_I2C4

__HAL_DBGMCU_FREEZE_LPTIM1

__HAL_DBGMCU_UNFREEZE_LPTIM1

__HAL_DBGMCU_FREEZE_LPTIM3

__HAL_DBGMCU_UNFREEZE_LPTIM3

__HAL_DBGMCU_FREEZE_LPTIM4

__HAL_DBGMCU_UNFREEZE_LPTIM4

__HAL_DBGMCU_FREEZE_LPTIM5

__HAL_DBGMCU_UNFREEZE_LPTIM5

__HAL_DBGMCU_FREEZE_LPTIM6

__HAL_DBGMCU_UNFREEZE_LPTIM6

__HAL_DBGMCU_FREEZE_RTC

__HAL_DBGMCU_UNFREEZE_RTC

__HAL_DBGMCU_FREEZE_GPDMA1_0

__HAL_DBGMCU_UNFREEZE_GPDMA1_0

__HAL_DBGMCU_FREEZE_GPDMA1_1

__HAL_DBGMCU_UNFREEZE_GPDMA1_1

__HAL_DBGMCU_FREEZE_GPDMA1_2

__HAL_DBGMCU_UNFREEZE_GPDMA1_2

__HAL_DBGMCU_FREEZE_GPDMA1_3

__HAL_DBGMCU_UNFREEZE_GPDMA1_3

__HAL_DBGMCU_FREEZE_GPDMA1_4

__HAL_DBGMCU_UNFREEZE_GPDMA1_4

__HAL_DBGMCU_FREEZE_GPDMA1_5

__HAL_DBGMCU_UNFREEZE_GPDMA1_5

__HAL_DBGMCU_FREEZE_GPDMA1_6

__HAL_DBGMCU_UNFREEZE_GPDMA1_6

__HAL_DBGMCU_FREEZE_GPDMA1_7

__HAL_DBGMCU_UNFREEZE_GPDMA1_7

__HAL_DBGMCU_FREEZE_GPDMA2_0

__HAL_DBGMCU_UNFREEZE_GPDMA2_0

__HAL_DBGMCU_FREEZE_GPDMA2_1

__HAL_DBGMCU_UNFREEZE_GPDMA2_1

__HAL_DBGMCU_FREEZE_GPDMA2_2

__HAL_DBGMCU_UNFREEZE_GPDMA2_2

__HAL_DBGMCU_FREEZE_GPDMA2_3

__HAL_DBGMCU_UNFREEZE_GPDMA2_3

__HAL_DBGMCU_FREEZE_GPDMA2_4

__HAL_DBGMCU_UNFREEZE_GPDMA2_4

__HAL_DBGMCU_FREEZE_GPDMA2_5

__HAL_DBGMCU_UNFREEZE_GPDMA2_5

__HAL_DBGMCU_FREEZE_GPDMA2_6

__HAL_DBGMCU_UNFREEZE_GPDMA2_6

__HAL_DBGMCU_FREEZE_GPDMA2_7

__HAL_DBGMCU_UNFREEZE_GPDMA2_7

HAL FDCAN Error Code

HAL_FDCAN_ERROR_NONE
: No error

HAL_FDCAN_ERROR_TIMEOUT
: Timeout error

HAL_FDCAN_ERROR_NOT_INITIALIZED
: Peripheral not initialized

HAL_FDCAN_ERROR_NOT_READY
: Peripheral not ready

HAL_FDCAN_ERROR_NOT_STARTED
: Peripheral not started

HAL_FDCAN_ERROR_NOT_SUPPORTED
: Mode not supported

HAL_FDCAN_ERROR_PARAM
: Parameter error

HAL_FDCAN_ERROR_PENDING
: Pending operation

HAL_FDCAN_ERROR_RAM_ACCESS
: Message RAM Access Failure

HAL_FDCAN_ERROR_FIFO_EMPTY
: Put element in full FIFO

HAL_FDCAN_ERROR_FIFO_FULL
: Get element from empty FIFO

HAL_FDCAN_ERROR_LOG_OVERFLOW

Overflow of CAN Error Logging Counter

HAL_FDCAN_ERROR_RAM_WDG

Message RAM Watchdog event occurred

HAL_FDCAN_ERROR_PROTOCOL_ARBT

Protocol Error in Arbitration Phase (Nominal Bit Time is used)

HAL_FDCAN_ERROR_PROTOCOL_DATA

Protocol Error in Data Phase (Data Bit Time is used)

HAL_FDCAN_ERROR_RESERVED_AREA

Access to Reserved Address

HAL_FDCAN_ERROR_INVALID_CALLBACK

Invalid Callback error

I3C Notification ID definition

EVENT_ID_GETACCCR

I3C target complete controller-role hand-off (direct GETACCR CCC) event

EVENT_ID_IBIEND

I3C target IBI end process event

EVENT_ID_DAU

I3C target receive a dynamic address update (ENTDAA/RSTDAA/SETNEWDA CCC) event

EVENT_ID_GETx

I3C target receive any direct GETxxx CCC event

EVENT_ID_GETSTATUS

I3C target receive get status command (direct GETSTATUS CCC) event

EVENT_ID_SETMWL

I3C target receive maximum write length update (direct SETMWL CCC) event

EVENT_ID_SETMRL

I3C target receive maximum read length update(direct SETMRL CCC) event

EVENT_ID_RSTACT

I3C target detect reset pattern (broadcast or direct RSTACT CCC) event

EVENT_ID_ENTASx

I3C target receive activity state update (direct or broadcast ENTASx) event

EVENT_ID_ENEC_DISEC

I3C target receive a direct or broadcast ENEC/DISEC CCC event

EVENT_ID_DEFTGTS

I3C target receive a broadcast DEFTGTS CCC event

EVENT_ID_DEFGRPA

I3C target receive a group addressing (broadcast DEFGRPA CCC) event

EVENT_ID_WKP

I3C target wakeup event

EVENT_ID_IBI

I3C controller receive IBI event

EVENT_ID_CR

I3C controller controller-role request event

EVENT_ID_HJ

I3C controller hot-join event

HAL state definition

HAL_SMBUS_STATE_RESET

SMBUS not yet initialized or disabled

HAL_SMBUS_STATE_READY

SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY

SMBUS internal process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_TX

Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX

Master Data Reception process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_TX

Slave Data Transmission process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_RX

Slave Data Reception process is ongoing

HAL_SMBUS_STATE_TIMEOUT

Timeout state

HAL_SMBUS_STATE_ERROR

Reception process is ongoing

HAL_SMBUS_STATE_LISTEN

Address Listen Mode is ongoing

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_HAL_EC_OVS_DISCONT_MODE***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_HAL_EC_OVS_SCOPE_REG***

7.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t DataAlign***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t SamplingMode***
- ***FunctionalState DMAContinuousRequests***
- ***uint32_t Overrun***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***
  Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from system clock or PLL (Refer to reference manual for list of clocks available)) and clock prescaler. This parameter can be a value of ***ADC_HAL_EC_COMMON_CLOCK_SOURCE***. Note: The ADC clock configuration is common to all ADC instances. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of asynchronous clock, the selected clock must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADC instances are disabled.
- ***uint32_t ADC_InitTypeDef::Resolution***
  Configure the ADC resolution. This parameter can be a value of ***ADC_HAL_EC_RESOLUTION***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specify ADC data alignment in conversion data register (right or left). Refer to reference manual for alignments formats versus resolutions. This parameter can be a value of ***ADC_HAL_EC_DATA_ALIGN***
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***
- ***uint32_t ADC_InitTypeDef::EOCSelection***
  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.
- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***
  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: It is not recommended to use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait). Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).
- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***
  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  Specify the number of ranks that will be converted within the regular group sequencer. This parameter is dependent on ScanConvMode:
  - sequencer configured to fully configurable: Number of ranks in the scan sequence is configurable using this parameter. Note: After the first call of '**HAL_ADC_Init()**', each rank corresponding to parameter "NbrOfConversion" must be set using '**HAL_ADC_ConfigChannel()**'. Afterwards, when all needed sequencer ranks are set, parameter 'NbrOfConversion' can be updated without modifying configuration of sequencer ranks (sequencer ranks above 'NbrOfConversion' are discarded).
  - sequencer configured to not fully configurable: Number of ranks in the scan sequence is defined by number of channels set in the sequence. This parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***
  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE. Note: On this STM32 series, ADC group regular number of discontinuous ranks increment is fixed to one-by-one.
- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  Select the external event edge used to trigger ADC group regular conversion start If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***
- ***uint32_t ADC_InitTypeDef::SamplingMode***
  Select the sampling mode to be used for ADC group regular conversion. This parameter can be a value of ***ADC_regular_sampling_mode***
- ***FunctionalState ADC_InitTypeDef::DMAContinuousRequests***
  Specify whether the DMA requests are performed in one shot mode (DMA transfer stops when number of conversions is reached) or in continuous mode (DMA transfer unlimited, whatever number of conversions). This parameter can be set to ENABLE or DISABLE. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached.
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_HAL_EC_REG_OVR_DATA_BEHAVIOR***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear) Note: Error reporting with respect to the conversion mode:
  - Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.
  - Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***FunctionalState ADC_InitTypeDef::OversamplingMode***
  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected
- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***
  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***

- ***uint32_t Offset***
- ***uint32_t OffsetSign***
- ***FunctionalState OffsetSaturation***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specify the rank in the regular group sequencer. This parameter can be a value of ***ADC_HAL_EC_REG_SEQ_RANKS*** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt, Vbat, ...), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values.
- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***
  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***
  Select the offset number This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  Define the offset to be applied on the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_ChannelConfTypeDef::OffsetSign***
  Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSaturation***
  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

7.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***

- ***uint32_t Channel***
- ***FunctionalState ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t FilteringConfig***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_HAL_EC_AWD_NUMBER***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. Channels on ADC group regular and injected are not differentiated: Set value 'ADC_ANALOGWATCHDOG_SINGLE_xxx' to monitor 1 channel, value 'ADC_ANALOGWATCHDOG_ALL_xxx' to monitor all channels, 'ADC_ANALOGWATCHDOG_NONE' to monitor no channel. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADC_HAL_EC_CHANNEL***.
- ***FunctionalState ADC_AnalogWDGConfTypeDef::ITMode***
  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::FilteringConfig***
  Specify whether filtering should be use and the number of samples to consider. Before setting flag or raising interrupt, analog watchdog can wait to have several consecutive out-of-window samples. This parameter allows to configure this number. This parameter only applies to Analog watchdog 1. For others, use value ADC_AWD_FILTERING_NONE. This parameter can be a value of ***ADC_analog_watchdog_filtering_config***.

7.1.5 ADC_InjectionConfigTypeDef

ADC_InjectionConfigTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***uint32_t ContextQueue***
- ***uint32_t ChannelCount***

Field Documentation

- ***uint32_t ADC_InjectionConfigTypeDef::ContextQueue***
  Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call
- ***uint32_t ADC_InjectionConfigTypeDef::ChannelCount***
  Number of channels in the injected sequence

7.1.6 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32h5xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***ADC_InjectionConfigTypeDef InjectionConfig***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* InjectedConvCpltCallback***
- ***void(* InjectedQueueOverflowCallback***
- ***void(* LevelOutOfWindow2Callback***
- ***void(* LevelOutOfWindow3Callback***
- ***void(* EndOfSamplingCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***ADC_TypeDef* __ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef __ADC_HandleTypeDef::Init***
  ADC initialization parameters and regular conversions setting
- ***DMA_HandleTypeDef* __ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef __ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t __ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t __ADC_HandleTypeDef::ErrorCode***
  ADC Error code
- ***ADC_InjectionConfigTypeDef __ADC_HandleTypeDef::InjectionConfig***
  ADC injected channel configuration build-up structure
- ***void(* __ADC_HandleTypeDef::ConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion complete callback
- ***void(* __ADC_HandleTypeDef::ConvHalfCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion DMA half-transfer callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 1 callback
- ***void(* __ADC_HandleTypeDef::ErrorCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC error callback
- ***void(* __ADC_HandleTypeDef::InjectedConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected conversion complete callback

- ***void(* __ADC_HandleTypeDef::InjectedQueueOverflowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected context queue overflow callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow2Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 2 callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow3Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 3 callback
- ***void(* __ADC_HandleTypeDef::EndOfSamplingCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC end of sampling callback
- ***void(* __ADC_HandleTypeDef::MspInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp Init callback
- ***void(* __ADC_HandleTypeDef::MspDeInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp DeInit callback

7.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

7.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset shared on 4 offset instances.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

7.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock or PLL.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLL;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().

2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   - ADC conversion by polling:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     - Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   - ADC conversion by interruption:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   - ADC conversion with transfer by DMA:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     - Conversion results are automatically transferred by DMA into destination variable address.
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Note: Callback functions must be implemented in user program:

- *HAL_ADC_ErrorCallback()*
- *HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)*
- *HAL_ADC_ConvCpltCallback()*
- *HAL_ADC_ConvHalfCpltCallback*

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
     - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
     - HAL_RCC_OscConfig(&RCC_OscInitStructure);
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

Callback registration

The compilation flag USE_HAL_ADC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_ADC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_ADC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback

- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_ADC_UnRegisterCallback to reset a callback to the default weak function.

@ref HAL_ADC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback

By default, after the @ref HAL_ADC_Init() and when the state is @ref HAL_ADC_STATE_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL_ADC_ConvCpltCallback(), @ref HAL_ADC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_ADC_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in @ref HAL_ADC_STATE_READY or @ref HAL_ADC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_ADC_RegisterCallback() before calling @ref HAL_ADC_DeInit() or @ref HAL_ADC_Init() function.

When the compilation flag USE_HAL_ADC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

7.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- ***HAL_ADC_ConfigChannel()***
- ***HAL_ADC_AnalogWDGConfig()***

7.2.4 Peripheral state and errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- *HAL_ADC_GetState()*
- *HAL_ADC_GetError()*

7.2.5 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- As prerequisite, ADC clock must be configured at RCC top level (refer to description of RCC configuration for ADC in header of this file).
- Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".
- Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DeInit

Function name

HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

Function description

Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. (function "HAL_ADC_MspDeInit()" is also called under the same conditions: all ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances). If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running.
- By default, HAL_ADC_DeInit() set ADC in mode deep power-down: this saves more power by reducing leakage currents and is particularly interesting before entering MCU low-power modes.

HAL_ADC_MspInit

Function name

void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_MspDeInit

Function name

void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)

Function description

DeInitialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- All ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances).

HAL_ADC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_RegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID, pADC_CallbackTypeDef pCallback)

Function description

Register a User ADC Callback To be used instead of the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  - HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  - HAL_ADC_ERROR_CB_ID ADC error callback ID
  - HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  - HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  - HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  - HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  - HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID
  - HAL_ADC_MSPINIT_CB_ID MspInit callback ID
  - HAL_ADC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_ADC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_UnRegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID)

Function description

Unregister a ADC Callback ADC callback is redirected to the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  - HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  - HAL_ADC_ERROR_CB_ID ADC error callback ID
  - HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  - HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  - HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  - HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  - HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID
  - HAL_ADC_MSPINIT_CB_ID MspInit callback ID
  - HAL_ADC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_ADC_Start

Function name

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled (when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

Poll for ADC event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
  - ADC_EOSMP_EVENT ADC End of Sampling event
  - ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)
  - ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)
  - ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)
  - ADC_OVR_EVENT ADC Overrun event
  - ADC_JQOVF_EVENT ADC Injected context queue overflow event
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group with interruption.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function according to initialization setting : EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_IT() must be called for ADC Slave first, then for ADC Master. For ADC Slave, ADC is enabled only (conversion is not started). For ADC Master, ADC is enabled and multimode conversion is started.
- To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions.
- By default, HAL_ADC_Start_IT() does not enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start conversion of regular group and transfer result through DMA.

Parameters

- **hadc:** ADC handle
- **pData:** Destination Buffer address.
- **Length:** Number of data to be transferred from ADC peripheral to memory

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on ADC group injected. If ADC group injected is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (const ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** group regular conversion data

Notes

- Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).
- This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_StartSampling

Function name

HAL_StatusTypeDef HAL_ADC_StartSampling (ADC_HandleTypeDef * hadc)

Function description

Start ADC conversion sampling phase of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : This function should only be called to start sampling when ADC_SAMPLING_MODE_TRIGGER_CONTROLED sampling mode has been selectedADC_SOFTWARE_START has been selected as trigger source

HAL_ADC_StopSampling

Function name

HAL_StatusTypeDef HAL_ADC_StopSampling (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion sampling phase of regular group and start conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : This function should only be called to stop sampling when ADC_SAMPLING_MODE_TRIGGER_CONTROLED sampling mode has been selectedADC_SOFTWARE_START has been selected as trigger sourceafter sampling has been started using HAL_ADC_StartSampling.

HAL_ADC_IRQHandler

Function name

void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

Function description

Handle ADC interrupt request.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvCpltCallback

Function name

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvHalfCpltCallback

Function name

void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion DMA half-transfer callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name

void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 1 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ErrorCallback

Function name

void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)

Function description

ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle parameter "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, const ADC_ChannelConfTypeDef * pConfig)

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

- **hadc:** ADC handle
- **pConfig:** Structure of ADC channel assigned to ADC group regular.

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, const ADC_AnalogWDGConfTypeDef * pAnalogWDGConfig)

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **pAnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".
- On this STM32 series, analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

HAL_ADC_GetState

Function name

uint32_t HAL_ADC_GetState (const ADC_HandleTypeDef * hadc)

Function description

Return the ADC handle state.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** handle state (bitfield on 32 bits)

Notes

- ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) "

HAL_ADC_GetError

Function name

uint32_t HAL_ADC_GetError (const ADC_HandleTypeDef * hadc)

Function description

Return the ADC error code.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop

Function name

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle
- **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values:
  - ADC_REGULAR_GROUP ADC regular conversion type.
  - ADC_INJECTED_GROUP ADC injected conversion type.
  - ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.

Return values

- **HAL:** status.

ADC_Enable

Function name

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

Enable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable

Function name

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

Disable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name

void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAHalfConvCplt

Function name

void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAError

Function name

void ADC_DMAError (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

7.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

7.3.1 ADC

ADC

ADC analog watchdog (AWD) filtering configuration

ADC_AWD_FILTERING_NONE

ADC AWD no filtering, one out-of-window sample to raise flag or interrupt

ADC_AWD_FILTERING_2SAMPLES

ADC AWD 2 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_3SAMPLES

ADC AWD 3 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_4SAMPLES

ADC AWD 4 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_5SAMPLES

ADC AWD 5 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_6SAMPLES

ADC AWD 6 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_7SAMPLES

ADC AWD 7 consecutives out-of-window samples to raise flag or interrupt

ADC_AWD_FILTERING_8SAMPLES

ADC AWD 8 consecutives out-of-window samples to raise flag or interrupt

ADC analog watchdog (AWD) mode

ADC_ANALOGWATCHDOG_NONE

ADC AWD not selected

ADC_ANALOGWATCHDOG_SINGLE_REG

ADC AWD applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

ADC AWD applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

ADC AWD applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

ADC AWD applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

ADC AWD applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

ADC AWD applied to regular and injected groups all channels

ADCx CFGR fields

ADC_CFGR_FIELDS

ADCx CFGR sub fields

ADC_CFGR_FIELDS_2

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC peripheral internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_JQOVF

Injected context queue overflow error

HAL_ADC_ERROR_INVALID_CALLBACK

Invalid Callback error

ADC Event type

ADC_EOSMP_EVENT

Notes:

- Analog watchdog 1 is available on all stm32 series Analog watchdog 2 and 3 are not available on all series ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog)

ADC_OVR_EVENT

ADC overrun event

ADC_JQOVF_EVENT

ADC Injected Context Queue Overflow event

ADC Exported Constants

ADC_AWD_EVENT

ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC flags definition

ADC_FLAG_RDY

ADC Ready flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

ADC_FLAG_JQOVF

ADC Injected Context Queue Overflow flag

Analog watchdog - ADC analog watchdog (AWD) number

ADC_ANALOGWATCHDOG_1

ADC analog watchdog number 1

ADC_ANALOGWATCHDOG_2

ADC analog watchdog number 2

ADC_ANALOGWATCHDOG_3

ADC analog watchdog number 3

ADC instance - Channel number

ADC_CHANNEL_0

External channel (GPIO pin) ADCx_IN0

ADC_CHANNEL_1

External channel (GPIO pin) ADCx_IN1

ADC_CHANNEL_2

External channel (GPIO pin) ADCx_IN2

ADC_CHANNEL_3

External channel (GPIO pin) ADCx_IN3

ADC_CHANNEL_4

External channel (GPIO pin) ADCx_IN4

ADC_CHANNEL_5

External channel (GPIO pin) ADCx_IN5

ADC_CHANNEL_6

External channel (GPIO pin) ADCx_IN6

ADC_CHANNEL_7

External channel (GPIO pin) ADCx_IN7

ADC_CHANNEL_8

External channel (GPIO pin) ADCx_IN8

ADC_CHANNEL_9

External channel (GPIO pin) ADCx_IN9

ADC_CHANNEL_10

External channel (GPIO pin) ADCx_IN10

ADC_CHANNEL_11

External channel (GPIO pin) ADCx_IN11

ADC_CHANNEL_12

External channel (GPIO pin) ADCx_IN12

ADC_CHANNEL_13

External channel (GPIO pin) ADCx_IN13

ADC_CHANNEL_14

External channel (GPIO pin) ADCx_IN14

ADC_CHANNEL_15

External channel (GPIO pin) ADCx_IN15

ADC_CHANNEL_16

External channel (GPIO pin) ADCx_IN16

ADC_CHANNEL_17

External channel (GPIO pin) ADCx_IN17

ADC_CHANNEL_18

External channel (GPIO pin) ADCx_IN18

ADC_CHANNEL_19

External channel (GPIO pin) ADCx_IN19

ADC_CHANNEL_VREFINT

Internal channel VrefInt: Internal voltage reference, channel specific to ADC1.

ADC_CHANNEL_TEMPSENSOR

Internal channel Temperature sensor, channel specific to ADC1.

ADC_CHANNEL_VBAT

Internal channel Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda, channel specific to ADC2.

ADC_CHANNEL_VDDCORE

Internal channel Vddcore, channel specific to ADC2.

Channel - Sampling time

ADC_SAMPLETIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

ADC_SAMPLETIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

ADC_SAMPLETIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

ADC_SAMPLETIME_24CYCLES_5

Sampling time 24.5 ADC clock cycles

ADC_SAMPLETIME_47CYCLES_5

Sampling time 47.5 ADC clock cycles

ADC_SAMPLETIME_92CYCLES_5

Sampling time 92.5 ADC clock cycles

ADC_SAMPLETIME_247CYCLES_5

Sampling time 247.5 ADC clock cycles

ADC_SAMPLETIME_640CYCLES_5

Sampling time 640.5 ADC clock cycles

ADC_SAMPLETIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles. If selected, this sampling time replaces sampling time 2.5 ADC clock cycles. These 2 sampling times cannot be used simultaneously.

Channel - Single or differential ending

ADC_SINGLE_ENDED

ADC channel ending set to single ended

ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential

ADC common - Clock source

ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock from AHB clock without prescaler

ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock from AHB clock with prescaler division by 2

ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock from AHB clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC conversion data alignment

ADC_DATAALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

ADC_DATAALIGN_LEFT

ADC conversion data alignment: left aligned (alignment on data register MSB bit 15)

ADC instance - Groups

ADC_REGULAR_GROUP

ADC group regular (available on all STM32 devices)

ADC_INJECTED_GROUP

ADC group injected (not available on all STM32 devices)

ADC_REGULAR_INJECTED_GROUP

ADC both groups regular and injected

Multimode - DMA transfer mode depending on ADC resolution

ADC_DMAACCESSMODE_DISABLED

DMA multimode disabled: each ADC uses its own DMA channel

ADC_DMAACCESSMODE_12_10_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 12 and 10 bits resolution

ADC_DMAACCESSMODE_8_6_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 8 and 6 bits resolution

Multimode - Mode

ADC_MODE_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC_DUALMODE_REGSIMULT

ADC dual mode enabled: group regular simultaneous

ADC_DUALMODE_INTERL

ADC dual mode enabled: Combined group regular interleaved

ADC_DUALMODE_INJECSIMULT

ADC dual mode enabled: group injected simultaneous

ADC_DUALMODE_ALTERTRIG

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

ADC_TWOSAMPLINGDELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

ADC_TWOSAMPLINGDELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

ADC_TWOSAMPLINGDELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC instance - Offset number

ADC_OFFSET_NONE

ADC offset disabled: no offset correction for the selected ADC channel

ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

ADC oversampling discontinuous mode: continuous mode (all conversions of OVS ratio are done from 1 trigger)

ADC_TRIGGEREDMODE_MULTI_TRIGGER

ADC oversampling discontinuous mode: discontinuous mode (each conversion of OVS ratio needs a trigger)

Oversampling - Ratio

ADC_OVERSAMPLING_RATIO_2

Notes:

- The oversampling ratio is the number of ADC conversions performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift) ADC oversampling ratio 2

ADC_OVERSAMPLING_RATIO_4

ADC oversampling ratio 4

ADC_OVERSAMPLING_RATIO_8

ADC oversampling ratio 8

ADC_OVERSAMPLING_RATIO_16

ADC oversampling ratio 16

ADC_OVERSAMPLING_RATIO_32

ADC oversampling ratio 32

ADC_OVERSAMPLING_RATIO_64

ADC oversampling ratio 64

ADC_OVERSAMPLING_RATIO_128

ADC oversampling ratio 128

ADC_OVERSAMPLING_RATIO_256

ADC oversampling ratio 256

Oversampling - Oversampling scope for ADC group regular

ADC_REGOVERSAMPLING_CONTINUED_MODE

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

Notes:

- The sum of the ADC conversions data is divided by "Rightbitshift" number to result as the ADC oversampling conversion data) ADC oversampling no shift

ADC_RIGHTBITSHIFT_1

ADC oversampling right shift of 1 ranks

ADC_RIGHTBITSHIFT_2

ADC oversampling right shift of 2 ranks

ADC_RIGHTBITSHIFT_3

ADC oversampling right shift of 3 ranks

ADC_RIGHTBITSHIFT_4

ADC oversampling right shift of 4 ranks

ADC_RIGHTBITSHIFT_5

ADC oversampling right shift of 5 ranks

ADC_RIGHTBITSHIFT_6

ADC oversampling right shift of 6 ranks

ADC_RIGHTBITSHIFT_7

ADC oversampling right shift of 7 ranks

ADC_RIGHTBITSHIFT_8

ADC oversampling right shift of 8 ranks

ADC group regular - Overrun behavior on conversion data

ADC_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

ADC_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer ranks

ADC_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC_REGULAR_RANK_9

ADC group regular sequencer rank 9

ADC_REGULAR_RANK_10

ADC group regular sequencer rank 10

ADC_REGULAR_RANK_11

ADC group regular sequencer rank 11

ADC_REGULAR_RANK_12

ADC group regular sequencer rank 12

ADC_REGULAR_RANK_13

ADC group regular sequencer rank 13

ADC_REGULAR_RANK_14

ADC group regular sequencer rank 14

ADC_REGULAR_RANK_15

ADC group regular sequencer rank 15

ADC_REGULAR_RANK_16

ADC group regular sequencer rank 16

ADC instance - Resolution

ADC_RESOLUTION_12B

ADC resolution 12 bits

ADC_RESOLUTION_10B

ADC resolution 10 bits

ADC_RESOLUTION_8B

ADC resolution 8 bits

ADC_RESOLUTION_6B

ADC resolution 6 bits

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag.

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag.

Return value:

- None

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)(4)
  - ADC_CHANNEL_TEMPSENSOR (1)(4)
  - ADC_CHANNEL_VBAT (2)(4)
  - ADC_CHANNEL_VDDCORE (2)(4)

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_19
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VDDCORE (2)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On this STM32 series, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On this STM32 series, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

Injected conversions trigger disabled (SW start)

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

Injected conversions trigger polarity set to rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions trigger polarity set to falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions trigger polarity set to both rising and falling edges

ADC group injected trigger source

ADC_INJECTED_SOFTWARE_START

ADC group injected conversion trigger software start

ADC_EXTERNALTRIGINJEC_T1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event.

ADC_EXTERNALTRIGINJEC_T1_CC4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_T2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event.

ADC_EXTERNALTRIGINJEC_T2_CC1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare).

ADC_EXTERNALTRIGINJEC_T3_CC4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_EXT_IT15

ADC group injected conversion trigger from external peripheral: external interrupt line 15.

ADC_EXTERNALTRIGINJEC_T1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event.

ADC_EXTERNALTRIGINJEC_T3_CC3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare).

ADC_EXTERNALTRIGINJEC_T3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event.

ADC_EXTERNALTRIGINJEC_T3_CC1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare).

ADC_EXTERNALTRIGINJEC_T6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event.

ADC_EXTERNALTRIGINJEC_LPTIM1_CH1

ADC group injected conversion trigger from external peripheral: LPTIM1 channel 1 event.

ADC_EXTERNALTRIGINJEC_LPTIM2_CH1

ADC group injected conversion trigger from external peripheral: LPTIM2 channel 1 event.

ADC_EXTERNALTRIGINJEC_T4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIGINJEC_T8_CC4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare).

ADC_EXTERNALTRIGINJEC_T8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIGINJEC_T8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIGINJEC_T15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Specific to devices STM32H563/H573xx.

ADC group injected - Sequencer ranks

ADC_INJECTED_RANK_1

ADC group injected sequencer rank 1

ADC_INJECTED_RANK_2

ADC group injected sequencer rank 2

ADC_INJECTED_RANK_3

ADC group injected sequencer rank 3

ADC_INJECTED_RANK_4

ADC group injected sequencer rank 4

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_EOSMP

ADC End of sampling interrupt source

ADC_IT_EOC

ADC End of regular conversion interrupt source

ADC_IT_EOS

ADC End of regular sequence of conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of injected conversion interrupt source

ADC_IT_JEOS

ADC End of injected sequence of conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC_IT_JQOVF

ADC Injected Context Queue Overflow interrupt source

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC group regular trigger disabled (SW start)

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC group regular conversion trigger polarity set to rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC group regular conversion trigger polarity set to falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular trigger source

ADC_SOFTWARE_START

ADC group regular conversion trigger software start

ADC_EXTERNALTRIG_T1_CC1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare).

ADC_EXTERNALTRIG_T1_CC2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare).

ADC_EXTERNALTRIG_T1_CC3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare).

ADC_EXTERNALTRIG_T2_CC2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare).

ADC_EXTERNALTRIG_T3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event.

ADC_EXTERNALTRIG_EXT_IT11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event.

ADC_EXTERNALTRIG_T1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event.

ADC_EXTERNALTRIG_T1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event.

ADC_EXTERNALTRIG_T2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event.

ADC_EXTERNALTRIG_T6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event.

ADC_EXTERNALTRIG_T3_CC4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare).

ADC_EXTERNALTRIG_EXT_IT15

ADC group regular conversion trigger from external peripheral: external interrupt line 15 event.

ADC_EXTERNALTRIG_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM1 channel 1 event.

ADC_EXTERNALTRIG_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 channel 1 event.

ADC_EXTERNALTRIG_T4_CC4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare). Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIG_T8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIG_T8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIG_T4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Specific to devices STM32H563/H573xx.

ADC_EXTERNALTRIG_T15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Specific to devices STM32H563/H573xx.

ADC group regular sampling mode

ADC_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

ADC_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

ADC_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling Trigger falling edge = stop sampling and start conversion

ADC sequencer scan mode

ADC_SCAN_DISABLE

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration, ...)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 series: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_INJ_JQOVF

Injected queue overflow occurrence

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver registers structures

8.1.1 ADC_InjOversamplingTypeDef

ADC_InjOversamplingTypeDef is defined in the stm32h5xx_hal_adc_ex.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio. This parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***

8.1.2 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32h5xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedOffsetSign***
- ***FunctionalState InjectedOffsetSaturation***
- ***uint32_t InjectedNbrOfConversion***
- ***FunctionalState InjectedDiscontinuousConvMode***
- ***FunctionalState AutoInjectedConv***
- ***FunctionalState QueueInjectedContext***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***FunctionalState InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADC_INJ_SEQ_RANKS***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt, ...), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***
  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***
  Selects the offset number. This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  Defines the offset to be applied on the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetSign***
  Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSaturation***
  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
  Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
  Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv***
  Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::QueueInjectedContext***
  Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function 'HAL_ADCEx_InjectedQueueOverflowCallback'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with as many calls to **HAL_ADCEx_InjectedConfigChannel()** as the 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
  Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***
  Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***
  Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

8.1.3 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32h5xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DMAAccessMode***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***
  Configures the ADC to operate in independent or multimode. This parameter can be a value of ***ADC_HAL_EC_MULTI_MODE***.
- ***uint32_t ADC_MultiModeTypeDef::DMAAccessMode***
  Configures the DMA mode for multimode ADC: selection whether 2 DMA channels (each ADC uses its own DMA channel) or 1 DMA channel (one DMA channel for both ADC, DMA of ADC master). This parameter can be a value of ***ADC_HAL_EC_MULTI_DMA_TRANSFER_RESOLUTION***.

- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***
  Configures the Delay between 2 sampling phases. This parameter can be a value of ***ADC_HAL_EC_MULTI_TWOSMP_DELAY***. Delay range depends on selected resolution: from 1 to 12 clock cycles for 12 bits, from 1 to 10 clock cycles for 10 bits, from 1 to 8 clock cycles for 8 bits, from 1 to 6 clock cycles for 6 bits.

8.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

8.2.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.
- Start conversion of ADC group injected.
- Stop conversion of ADC group injected.
- Poll for conversion complete on ADC group injected.
- Get result of ADC group injected channel conversion.
- Start conversion of ADC group injected and enable interruptions.
- Stop conversion of ADC group injected and disable interruptions.
- When multimode feature is available, start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.

This section contains the following APIs:

- ***HAL_ADCEx_Calibration_Start()***
- ***HAL_ADCEx_Calibration_GetValue()***
- ***HAL_ADCEx_Calibration_SetValue()***
- ***HAL_ADCEx_InjectedStart()***
- ***HAL_ADCEx_InjectedStop()***
- ***HAL_ADCEx_InjectedPollForConversion()***
- ***HAL_ADCEx_InjectedStart_IT()***
- ***HAL_ADCEx_InjectedStop_IT()***
- ***HAL_ADCEx_MultiModeStart_DMA()***
- ***HAL_ADCEx_MultiModeStop_DMA()***
- ***HAL_ADCEx_MultiModeGetValue()***
- ***HAL_ADCEx_InjectedGetValue()***
- ***HAL_ADCEx_InjectedConvCpltCallback()***
- ***HAL_ADCEx_InjectedQueueOverflowCallback()***
- ***HAL_ADCEx_LevelOutOfWindow2Callback()***
- ***HAL_ADCEx_LevelOutOfWindow3Callback()***
- ***HAL_ADCEx_EndOfSamplingCallback()***
- ***HAL_ADCEx_RegularStop()***
- ***HAL_ADCEx_RegularStop_IT()***
- ***HAL_ADCEx_RegularStop_DMA()***
- ***HAL_ADCEx_RegularMultiModeStop_DMA()***

8.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode when multimode feature is available

- Enable or Disable Injected Queue
- Disable ADC voltage regulator
- Enter ADC deep-power-down mode

This section contains the following APIs:

- ***HAL_ADCEx_InjectedConfigChannel()***
- ***HAL_ADCEx_MultiModeConfigChannel()***
- ***HAL_ADCEx_EnableInjectedQueue()***
- ***HAL_ADCEx_DisableInjectedQueue()***
- ***HAL_ADCEx_DisableVoltageRegulator()***
- ***HAL_ADCEx_EnterADCDeepPowerDownMode()***

8.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).

Parameters

- **hadc:** ADC handle
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **HAL:** status

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (const ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Get the calibration factor.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **Calibration:** value.

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set the calibration factor to overwrite automatic conversion result.

Parameters

- **hadc:** ADC handle
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
- **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)

Return values

- **HAL:** state

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled when multimode feature is available: HAL_ADCEx_InjectedStart() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for injected group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group with interruption.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function according to initialization setting : JEOC (end of conversion) or JEOS (end of sequence)
- Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels, disable interruption of end-of-conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).
- In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable Multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_MultiModeGetValue

Function name

uint32_t HAL_ADCEx_MultiModeGetValue (const ADC_HandleTypeDef * hadc)

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_InjectedGetValue

Function name

uint32_t HAL_ADCEx_InjectedGetValue (const ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

Get ADC injected group conversion result.

Parameters

- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
  - ADC_INJECTED_RANK_1 ADC group injected rank 1
  - ADC_INJECTED_RANK_2 ADC group injected rank 2
  - ADC_INJECTED_RANK_3 ADC group injected rank 3
  - ADC_INJECTED_RANK_4 ADC group injected rank 4

Return values

- **ADC:** group injected conversion data

Notes

- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 series). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name

void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Injected conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name

void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)

Function description

Injected context queue overflow callback.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 2 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 3 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

End Of Sampling callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of ADC groups regular and injected, disable interrution of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop DMA-based multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral if no injected conversion is on-going.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. Multimode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADCEx_RegularStop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, const ADC_InjectionConfTypeDef * pConfigInjected)

Function description

Configure a channel to be assigned to ADC group injected.

Parameters

- **hadc:** ADC handle
- **pConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_InjectionConfTypeDef".
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Caution: For Injected Context Queue use, a context must be fully defined before start of injected conversion. All channels are configured consecutively for the same ADC instance. Therefore, the number of calls to HAL_ADCEx_InjectedConfigChannel() must be equal to the value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context is intended to be used (or if there is no use of the Injected Queue Context feature) and if the context contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (i.e. 3 times) before starting a conversion. This function must not be called to configure a 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts are intended to be used and each of them contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_MultiModeConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, const ADC_MultiModeTypeDef * pMultimode)

Function description

Enable ADC multimode and configure multimode parameters.

Parameters

- **hadc:** Master ADC handle
- **pMultimode:** Structure of ADC multimode configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To move back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

HAL_ADCEx_EnableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_EnableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Enable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function resets CFGR register JQDIS bit in order to enable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Disable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function sets CFGR register JQDIS bit in order to disable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableVoltageRegulator

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)

Function description

Disable ADC voltage regulator.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled.
- To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

Enter ADC deep-power-down mode.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering stop modes.
- Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost.
- To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

8.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 ADCEx

ADCEx

ADC Extended Exported Macros

ADC_FORCE_MODE_INDEPENDENT

Description:

- Force ADC instance in multimode mode independent (multimode disable).

Parameters:

- __HANDLE__: ADC handle.

Return value:

- None

Notes:

- This macro must be used only in case of transition from multimode to mode independent and in case of unknown previous state, to ensure ADC configuration is in mode independent. Standard way of multimode configuration change is done from HAL ADC handle of ADC master using function "HAL_ADCEx_MultiModeConfigChannel(..., ADC_MODE_INDEPENDENT)")". Usage of this macro is not the Standard way of multimode configuration and can lead to have HAL ADC handles status misaligned. Usage of this macro must be limited to cases mentioned above.

ADC Extended Offset Sign

ADC_OFFSET_SIGN_NEGATIVE

Offset sign negative, offset is subtracted

ADC_OFFSET_SIGN_POSITIVE

Offset sign positive, offset is added

9 HAL CEC Generic Driver

9.1 CEC Firmware driver registers structures

9.1.1 CEC_InitTypeDef

CEC_InitTypeDef is defined in the stm32h5xx_hal_cec.h

Data Fields

- ***uint32_t SignalFreeTime***
- ***uint32_t Tolerance***
- ***uint32_t BRERxStop***
- ***uint32_t BREErrorBitGen***
- ***uint32_t LBPEErrorBitGen***
- ***uint32_t BroadcastMsgNoErrorBitGen***
- ***uint32_t SignalFreeTimeOption***
- ***uint32_t ListenMode***
- ***uint16_t OwnAddress***
- ***uint8_t * RxBuffer***

Field Documentation

- ***uint32_t CEC_InitTypeDef::SignalFreeTime***
  Set SFT field, specifies the Signal Free Time. It can be one of CEC_Signal_Free_Time and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods
- ***uint32_t CEC_InitTypeDef::Tolerance***
  Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of CEC_Tolerance : it is either CEC_STANDARD_TOLERANCE or CEC_EXTENDED_TOLERANCE
- ***uint32_t CEC_InitTypeDef::BRERxStop***
  Set BRESTP bit CEC_BRERxStop : specifies whether or not a Bit Rising Error stops the reception. CEC_NO_RX_STOP_ON_BRE: reception is not stopped. CEC_RX_STOP_ON_BRE: reception is stopped.
- ***uint32_t CEC_InitTypeDef::BREErrorBitGen***
  Set BREGEN bit CEC_BREErrorBitGen : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection. CEC_BRE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_BRE_ERRORBIT_GENERATION: error-bit generation if BRESTP is set.
- ***uint32_t CEC_InitTypeDef::LBPEErrorBitGen***
  Set LBPEGEN bit CEC_LBPEErrorBitGen : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection. CEC_LBPE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_LBPE_ERRORBIT_GENERATION: error-bit generation.
- ***uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen***
  Set BRDNOGEN bit CEC_BroadCastMsgErrorBitGen : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message.It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:1) CEC_BROADCASTERROR_ERRORBIT_GENERATION. a) BRE detection: error-bit generation on the CEC line if BRESTP=CEC_RX_STOP_ON_BRE and BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION. b) LBPE detection: error-bit generation on the CEC line if LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION.2) CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.
- ***uint32_t CEC_InitTypeDef::SignalFreeTimeOption***
  Set SFTOP bit CEC_SFT_Option : specifies when SFT timer starts. CEC_SFT_START_ON_TXSOM SFT: timer starts when TXSOM is set by software. CEC_SFT_START_ON_TX_RX_END: SFT timer starts automatically at the end of message transmission/reception.

- ***uint32_t CEC_InitTypeDef::ListenMode***
  Set LSTN bit CEC_Listening_Mode : specifies device listening mode. It can take two values:CEC_REDUCED_LISTENING_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.CEC_FULL_LISTENING_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- ***uint16_t CEC_InitTypeDef::OwnAddress***
  Own addresses configuration This parameter can be a value of CEC_OWN_ADDRESS
- ***uint8_t* CEC_InitTypeDef::RxBuffer***
  CEC Rx buffer pointer

9.1.2 __CEC_HandleTypeDef

__CEC_HandleTypeDef is defined in the stm32h5xx_hal_cec.h

Data Fields

- ***CEC_TypeDef * Instance***
- ***CEC_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferCount***
- ***uint16_t RxXferSize***
- ***HAL_LockTypeDef Lock***
- ***HAL_CEC_StateTypeDef gState***
- ***HAL_CEC_StateTypeDef RxState***
- ***uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CEC_TypeDef* __CEC_HandleTypeDef::Instance***
  CEC registers base address
- ***CEC_InitTypeDef __CEC_HandleTypeDef::Init***
  CEC communication parameters
- ***const uint8_t* __CEC_HandleTypeDef::pTxBuffPtr***
  Pointer to CEC Tx transfer Buffer
- ***uint16_t __CEC_HandleTypeDef::TxXferCount***
  CEC Tx Transfer Counter
- ***uint16_t __CEC_HandleTypeDef::RxXferSize***
  CEC Rx Transfer size, 0: header received only
- ***HAL_LockTypeDef __CEC_HandleTypeDef::Lock***
  Locking object
- ***HAL_CEC_StateTypeDef __CEC_HandleTypeDef::gState***
  CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of HAL_CEC_StateTypeDef
- ***HAL_CEC_StateTypeDef __CEC_HandleTypeDef::RxState***
  CEC state information related to Rx operations. This parameter can be a value of HAL_CEC_StateTypeDef
- ***uint32_t __CEC_HandleTypeDef::ErrorCode***
  For errors handling purposes, copy of ISR register in case error is reported
- ***void(* __CEC_HandleTypeDef::TxCpltCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Tx Transfer completed callback
- ***void(* __CEC_HandleTypeDef::RxCpltCallback)(struct __CEC_HandleTypeDef *hcec, uint32_t RxFrameSize)***
  CEC Rx Transfer completed callback

- ***void(* __CEC_HandleTypeDef::ErrorCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC error callback
- ***void(* __CEC_HandleTypeDef::MspInitCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Msp Init callback
- ***void(* __CEC_HandleTypeDef::MspDeInitCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Msp DeInit callback

9.2 CEC Firmware driver API description

The following section lists the various functions of the CEC library.

9.2.1 How to use this driver

The CEC HAL driver can be used as follow:

1. Declare a CEC_HandleTypeDef handle structure.
2. Initialize the CEC low level resources by implementing the HAL_CEC_MspInit ()API:
   a. Enable the CEC interface clock.
   b. CEC pins configuration:
      ◦ Enable the clock for the CEC GPIOs.
      ◦ Configure these CEC pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_CEC_Transmit_IT() and HAL_CEC_Receive_IT() APIs):
      ◦ Configure the CEC interrupt priority.
      ◦ Enable the NVIC CEC IRQ handle.
      ◦ The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_CEC_ENABLE_IT() and __HAL_CEC_DISABLE_IT() inside the transmit and receive process.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL_CEC_Init() API.

Note: This API (HAL_CEC_Init()) configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_CEC_MspInit() API.

Callback registration

9.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the CEC

- The following parameters need to be configured:
  – SignalFreeTime
  – Tolerance
  – BRERxStop (RX stopped or not upon Bit Rising Error)
  – BREErrorBitGen (Error-Bit generation in case of Bit Rising Error)
  – LBPEErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
  – BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
  – SignalFreeTimeOption (SFT Timer start definition)
  – OwnAddress (CEC device address)
  – ListenMode

This section contains the following APIs:

- ***HAL_CEC_Init()***
- ***HAL_CEC_DeInit()***
- ***HAL_CEC_SetDeviceAddress()***
- ***HAL_CEC_MspInit()***
- ***HAL_CEC_MspDeInit()***

- *HAL_CEC_RegisterCallback()*
- *HAL_CEC_UnRegisterCallback()*
- *HAL_CEC_RegisterRxCpltCallback()*
- *HAL_CEC_UnRegisterRxCpltCallback()*

9.2.3 IO operation functions

This section contains the following APIs:

- *HAL_CEC_Transmit_IT()*
- *HAL_CEC_GetLastReceivedFrameSize()*
- *HAL_CEC_ChangeRxBuffer()*
- *HAL_CEC_IRQHandler()*
- *HAL_CEC_TxCpltCallback()*
- *HAL_CEC_RxCpltCallback()*
- *HAL_CEC_ErrorCallback()*

9.2.4 Peripheral Control function

This subsection provides a set of functions allowing to control the CEC.

- HAL_CEC_GetState() API can be helpful to check in run-time the state of the CEC peripheral.
- HAL_CEC_GetError() API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- *HAL_CEC_GetState()*
- *HAL_CEC_GetError()*

9.2.5 Detailed description of functions

HAL_CEC_Init

Function name

HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)

Function description

Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_DeInit

Function name

HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)

Function description

DeInitializes the CEC peripheral.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_SetDeviceAddress

Function name

HAL_StatusTypeDef HAL_CEC_SetDeviceAddress (CEC_HandleTypeDef * hcec, uint16_t CEC_OwnAddress)

Function description

Initializes the Own Address of the CEC device.

Parameters

- **hcec:** CEC handle
- **CEC_OwnAddress:** The CEC own address.

Return values

- **HAL:** status

HAL_CEC_MspInit

Function name

void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP Init.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_MspDeInit

Function name

void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP DeInit.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CEC_RegisterCallback (CEC_HandleTypeDef * hcec, HAL_CEC_CallbackIDTypeDef CallbackID, pCEC_CallbackTypeDef pCallback)

Function description

Register a User CEC Callback To be used instead of the weak predefined callback.

Parameters

- **hcec:** CEC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_CEC_TX_CPLT_CB_ID Tx Complete callback ID
  - HAL_CEC_ERROR_CB_ID Error callback ID
  - HAL_CEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CEC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_CEC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CEC_UnRegisterCallback (CEC_HandleTypeDef * hcec, HAL_CEC_CallbackIDTypeDef CallbackID)

Function description

Unregister an CEC Callback CEC callback is redirected to the weak predefined callback.

Parameters

- **hcec:** uart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CEC_TX_CPLT_CB_ID Tx Complete callback ID
  - HAL_CEC_ERROR_CB_ID Error callback ID
  - HAL_CEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CEC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_CEC_RegisterRxCpltCallback

Function name

HAL_StatusTypeDef HAL_CEC_RegisterRxCpltCallback (CEC_HandleTypeDef * hcec, pCEC_RxCallbackTypeDef pCallback)

Function description

Register CEC RX complete Callback To be used instead of the weak HAL_CEC_RxCpltCallback() predefined callback.

Parameters

- **hcec:** CEC handle
- **pCallback:** pointer to the Rx transfer compelete Callback function

Return values

- **HAL:** status

HAL_CEC_UnRegisterRxCpltCallback

Function name

HAL_StatusTypeDef HAL_CEC_UnRegisterRxCpltCallback (CEC_HandleTypeDef * hcec)

Function description

UnRegister CEC RX complete Callback CEC RX complete Callback is redirected to the weak HAL_CEC_RxCpltCallback() predefined callback.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_Transmit_IT

Function name

HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t InitiatorAddress, uint8_t DestinationAddress, const uint8_t * pData, uint32_t Size)

Function description

Send data in interrupt mode.

Parameters

- **hcec:** CEC handle
- **InitiatorAddress:** Initiator address
- **DestinationAddress:** destination logical address
- **pData:** pointer to input byte data buffer
- **Size:** amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).

Return values

- **HAL:** status

HAL_CEC_GetLastReceivedFrameSize

Function name

uint32_t HAL_CEC_GetLastReceivedFrameSize (const CEC_HandleTypeDef * hcec)

Function description

Get size of the received frame.

Parameters

- **hcec:** CEC handle

Return values

- **Frame:** size

HAL_CEC_ChangeRxBuffer

Function name

void HAL_CEC_ChangeRxBuffer (CEC_HandleTypeDef * hcec, uint8_t * Rxbuffer)

Function description

Change Rx Buffer.

Parameters

- **hcec:** CEC handle
- **Rxbuffer:** Rx Buffer

Return values

- **Frame:** size

Notes

- This function can be called only inside the HAL_CEC_RxCpltCallback()

HAL_CEC_IRQHandler

Function name

void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)

Function description

This function handles CEC interrupt requests.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_TxCpltCallback

Function name

void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)

Function description

Tx Transfer completed callback.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_RxCpltCallback

Function name

void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec, uint32_t RxFrameSize)

Function description

Rx Transfer completed callback.

Parameters

- **hcec:** CEC handle
- **RxFrameSize:** Size of frame

Return values

- **None:**

HAL_CEC_ErrorCallback

Function name

void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)

Function description

CEC error callbacks.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_GetState

Function name

HAL_CEC_StateTypeDef HAL_CEC_GetState (const CEC_HandleTypeDef * hcec)

Function description

return the CEC state

Parameters

- **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC module.

Return values

- **HAL:** state

HAL_CEC_GetError

Function name

uint32_t HAL_CEC_GetError (const CEC_HandleTypeDef * hcec)

Function description

Return the CEC error code.

Parameters

- **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC.

Return values

- **CEC:** Error Code

9.3 CEC Firmware driver defines

The following section lists the various define and macros of the module.

9.3.1 CEC

CEC

CEC all RX or TX errors flags

CEC_ISR_ALL_ERROR

CEC Error Bit Generation if Bit Rise Error reported

CEC_BRE_ERRORBIT_NO_GENERATION

CEC_BRE_ERRORBIT_GENERATION

CEC Reception Stop on Error

CEC_NO_RX_STOP_ON_BRE

CEC_RX_STOP_ON_BRE

CEC Error Bit Generation on Broadcast message

CEC_BROADCASTERROR_ERRORBIT_GENERATION

CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION

CEC Error Code

HAL_CEC_ERROR_NONE

no error

HAL_CEC_ERROR_RXOVR

CEC Rx-Overrun

HAL_CEC_ERROR_BRE

CEC Rx Bit Rising Error

HAL_CEC_ERROR_SBPE

CEC Rx Short Bit period Error

HAL_CEC_ERROR_LBPE

CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACKE

CEC Rx Missing Acknowledge

HAL_CEC_ERROR_ARBLST

CEC Arbitration Lost

HAL_CEC_ERROR_TXUDR

CEC Tx-Buffer Underrun

HAL_CEC_ERROR_TXERR

CEC Tx-Error

HAL_CEC_ERROR_TXACKE

CEC Tx Missing Acknowledge

HAL_CEC_ERROR_INVALID_CALLBACK

Invalid Callback Error

CEC Exported Macros

__HAL_CEC_RESET_HANDLE_STATE

Description:

- Reset CEC handle gstate & RxState.

Parameters:

- __HANDLE__: CEC handle.

Return value:

- None

__HAL_CEC_GET_FLAG

Description:

- Checks whether or not the specified CEC interrupt flag is set.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the flag to check.
  - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
  - CEC_FLAG_TXERR: Tx Error.
  - CEC_FLAG_TXUDR: Tx-Buffer Underrun.
  - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
  - CEC_FLAG_TXBR: Tx-Byte Request.
  - CEC_FLAG_ARBLST: Arbitration Lost
  - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
  - CEC_FLAG_LBPE: Rx Long period Error
  - CEC_FLAG_SBPE: Rx Short period Error
  - CEC_FLAG_BRE: Rx Bit Rising Error
  - CEC_FLAG_RXOVR: Rx Overrun.
  - CEC_FLAG_RXEND: End Of Reception.
  - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- ITStatus

__HAL_CEC_CLEAR_FLAG

Description:

- Clears the interrupt or status flag when raised (write at 1)

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
  - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
  - CEC_FLAG_TXERR: Tx Error.
  - CEC_FLAG_TXUDR: Tx-Buffer Underrun.
  - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
  - CEC_FLAG_TXBR: Tx-Byte Request.
  - CEC_FLAG_ARBLST: Arbitration Lost
  - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
  - CEC_FLAG_LBPE: Rx Long period Error
  - CEC_FLAG_SBPE: Rx Short period Error
  - CEC_FLAG_BRE: Rx Bit Rising Error
  - CEC_FLAG_RXOVR: Rx Overrun.
  - CEC_FLAG_RXEND: End Of Reception.
  - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- none

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to enable. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_GET_IT_SOURCE

Description:

- Checks whether or not the specified CEC interrupt is enabled.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to check. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- FlagStatus

__HAL_CEC_ENABLE

Description:

- Enables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_DISABLE

Description:

- Disables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_FIRST_BYTE_TX_SET

Description:

- Set Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_LAST_BYTE_TX_SET

Description:

- Set Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none: If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.

__HAL_CEC_GET_TRANSMISSION_START_FLAG

Description:

- Get Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

__HAL_CEC_GET_TRANSMISSION_END_FLAG

Description:

- Get Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

__HAL_CEC_CLEAR_OAR

Description:

- Clear OAR register.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_SET_OAR

Description:

- Set OAR register (without resetting previously set address in case of multi-address mode) To reset OAR,

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __ADDRESS__: Own Address value (CEC logical address is identified by bit position)

Return value:

- none

CEC Flags definition

CEC_FLAG_TXACKE

CEC_FLAG_TXERR

CEC_FLAG_TXUDR

CEC_FLAG_TXEND

CEC_FLAG_TXBR

CEC_FLAG_ARBLST

CEC_FLAG_RXACKE

CEC_FLAG_LBPE

CEC_FLAG_SBPE

CEC_FLAG_BRE

CEC_FLAG_RXOVR

CEC_FLAG_RXEND

CEC_FLAG_RXBR

CEC all RX errors interrupts enabling flag

CEC_IER_RX_ALL_ERR

CEC all TX errors interrupts enabling flag

CEC_IER_TX_ALL_ERR

CEC Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

CEC Interrupts definition

CEC_IT_TXACKE

CEC_IT_TXERR

CEC_IT_TXUDR

CEC_IT_TXEND

CEC_IT_TXBR

CEC_IT_ARBLST

CEC_IT_RXACKE

CEC_IT_LBPE

CEC_IT_SBPE

CEC_IT_BRE

CEC_IT_RXOVR

CEC_IT_RXEND

CEC_IT_RXBR

CEC Error Bit Generation if Long Bit Period Error reported

CEC_LBPE_ERRORBIT_NO_GENERATION

CEC_LBPE_ERRORBIT_GENERATION

CEC Listening mode option

CEC_REDUCED_LISTENING_MODE

CEC_FULL_LISTENING_MODE

CEC Device Own Address position in CEC CFGR register

CEC_CFGR_OAR_LSB_POS

CEC Own Address

CEC_OWN_ADDRESS_NONE

CEC_OWN_ADDRESS_0

CEC_OWN_ADDRESS_1

CEC_OWN_ADDRESS_2

CEC_OWN_ADDRESS_3

CEC_OWN_ADDRESS_4

CEC_OWN_ADDRESS_5

CEC_OWN_ADDRESS_6

CEC_OWN_ADDRESS_7

CEC_OWN_ADDRESS_8

CEC_OWN_ADDRESS_9

CEC_OWN_ADDRESS_10

CEC_OWN_ADDRESS_11

CEC_OWN_ADDRESS_12

CEC_OWN_ADDRESS_13

CEC_OWN_ADDRESS_14

CEC Signal Free Time start option

CEC_SFT_START_ON_TXSOM

CEC_SFT_START_ON_TX_RX_END

CEC Signal Free Time setting parameter

CEC_DEFAULT_SFT

CEC_0_5_BITPERIOD_SFT

CEC_1_5_BITPERIOD_SFT

CEC_2_5_BITPERIOD_SFT

CEC_3_5_BITPERIOD_SFT

CEC_4_5_BITPERIOD_SFT

CEC_5_5_BITPERIOD_SFT

CEC_6_5_BITPERIOD_SFT

CEC State Code Definition

HAL_CEC_STATE_RESET

Peripheral is not yet Initialized Value is allowed for gState and RxState

HAL_CEC_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_CEC_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_CEC_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_CEC_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_CEC_STATE_BUSY_RX_TX

an internal process is ongoing Value is allowed for gState only

HAL_CEC_STATE_ERROR

Error Value is allowed for gState only

CEC Receiver Tolerance

CEC_STANDARD_TOLERANCE

CEC_EXTENDED_TOLERANCE

10 HAL COMP Generic Driver

10.1 COMP Firmware driver registers structures

10.1.1 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32h5xx_hal_comp.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t Hysteresis***
- ***uint32_t OutputPol***
- ***uint32_t BlankingSrce***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::Mode***
  Set comparator operating mode to adjust power and speed. Note: For the characteristics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of ***COMP_PowerMode***
- ***uint32_t COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_InputPlus***
- ***uint32_t COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_InputMinus***
- ***uint32_t COMP_InitTypeDef::Hysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_Hysteresis***
- ***uint32_t COMP_InitTypeDef::OutputPol***
  Set comparator output polarity. This parameter can be a value of ***COMP_OutputPolarity*** Note: Specific to comparator of this STM32 series: comparator output triggers interruption on high level. HAL_COMP_Start_x functions can change output polarity depending on initial output level.
- ***uint32_t COMP_InitTypeDef::BlankingSrce***
  Set comparator blanking source. This parameter can be a value of ***COMP_BlankingSrce***
- ***uint32_t COMP_InitTypeDef::TriggerMode***
  Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of ***COMP_EXTI_TriggerMode***

10.1.2 __COMP_HandleTypeDef

__COMP_HandleTypeDef is defined in the stm32h5xx_hal_comp.h

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TriggerCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***uint8_t InterruptAutoRearm***

Field Documentation

- ***COMP_TypeDef* __COMP_HandleTypeDef::Instance***
  Register base address

- ***COMP_InitTypeDef __COMP_HandleTypeDef::Init***
  COMP required parameters
- ***HAL_LockTypeDef __COMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_COMP_StateTypeDef __COMP_HandleTypeDef::State***
  COMP communication state
- ***__IO uint32_t __COMP_HandleTypeDef::ErrorCode***
  COMP error code
- ***void(* __COMP_HandleTypeDef::TriggerCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP trigger callback
- ***void(* __COMP_HandleTypeDef::MspInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp Init callback
- ***void(* __COMP_HandleTypeDef::MspDeInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp DeInit callback
- ***uint8_t __COMP_HandleTypeDef::InterruptAutoRearm***
  COMP interrupt auto rearm setting

10.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

10.2.1 COMP Peripheral features

The STM32H5xx device family integrates one analog comparator instance: COMP1.

1. Comparators input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. Comparators output level is available using HAL_COMP_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. The comparators have interrupt capability through direct line to NVIC (featuring low latency interrupt).
   Caution: Specific behavior for comparator of this STM32 series: comparator output triggers interruption on high level - triggering on level (instead of edge) implies to disable interrupt in comparator IRQ handler. In case of further operation needed in interrupt mode, comparator interruption must be rearmed. - triggering on high level implies that comparator output initial state must at low level. Then, comparator can trig signal on rising edge. Trigger a signal on falling edge is possible by inverting comparator polarity.
   ==========

10.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32H5xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL_COMP_MspInit():
   – Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL_GPIO_Init().
   – If needed, configure the GPIO connected to comparator output in alternate function mode using HAL_GPIO_Init().
   – If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.

2. Configure the comparator using HAL_COMP_Init() function:
   – Select the input minus (inverting input)
   – Select the input plus (non-inverting input)
   – Select the hysteresis
   – Select the blanking source
   – Select the output polarity
   – Select the power mode

Note: HAL_COMP_Init() calls "HAL_COMP_MspInit()", COMP clock enable using system RCC must be implemented in this function.

3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL_COMP_Init() with new input structure parameters values.
4. Enable the comparator using HAL_COMP_Start(), HAL_COMP_Start_IT_OneShot() or HAL_COMP_Start_IT_AutoRearm() Note: Using HAL_COMP_Start_IT_OneShot() or HAL_COMP_Start_IT_AutoRearm(), these functions can change comparator output polarity to match initial comparator output level constraint. Note: Using HAL_COMP_Start_IT_OneShot(), after each interruption triggered the interruption is disabled in IRQ handler. If needed, compararor interruption can be rearmed by calling again start function. Note: In case of comparator and interruption used to exit from low power mode, user most ensure of stable comparator input voltage (risk would be that comparator trigs early and IT disabled in IRQ handler before device entering in low power mode, inducing no further system wake up possible). Most appropriate function is HAL_COMP_Start_IT_AutoRearm() because compararor triggers remains enable, ensuring system wake up capability.
5. Use HAL_COMP_TriggerCallback() or HAL_COMP_GetOutputLevel() functions to manage comparator outputs (events and output level).
6. Disable the comparator using HAL_COMP_Stop() or HAL_COMP_Stop_IT() functions.
7. De-initialize the comparator using HAL_COMP_DeInit() function.
8. For safety purpose, comparator configuration can be locked using HAL_COMP_Lock() function. The only way to unlock the comparator is a device hardware reset.

Callback registration

The compilation flag USE_HAL_COMP_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_COMP_RegisterCallback() to register an interrupt callback.

Function HAL_COMP_RegisterCallback() allows to register following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_COMP_UnRegisterCallback to reset a callback to the default weak function.

HAL_COMP_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_COMP_Init() and when the state is HAL_COMP_STATE_RESET all callbacks are set to the corresponding weak functions: example HAL_COMP_TriggerCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_COMP_Init() / HAL_COMP_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_COMP_Init() / HAL_COMP_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_COMP_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_COMP_STATE_READY or HAL_COMP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_COMP_RegisterCallback() before calling HAL_COMP_DeInit() or HAL_COMP_Init() function.

When the compilation flag USE_HAL_COMP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

10.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- *HAL_COMP_Init()*
- *HAL_COMP_DeInit()*
- *HAL_COMP_MspInit()*
- *HAL_COMP_MspDeInit()*
- *HAL_COMP_RegisterCallback()*
- *HAL_COMP_UnRegisterCallback()*

10.2.4 IO operation functions

This section provides functions allowing to:

- Start a comparator instance.
- Stop a comparator instance.

This section contains the following APIs:

- *HAL_COMP_Start()*
- *HAL_COMP_Stop()*
- *HAL_COMP_Start_IT_OneShot()*
- *HAL_COMP_Start_IT_AutoRearm()*
- *HAL_COMP_Stop_IT()*
- *HAL_COMP_IRQHandler()*

10.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the comparators.

This section contains the following APIs:

- *HAL_COMP_Lock()*
- *HAL_COMP_GetOutputLevel()*
- *HAL_COMP_TriggerCallback()*

10.2.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_COMP_GetState()*
- *HAL_COMP_GetError()*

10.2.7 Detailed description of functions

HAL_COMP_Init

Function name

HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_COMP_DeInit

Function name

HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP peripheral.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name

void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_MspDeInit

Function name

void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_RegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID, pCOMP_CallbackTypeDef pCallback)

Function description

Register a User COMP Callback To be used instead of the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  - HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  - HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_COMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_UnRegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID)

Function description

Unregister a COMP Callback COMP callback is redirected to the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  - HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  - HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_COMP_Start

Function name

HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop

Function name

HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)

Function description

Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Start_IT_OneShot

Function name

HAL_StatusTypeDef HAL_COMP_Start_IT_OneShot (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator with interruption low latency, interruption disabled at first trigger occurrence.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Interruption low latency is achieved through direct line to NVIC (instead of going through EXTI).
- If needed, comparartor interruption can be rearmed by calling again this function.
- Specific to comparator of this STM32 series: comparator output triggers interruption on high level. This function can change output polarity depending on initial output level.

HAL_COMP_Start_IT_AutoRearm

Function name

HAL_StatusTypeDef HAL_COMP_Start_IT_AutoRearm (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator with interruption low latency, interruption rearmed at each trigger occurrence.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Interruption low latency is achieved through direct line to NVIC (instead of going through EXTI).
- If needed, comparartor interruption can be rearmed by calling again this function.
- Specific to comparator of this STM32 series: comparator output triggers interruption on high level. This function can change output polarity depending on initial output level.

HAL_COMP_Stop_IT

Function name

HAL_StatusTypeDef HAL_COMP_Stop_IT (COMP_HandleTypeDef * hcomp)

Function description

Disable the interrupt and Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_IRQHandler

Function name

void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)

Function description

Comparator IRQ handler.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_Lock

Function name

HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)

Function description

Lock the selected comparator configuration.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- A system reset is required to unlock the comparator configuration.

HAL_COMP_GetOutputLevel

Function name

uint32_t HAL_COMP_GetOutputLevel (const COMP_HandleTypeDef * hcomp)

Function description

Return the output level (high or low) of the selected comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **Returns:** the selected comparator output level:
  - COMP_OUTPUT_LEVEL_LOW
  - COMP_OUTPUT_LEVEL_HIGH

Notes

- The output level depends on the selected polarity. If the polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus
- Specific to comparator of this STM32 series: comparator output triggers interruption on high level. HAL_COMP_Start_x functions can change output polarity depending on initial output level.

HAL_COMP_TriggerCallback

Function name

void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)

Function description

Comparator trigger callback.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_GetState

Function name

HAL_COMP_StateTypeDef HAL_COMP_GetState (const COMP_HandleTypeDef * hcomp)

Function description

Return the COMP handle state.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** state

HAL_COMP_GetError

Function name

uint32_t HAL_COMP_GetError (const COMP_HandleTypeDef * hcomp)

Function description

Return the COMP error code.

Parameters

- **hcomp:** COMP handle

Return values

- **COMP:** error code

10.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 COMP

COMP

COMP blanking source

COMP_BLANKINGSRC_NONE

Comparator output without blanking

COMP_BLANKINGSRC_TIM1_OC5

TIM1 OC5 selected as blanking source

COMP_BLANKINGSRC_TIM2_OC3

TIM2 OC3 selected as blanking source

COMP_BLANKINGSRC_TIM3_OC3

TIM3 OC3 selected as blanking source

COMP_BLANKINGSRC_TIM3_OC4

TIM3 OC4 selected as blanking source

COMP_BLANKINGSRC_LPTIM1_OC2

LPTIM1 OC2 selected as blanking source

COMP_BLANKINGSRC_LPTIM2_OC2

LPTIM2 OC2 selected as blanking source

COMP Error Code

HAL_COMP_ERROR_NONE

No error

HAL_COMP_ERROR_INVALID_CALLBACK

Invalid Callback error

COMP Exported Types

COMP_STATE_BITFIELD_LOCK

COMP EXTI Lines

COMP_EXTI_LINE_COMP1

EXTI line 29 connected to COMP1 output

COMP_EXTI_IT

EXTI line event with interruption

COMP_EXTI_RISING

EXTI line event on rising edge

COMP_EXTI_FALLING

EXTI line event on falling edge

COMP external interrupt line management

__HAL_COMP_GET_IT_SOURCE

Description:

- Checks if the specified COMP interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the COMP Handle. This parameter can be COMP1.
- __INTERRUPT__: specifies the COMP interrupt source to check. This parameter can be one of the following values:
  - COMP_IT_EN Comparator interrupt enable

Return value:

- State: of interruption (TRUE or FALSE)

__HAL_COMP_GET_FLAG

Description:

- Checks whether the specified COMP flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - COMP_FLAG_C1I Comparator 1 Interrupt Flag

Return value:

- State: of flag (TRUE or FALSE)

__HAL_COMP_CLEAR_FLAG

Description:

- Clears the specified COMP pending flag.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - COMP_CLEAR_C1IF Clear Comparator 1 Interrupt Flag

Return value:

- None

__HAL_COMP_CLEAR_C1IFLAG

Description:

- Clear the COMP C1I flag.

Return value:

- None

__HAL_COMP_ENABLE_IT

Description:

- Enable the specified COMP interrupt.

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __INTERRUPT__: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
  - COMP_CFGR1_ITEN Comparator interrupt

Return value:

- None

__HAL_COMP_DISABLE_IT

Description:

- Disable the specified COMP interrupt.

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __INTERRUPT__: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
  - COMP_CFGR1_ITEN Comparator interrupt

Return value:

- None

COMP output to EXTI

COMP_TRIGGERMODE_NONE

Comparator output triggering no External Interrupt Line

COMP_TRIGGERMODE_IT_RISING_FALLING

Comparator output triggering interrupt on rising and falling edges. Note: Specific to comparator of this STM32 series: comparator output triggers interruption on high level. HAL_COMP_Start_x functions can change output polarity depending on initial output level.

COMP Flag

COMP_FLAG_C1I

Comparator 1 Interrupt Flag

COMP_FLAG_LOCK

Lock flag

COMP private macros to get EXTI line associated with comparators

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset COMP handle state.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

COMP_CLEAR_ERRORCODE

Description:

- Clear COMP error code (set it to no error code "HAL_COMP_ERROR_NONE").

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_ENABLE

Description:

- Enable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_DISABLE

Description:

- Disable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_LOCK

Description:

- Lock the specified comparator configuration.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

Notes:

- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock')".

__HAL_COMP_IS_LOCKED

Description:

- Check whether the specified comparator is locked.

Parameters:

- __HANDLE__: COMP handle

Return value:

- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP hysteresis

COMP_HYSTERESIS_NONE

No hysteresis

COMP_HYSTERESIS_LOW

Hysteresis level low

COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

COMP_HYSTERESIS_HIGH

Hysteresis level high

COMP input minus (inverting input)

COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1

COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PC4)

COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PB1)

COMP_INPUT_MINUS_IO3

Comparator input minus connected to IO3 (pin PA5)

COMP_INPUT_MINUS_TEMPSENSOR

Comparator input minus connected to internal temperature sensor (also accessible through ADC peripheral)

COMP_INPUT_MINUS_VBAT

Comparator input minus connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have input voltage always below Vdda.

COMP input plus (non-inverting input)

COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PB0)

COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PA0)

COMP_INPUT_PLUS_IO3

Comparator input plus connected to IO3 (pin PB2)

COMP_INPUT_PLUS_DAC1_CH1

Comparator input plus connected to (DAC1_CH1)

COMP Interrupts Definitions

COMP_IT_EN

COMP Interruption Clear Flags

COMP_CLEAR_C1IF

Clear Comparator 1 Interrupt Flag

COMP Output Level

COMP_OUTPUT_LEVEL_LOW

COMP_OUTPUT_LEVEL_HIGH

COMP output Polarity

COMP_OUTPUTPOL_NONINVERTED

COMP output level is not inverted (comparator output is high when the input plus is at a higher voltage than the input minus)

COMP_OUTPUTPOL_INVERTED

COMP output level is inverted (comparator output is low when the input plus is at a higher voltage than the input minus)

COMP power mode

COMP_POWERMODE_HIGHSPEED

High Speed

COMP_POWERMODE_MEDIUMSPEED

Medium Speed

COMP_POWERMODE_ULTRALOWPOWER

Ultra-low power mode

11 HAL CORDIC Generic Driver

11.1 CORDIC Firmware driver registers structures

11.1.1 __CORDIC_HandleTypeDef

__CORDIC_HandleTypeDef is defined in the stm32h5xx_hal_cordic.h

Data Fields

- ***CORDIC_TypeDef * Instance***
- ***const int32_t * pInBuff***
- ***int32_t * pOutBuff***
- ***uint32_t NbCalcToOrder***
- ***uint32_t NbCalcToGet***
- ***uint32_t DMADirection***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CORDIC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* CalculateCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CORDIC_TypeDef* __CORDIC_HandleTypeDef::Instance***
  Register base address
- ***const int32_t* __CORDIC_HandleTypeDef::pInBuff***
  Pointer to CORDIC input data buffer
- ***int32_t* __CORDIC_HandleTypeDef::pOutBuff***
  Pointer to CORDIC output data buffer
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToOrder***
  Remaining number of calculation to order
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToGet***
  Remaining number of calculation result to get
- ***uint32_t __CORDIC_HandleTypeDef::DMADirection***
  Direction of CORDIC DMA transfers
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaIn***
  CORDIC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaOut***
  CORDIC peripheral output data DMA handle parameters
- ***HAL_LockTypeDef __CORDIC_HandleTypeDef::Lock***
  CORDIC locking object
- ***__IO HAL_CORDIC_StateTypeDef __CORDIC_HandleTypeDef::State***
  CORDIC state
- ***__IO uint32_t __CORDIC_HandleTypeDef::ErrorCode***
  CORDIC peripheral error code This parameter can be a value of ***CORDIC_Error_Code***
- ***void(* __CORDIC_HandleTypeDef::ErrorCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC error callback
- ***void(* __CORDIC_HandleTypeDef::CalculateCpltCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC calculate complete callback

- ***void(* __CORDIC_HandleTypeDef::MspInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp Init callback
- ***void(* __CORDIC_HandleTypeDef::MspDeInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp DeInit callback

11.1.2 CORDIC_ConfigTypeDef

CORDIC_ConfigTypeDef is defined in the stm32h5xx_hal_cordic.h

Data Fields

- ***uint32_t Function***
- ***uint32_t Scale***
- ***uint32_t InSize***
- ***uint32_t OutSize***
- ***uint32_t NbWrite***
- ***uint32_t NbRead***
- ***uint32_t Precision***

Field Documentation

- ***uint32_t CORDIC_ConfigTypeDef::Function***
  Function This parameter can be a value of ***CORDIC_Function***
- ***uint32_t CORDIC_ConfigTypeDef::Scale***
  Scaling factor This parameter can be a value of ***CORDIC_Scale***
- ***uint32_t CORDIC_ConfigTypeDef::InSize***
  Width of input data This parameter can be a value of ***CORDIC_In_Size***
- ***uint32_t CORDIC_ConfigTypeDef::OutSize***
  Width of output data This parameter can be a value of ***CORDIC_Out_Size***
- ***uint32_t CORDIC_ConfigTypeDef::NbWrite***
  Number of 32-bit write expected for one calculation This parameter can be a value of ***CORDIC_Nb_Write***
- ***uint32_t CORDIC_ConfigTypeDef::NbRead***
  Number of 32-bit read expected after one calculation This parameter can be a value of ***CORDIC_Nb_Read***
- ***uint32_t CORDIC_ConfigTypeDef::Precision***
  Number of cycles for calculation This parameter can be a value of ***CORDIC_Precision_In_Cycles_Number***

11.2 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

11.2.1 How to use this driver

The CORDIC HAL driver can be used as follows:

1. Initialize the CORDIC low level resources by implementing the HAL_CORDIC_MspInit():
   - Enable the CORDIC interface clock using __HAL_RCC_CORDIC_CLK_ENABLE()
   - In case of using interrupts (e.g. HAL_CORDIC_Calculate_IT())
     - Configure the CORDIC interrupt priority using HAL_NVIC_SetPriority()
     - Enable the CORDIC IRQ handler using HAL_NVIC_EnableIRQ()
     - In CORDIC IRQ handler, call HAL_CORDIC_IRQHandler()
   - In case of using DMA to control data transfer (e.g. HAL_CORDIC_Calculate_DMA())
     - Enable the DMA2 interface clock using __HAL_RCC_DMA2_CLK_ENABLE()
     - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
     - Associate the initialized DMA handle to the CORDIC DMA handle using __HAL_LINKDMA()
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. Resort to HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the CORDIC HAL using HAL_CORDIC_Init(). This function
   – resorts to HAL_CORDIC_MspInit() for low-level initialization,
3. Configure CORDIC processing (calculation) using HAL_CORDIC_Configure(). This function configures:
   – Processing functions: Cosine, Sine, Phase, Modulus, Arctangent, Hyperbolic cosine, Hyperbolic sine, Hyperbolic arctangent, Natural log, Square root
   – Scaling factor: 1 to 2exp(-7)
   – Width of input data: 32 bits input data size (Q1.31 format) or 16 bits input data size (Q1.15 format)
   – Width of output data: 32 bits output data size (Q1.31 format) or 16 bits output data size (Q1.15 format)
   – Number of 32-bit write expected for one calculation: One 32-bits write or Two 32-bit write
   – Number of 32-bit read expected after one calculation: One 32-bits read or Two 32-bit read
   – Precision: 1 to 15 cycles for calculation (the more cycles, the better precision)
4. Four processing (calculation) functions are available:
   – Polling mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished API is HAL_CORDIC_Calculate
   – Polling Zero-overhead mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished A bit faster than standard polling mode, but blocking also AHB bus API is HAL_CORDIC_CalculateZO
   – Interrupt mode: processing API is not blocking functions i.e. it processes the data under interrupt API is HAL_CORDIC_Calculate_IT
   – DMA mode: processing API is not blocking functions and the CPU is not used for data transfer, i.e. the data transfer is ensured by DMA API is HAL_CORDIC_Calculate_DMA
5. Call HAL_CORDIC_DeInit() to de-initialize the CORDIC peripheral. This function
   – resorts to HAL_CORDIC_MspDeInit() for low-level de-initialization,

Callback registration

11.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CORDIC peripheral and the associated handle
- DeInitialize the CORDIC peripheral
- Initialize the CORDIC MSP (MCU Specific Package)
- De-Initialize the CORDIC MSP

This section contains the following APIs:

- ***HAL_CORDIC_Init()***
- ***HAL_CORDIC_DeInit()***
- ***HAL_CORDIC_MspInit()***
- ***HAL_CORDIC_MspDeInit()***
- ***HAL_CORDIC_RegisterCallback()***
- ***HAL_CORDIC_UnRegisterCallback()***
- ***HAL_CORDIC_RegisterCallback()***

11.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure the CORDIC peripheral: function, precision, scaling factor, number of input data and output data, size of input data and output data.
- Calculate output data of CORDIC processing on input date, using the existing CORDIC configuration

Four processing functions are available for calculation:

- Polling mode
- Polling mode, with Zero-Overhead register access
- Interrupt mode
- DMA mode

This section contains the following APIs:

- *HAL_CORDIC_Configure()*
- *HAL_CORDIC_Calculate()*
- *HAL_CORDIC_CalculateZO()*
- *HAL_CORDIC_Calculate_IT()*
- *HAL_CORDIC_Calculate_DMA()*

11.2.4 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt calculate complete
- DMA or Interrupt error

This section contains the following APIs:

- *HAL_CORDIC_ErrorCallback()*
- *HAL_CORDIC_CalculateCpltCallback()*

11.2.5 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_CORDIC_IRQHandler()*

11.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_CORDIC_GetState()*
- *HAL_CORDIC_GetError()*

11.2.7 Detailed description of functions

HAL_CORDIC_Init

Function name

HAL_StatusTypeDef HAL_CORDIC_Init (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC peripheral and the associated handle.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_DeInit

Function name

HAL_StatusTypeDef HAL_CORDIC_DeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC peripheral.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_MspInit

Function name

void HAL_CORDIC_MspInit (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_MspDeInit

Function name

void HAL_CORDIC_MspDeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_RegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID, pCORDIC_CallbackTypeDef pCallback)

Function description

HAL_CORDIC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_UnRegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID)

Function description

Unregister a CORDIC CallBack.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CORDIC_ERROR_CB_ID error Callback ID
  - HAL_CORDIC_CALCULATE_CPLT_CB_ID calculate complete Callback ID
  - HAL_CORDIC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CORDIC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_CORDIC_Configure

Function name

HAL_StatusTypeDef HAL_CORDIC_Configure (CORDIC_HandleTypeDef * hcordic, const CORDIC_ConfigTypeDef * sConfig)

Function description

Configure the CORDIC processing according to the specified parameters in the CORDIC_ConfigTypeDef structure.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **sConfig:** pointer to a CORDIC_ConfigTypeDef structure that contains the CORDIC configuration information.

Return values

- **HAL:** status

HAL_CORDIC_Calculate

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in polling mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_CalculateZO

Function name

HAL_StatusTypeDef HAL_CORDIC_CalculateZO (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in Zero-Overhead mode (output data being read soon as input data are written), according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_Calculate_IT

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_IT (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc)

Function description

Carry out data of CORDIC processing in interrupt mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.

Return values

- **HAL:** status

HAL_CORDIC_Calculate_DMA

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_DMA (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t DMADirection)

Function description

Carry out input and/or output data of CORDIC processing in DMA mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **DMADirection:** Direction of DMA transfers. This parameter can be one of the following values:
  - CORDIC DMA direction CORDIC DMA direction

Return values

- **HAL:** status

Notes

- pInBuff or pOutBuff is unused in case of unique DMADirection transfer, and can be set to NULL value in this case.
- pInBuff and pOutBuff buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the Peripheral.

HAL_CORDIC_ErrorCallback

Function name

void HAL_CORDIC_ErrorCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC error callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_CalculateCpltCallback

Function name

void HAL_CORDIC_CalculateCpltCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC calculate complete callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_IRQHandler

Function name

void HAL_CORDIC_IRQHandler (CORDIC_HandleTypeDef * hcordic)

Function description

Handle CORDIC interrupt request.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_GetState

Function name

HAL_CORDIC_StateTypeDef HAL_CORDIC_GetState (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC handle state.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **HAL:** state

HAL_CORDIC_GetError

Function name

uint32_t HAL_CORDIC_GetError (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC peripheral error.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **Error:** bit-map

Notes

- The returned error is a bit-map combination of possible errors

11.3 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 CORDIC

CORDIC

DMA Read Request Enable bit

CORDIC_DMA_REN

DMA Read requests enable

DMA Write Request Enable bit

CORDIC_DMA_WEN

DMA Write channel enable

CORDIC DMA direction

CORDIC_DMA_DIR_NONE

DMA direction : none

CORDIC_DMA_DIR_IN

DMA direction : Input of CORDIC

CORDIC_DMA_DIR_OUT

DMA direction : Output of CORDIC

CORDIC_DMA_DIR_IN_OUT

DMA direction : Input and Output of CORDIC

CORDIC Error code

HAL_CORDIC_ERROR_NONE

No error

HAL_CORDIC_ERROR_PARAM

Wrong parameter error

HAL_CORDIC_ERROR_NOT_READY

Peripheral not ready

HAL_CORDIC_ERROR_TIMEOUT

Timeout error

HAL_CORDIC_ERROR_DMA

DMA error

HAL_CORDIC_ERROR_INVALID_CALLBACK

Invalid Callback error

CORDIC Exported Macros

__HAL_CORDIC_RESET_HANDLE_STATE

Description:

- Reset CORDIC handle state.

Parameters:

- __HANDLE__: CORDIC handle

Return value:

- None

__HAL_CORDIC_ENABLE_IT

Description:

- Enable the CORDIC interrupt when result is ready.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_DISABLE_IT

Description:

- Disable the CORDIC interrupt.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_GET_IT

Description:

- Check whether the specified CORDIC interrupt occurred or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_CORDIC_CLEAR_IT

Description:

- Clear specified CORDIC interrupt status.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to clear

Return value:

- None

__HAL_CORDIC_GET_FLAG

Description:

- Check whether the specified CORDIC status flag is set or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to check This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_CORDIC_CLEAR_FLAG

Description:

- Clear specified CORDIC status flag.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to clear This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- None

__HAL_CORDIC_GET_IT_SOURCE

Description:

- Check whether the specified CORDIC interrupt is enabled or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- FlagStatus

CORDIC status flags

CORDIC_FLAG_RRDY

Result Ready Flag

CORDIC Function

CORDIC_FUNCTION_COSINE

Cosine

CORDIC_FUNCTION_SINE

Sine

CORDIC_FUNCTION_PHASE

Phase

CORDIC_FUNCTION_MODULUS

Modulus

CORDIC_FUNCTION_ARCTANGENT

Arctangent

CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

CORDIC_FUNCTION_HSINE

Hyperbolic Sine

CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

CORDIC_FUNCTION_SQUAREROOT

Square Root

CORDIC Interrupts Enable bit

CORDIC_IT_IEN

Result ready interrupt enable

CORDIC input data size

CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

CORDIC Number of 32-bit read required after one calculation

CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

CORDIC Number of 32-bit write required for one calculation

CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

CORDIC Results Size

CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

CORDIC Precision in Cycles Number

CORDIC_PRECISION_1CYCLE

CORDIC_PRECISION_2CYCLES

CORDIC_PRECISION_3CYCLES

CORDIC_PRECISION_4CYCLES

CORDIC_PRECISION_5CYCLES

CORDIC_PRECISION_6CYCLES

CORDIC_PRECISION_7CYCLES

CORDIC_PRECISION_8CYCLES

CORDIC_PRECISION_9CYCLES

CORDIC_PRECISION_10CYCLES

CORDIC_PRECISION_11CYCLES

CORDIC_PRECISION_12CYCLES

CORDIC_PRECISION_13CYCLES

CORDIC_PRECISION_14CYCLES

CORDIC_PRECISION_15CYCLES

CORDIC Scaling factor

CORDIC_SCALE_0

CORDIC_SCALE_1

CORDIC_SCALE_2

CORDIC_SCALE_3

CORDIC_SCALE_4

CORDIC_SCALE_5

CORDIC_SCALE_6

CORDIC_SCALE_7

12 HAL CORTEX Generic Driver

12.1 CORTEX Firmware driver registers structures

12.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32h5xx_hal_cortex.h

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint32_t LimitAddress***
- ***uint8_t AttributesIndex***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the index of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint32_t MPU_Region_InitTypeDef::LimitAddress***
  Specifies the limit address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::AttributesIndex***
  Specifies the memory attributes index. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***

12.1.2 MPU_Attributes_InitTypeDef

MPU_Attributes_InitTypeDef is defined in the stm32h5xx_hal_cortex.h

Data Fields

- ***uint8_t Number***
- ***uint8_t Attributes***

Field Documentation

- ***uint8_t MPU_Attributes_InitTypeDef::Number***
  Specifies the number of the memory attributes to configure. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***
- ***uint8_t MPU_Attributes_InitTypeDef::Attributes***
  Specifies the memory attributes value. Attributes This parameter can be a combination of ***CORTEX_MPU_Attributes***

12.2 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

12.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M33 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().

Note: When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- *Lowest pre-emption priority*
- *Lowest sub priority*
- *Lowest hardware priority (IRQ number)*

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32h5xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

How to configure MPU (secure and non secure) using CORTEX HAL driver

This section provides functions allowing to Enable and configure the MPU secure and non-secure.

1. Enable the MPU using HAL_MPU_Enable() function.
2. Disable the MPU using HAL_MPU_Disable() function.
3. Enable the MPU using HAL_MPU_Enable_NS() function to address the non secure MPU.
4. Disable the MPU using HAL_MPU_Disable_NS() function to address the non secure MPU.
5. Configure the MPU region using HAL_MPU_ConfigRegion() and HAL_MPU_ConfigRegion_NS() to address the non secure MPU.
6. Configure the MPU Memory attributes using HAL_MPU_ConfigMemoryAttributes() and HAL_MPU_ConfigMemoryAttributes_NS() to address the non secure MPU.

12.2.2 NVIC functions

This section provides the CORTEX HAL driver functions for NVIC functionalities

This section contains the following APIs:

- *HAL_NVIC_SetPriorityGrouping()*
- *HAL_NVIC_SetPriority()*
- *HAL_NVIC_EnableIRQ()*
- *HAL_NVIC_DisableIRQ()*
- *HAL_NVIC_SystemReset()*
- *HAL_NVIC_GetPriorityGrouping()*
- *HAL_NVIC_GetPriority()*
- *HAL_NVIC_SetPendingIRQ()*
- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_NVIC_GetActive()*

12.2.3 SYSTICK functions

This section provides the CORTEX HAL driver functions for SYSTICK functionalities

This section contains the following APIs:

- *HAL_SYSTICK_Config()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*

12.2.4 MPU functions

This section provides the CORTEX HAL driver functions for MPU functionalities

This section contains the following APIs:

- *HAL_MPU_Enable()*
- *HAL_MPU_Enable_NS()*
- *HAL_MPU_Disable()*
- *HAL_MPU_Disable_NS()*
- *HAL_MPU_ConfigRegion()*
- *HAL_MPU_ConfigRegion_NS()*
- *HAL_MPU_ConfigMemoryAttributes()*
- *HAL_MPU_ConfigMemoryAttributes_NS()*

12.2.5 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)

Function description

Set the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority

Return values

- **None:**

Notes

- When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Set the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))
- **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

Function description

Enable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)

Function description

Disable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

void HAL_NVIC_SystemReset (void)

Function description

Initiate a system reset request to reset the MCU.

Return values

- **None:**

HAL_NVIC_GetPriorityGrouping

Function name

uint32_t HAL_NVIC_GetPriorityGrouping (void)

Function description

Get the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t *const pPreemptPriority, uint32_t *const pSubPriority)

Function description

Get the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

Function description

Get Pending Interrupt (read the pending register in the NVIC and return the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)

Function description

Set Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clear the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Get active interrupt (read the active register in NVIC and return the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h5xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_SYSTICK_Config

Function name

uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)

Function description

Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick): Counter is in free running mode to generate periodic interrupts.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configure the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  - SYSTICK_CLKSOURCE_LSI: LSI clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_LSE: LSE clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

Handle SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_MPU_Enable

Function name

void HAL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable the MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable

Function name

void HAL_MPU_Disable (void)

Function description

Disable the MPU.

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (const MPU_Region_InitTypeDef *const pMPU_RegionInit)

Function description

Initialize and configure the Region and the memory to be protected.

Parameters

- **pMPU_RegionInit:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_ConfigMemoryAttributes

Function name

void HAL_MPU_ConfigMemoryAttributes (const MPU_Attributes_InitTypeDef *const pMPU_AttributesInit)

Function description

Initialize and configure the memory attributes.

Parameters

- **pMPU_AttributesInit:** Pointer to a MPU_Attributes_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_Enable_NS

Function name

void HAL_MPU_Enable_NS (uint32_t MPU_Control)

Function description

Enable the non-secure MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable_NS

Function name

void HAL_MPU_Disable_NS (void)

Function description

Disable the non-secure MPU.

Return values

- **None:**

HAL_MPU_ConfigRegion_NS

Function name

void HAL_MPU_ConfigRegion_NS (const MPU_Region_InitTypeDef *const pMPU_RegionInit)

Function description

Initialize and configure the Region and the memory to be protected for non-secure MPU.

Parameters

- **pMPU_RegionInit:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_ConfigMemoryAttributes_NS

Function name

void HAL_MPU_ConfigMemoryAttributes_NS (const MPU_Attributes_InitTypeDef *const pMPU_AttributesInit)

Function description

Initialize and configure the memory attributes for non-secure MPU.

Parameters

- **pMPU_AttributesInit:** Pointer to a MPU_Attributes_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

12.3 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

12.3.1 CORTEX

CORTEX

CORTEX Exported Macros

OUTER

INNER_OUTER

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_NOT_SHAREABLE

MPU region not shareable

MPU_ACCESS_OUTER_SHAREABLE

MPU region outer shareable

MPU_ACCESS_INNER_SHAREABLE

MPU region inner shareable

CORTEX MPU Attributes

MPU_DEVICE_nGnRnE

Device, noGather, noReorder, noEarly acknowledge.

MPU_DEVICE_nGnRE

Device, noGather, noReorder, Early acknowledge.

MPU_DEVICE_nGRE

Device, noGather, Reorder, Early acknowledge.

MPU_DEVICE_GRE

Device, Gather, Reorder, Early acknowledge.

MPU_WRITE_THROUGH

Normal memory, write-through.

MPU_NOT_CACHEABLE

Normal memory, non-cacheable.

MPU_WRITE_BACK

Normal memory, write-back.

MPU_TRANSIENT

Normal memory, transient.

MPU_NON_TRANSIENT

Normal memory, non-transient.

MPU_NO_ALLOCATE

Normal memory, no allocate.

MPU_W_ALLOCATE

Normal memory, write allocate.

MPU_R_ALLOCATE

Normal memory, read allocate.

MPU_RW_ALLOCATE

Normal memory, read/write allocate.

CORTEX MPU Memory Attributes Number

MPU_ATTRIBUTES_NUMBER0

MPU attribute number 0

MPU_ATTRIBUTES_NUMBER1

MPU attribute number 1

MPU_ATTRIBUTES_NUMBER2

MPU attribute number 2

MPU_ATTRIBUTES_NUMBER3

MPU attribute number 3

MPU_ATTRIBUTES_NUMBER4

MPU attribute number 4

MPU_ATTRIBUTES_NUMBER5

MPU attribute number 5

MPU_ATTRIBUTES_NUMBER6

MPU attribute number 6

MPU_ATTRIBUTES_NUMBER7

MPU attribute number 7

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU is disabled during HardFault and NMI handlers, privileged software access to the default memory map is disabled

MPU_HARDFAULT_NMI

MPU is enabled during HardFault and NMI handlers, privileged software access to the default memory map is disabled

MPU_PRIVILEGED_DEFAULT

MPU is disabled during HardFault and NMI handlers, privileged software access to the default memory map is enabled

MPU_HFNMI_PRIVDEF

MPU is enabled during HardFault and NMI handlers, privileged software access to the default memory map is enabled

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU region execution permitted (if read permitted)

MPU_INSTRUCTION_ACCESS_DISABLE

MPU region execution not permitted

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU region enabled

MPU_REGION_DISABLE

MPU region disabled

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU region number 0

MPU_REGION_NUMBER1

MPU region number 1

MPU_REGION_NUMBER2

MPU region number 2

MPU_REGION_NUMBER3

MPU region number 3

MPU_REGION_NUMBER4

MPU region number 4

MPU_REGION_NUMBER5

MPU region number 5

MPU_REGION_NUMBER6

MPU region number 6

MPU_REGION_NUMBER7

MPU region number 7

MPU_REGION_NUMBER8

MPU region number 8

MPU_REGION_NUMBER9

MPU region number 9

MPU_REGION_NUMBER10

MPU region number 10

MPU_REGION_NUMBER11

MPU region number 11

CORTEX MPU Region Permission Attributes

MPU_REGION_PRIV_RW

MPU region Read/write by privileged code only

MPU_REGION_ALL_RW

MPU region Read/write by any privilege level

MPU_REGION_PRIV_RO

MPU region Read-only by privileged code only

MPU_REGION_ALL_RO

MPU region Read-only by any privilege level

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0

0 bit for pre-emption priority, 4 bits for subpriority

NVIC_PRIORITYGROUP_1

1 bit for pre-emption priority, 3 bits for subpriority

NVIC_PRIORITYGROUP_2

2 bits for pre-emption priority, 2 bits for subpriority

NVIC_PRIORITYGROUP_3

3 bits for pre-emption priority, 1 bit for subpriority

NVIC_PRIORITYGROUP_4

4 bits for pre-emption priority, 0 bit for subpriority

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source

SYSTICK_CLKSOURCE_LSI

LSI clock selected as SysTick clock source

SYSTICK_CLKSOURCE_LSE

LSE clock selected as SysTick clock source

SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source

13 HAL CRC Generic Driver

13.1 CRC Firmware driver registers structures

13.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32h5xx_hal_crc.h

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***
  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.
- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***
  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.
- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***
  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal, representation e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.
- ***uint32_t CRC_InitTypeDef::CRCLength***
  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of
  - **CRC_POLYLENGTH_32B** (32-bit CRC),
  - **CRC_POLYLENGTH_16B** (16-bit CRC),
  - **CRC_POLYLENGTH_8B** (8-bit CRC),
  - **CRC_POLYLENGTH_7B** (7-bit CRC).
- ***uint32_t CRC_InitTypeDef::InitValue***
  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.
- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***
  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values
  - **CRC_INPUTDATA_INVERSION_NONE** no input data inversion
  - **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2
  - **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C
  - **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either
  - **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion,
  - **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

13.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32h5xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either
  - **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data)
  - **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data)
  - **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data)

  Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

13.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

13.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  - specify generating polynomial (peripheral default or non-default one)
  - specify initialization value (peripheral default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

13.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
- DeInitialize the CRC peripheral

- Initialize the CRC MSP (MCU Specific Package)
- DeInitialize the CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

13.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one.

or

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- ***HAL_CRC_Accumulate()***
- ***HAL_CRC_Calculate()***

13.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- ***HAL_CRC_GetState()***

13.2.5 Detailed description of functions

HAL_CRC_Init

Function name

HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)

Function description

Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_DeInit

Function name

HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC peripheral.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_MspInit

Function name

void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)

Function description

Initializes the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_MspDeInit

Function name

void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_Accumulate

Function name

uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name

uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name

HAL_CRC_StateTypeDef HAL_CRC_GetState (const CRC_HandleTypeDef * hcrc)

Function description

Return the CRC handle state.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** state

13.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

13.3.1 CRC

CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

Enable initial CRC default value

DEFAULT_INIT_VALUE_DISABLE

Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

Enable default generating polynomial 0x04C11DB7

DEFAULT_POLYNOMIAL_DISABLE

Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY

X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store data in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: Value to be stored in the ID register

Return value:

- None

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

__HAL_CRC_GET_IDR

Description:

- Return the data stored in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- Value: of the ID register

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

Undefined input data format

CRC_INPUTDATA_FORMAT_BYTES

Input data in byte format

CRC_INPUTDATA_FORMAT_HALFWORDS

Input data in half-word format

CRC_INPUTDATA_FORMAT_WORDS

Input data in word format

Polynomial sizes to configure the peripheral

CRC_POLYLENGTH_32B

Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B

Resort to a 16-bit long generating polynomial

CRC_POLYLENGTH_8B

Resort to a 8-bit long generating polynomial

CRC_POLYLENGTH_7B

Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

32-bit long CRC

HAL_CRC_LENGTH_16B

16-bit long CRC

HAL_CRC_LENGTH_8B

8-bit long CRC

HAL_CRC_LENGTH_7B

7-bit long CRC

14 HAL CRC Extension Driver

14.1 CRCEx Firmware driver API description

The following section lists the various functions of the CRCEx library.

14.1.1 How to use this driver

- Set user-defined generating polynomial through HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

14.1.2 Extended configuration functions

This section provides functions allowing to:

- Configure the generating polynomial
- Configure the input data inversion
- Configure the output data inversion

This section contains the following APIs:

- ***HAL_CRCEx_Polynomial_Set()***
- ***HAL_CRCEx_Input_Data_Reverse()***
- ***HAL_CRCEx_Output_Data_Reverse()***

14.1.3 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name

HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)

Function description

Initialize the CRC polynomial if different from default one.

Parameters

- **hcrc:** CRC handle
- **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long). This parameter is written in normal representation, e.g.
  - for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
  - for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021
- **PolyLength:** CRC polynomial length. This parameter can be one of the following values:
  - CRC_POLYLENGTH_7B 7-bit long CRC (generating polynomial of degree 7)
  - CRC_POLYLENGTH_8B 8-bit long CRC (generating polynomial of degree 8)
  - CRC_POLYLENGTH_16B 16-bit long CRC (generating polynomial of degree 16)
  - CRC_POLYLENGTH_32B 32-bit long CRC (generating polynomial of degree 32)

Return values

- **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)

Function description

Set the Reverse Input data mode.

Parameters

- **hcrc:** CRC handle
- **InputReverseMode:** Input Data inversion mode. This parameter can be one of the following values:
  - CRC_INPUTDATA_INVERSION_NONE no change in bit order (default value)
  - CRC_INPUTDATA_INVERSION_BYTE Byte-wise bit reversal
  - CRC_INPUTDATA_INVERSION_HALFWORD HalfWord-wise bit reversal
  - CRC_INPUTDATA_INVERSION_WORD Word-wise bit reversal

Return values

- **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)

Function description

Set the Reverse Output data mode.

Parameters

- **hcrc:** CRC handle
- **OutputReverseMode:** Output Data inversion mode. This parameter can be one of the following values:
  - CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion (default value)
  - CRC_OUTPUTDATA_INVERSION_ENABLE bit-level inversion (e.g. for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values

- **HAL:** status

14.2 CRCEx Firmware driver defines

The following section lists the various define and macros of the module.

14.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE

Description:

- Set CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:

- Unset CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

No input data inversion

CRC_INPUTDATA_INVERSION_BYTE

Byte-wise input data inversion

CRC_INPUTDATA_INVERSION_HALFWORD

HalfWord-wise input data inversion

CRC_INPUTDATA_INVERSION_WORD

Word-wise input data inversion

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE

No output data inversion

CRC_OUTPUTDATA_INVERSION_ENABLE

Bit-wise output data inversion

15 HAL CRYP Generic Driver

15.1 CRYP Firmware driver registers structures

15.1.1 CRYP_ConfigTypeDef

CRYP_ConfigTypeDef is defined in the stm32h5xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t * Header***
- ***uint32_t HeaderSize***
- ***uint32_t * B0***
- ***uint32_t DataWidthUnit***
- ***uint32_t HeaderWidthUnit***
- ***uint32_t KeyIVConfigSkip***
- ***uint32_t KeyMode***
- ***uint32_t KeySelect***
- ***uint32_t KeyProtection***

Field Documentation

- ***uint32_t CRYP_ConfigTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ConfigTypeDef::KeySize***
  Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. 128 or 256 bit key length in TinyAES This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ConfigTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ConfigTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer
- ***uint32_t* CRYP_ConfigTypeDef::B0***
  B0 is first authentication block used only in AES CCM mode
- ***uint32_t CRYP_ConfigTypeDef::DataWidthUnit***
  Payload Data Width Unit, this parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::HeaderWidthUnit***
  Header Width Unit, this parameter can be value of ***CRYP_Header_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::KeyIVConfigSkip***
  CRYP peripheral Key and IV configuration skip, to config Key and Initialization Vector only once and to skip configuration for consecutive processings. This parameter can be a value of ***CRYP_Configuration_Skip***
- ***uint32_t CRYP_ConfigTypeDef::KeyMode***
  Key mode selection, this parameter can be value of ***CRYP_Key_Mode***
- ***uint32_t CRYP_ConfigTypeDef::KeySelect***
  Only for SAES : Key selection, this parameter can be value of ***CRYP_Key_Select***

- ***uint32_t CRYP_ConfigTypeDef::KeyProtection***
  Only for SAES : Key protection, this parameter can be value of ***CRYP_Key_Protection***

15.1.2 CRYP_ContextTypeDef

CRYP_ContextTypeDef is defined in the stm32h5xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t DataWidthUnit***
- ***uint32_t KeyIVConfigSkip***
- ***uint32_t KeyMode***
- ***uint32_t Phase***
- ***uint32_t KeyIVConfig***
- ***uint32_t CR_Reg***
- ***uint32_t IER_Reg***
- ***uint32_t IVR0_Reg***
- ***uint32_t IVR1_Reg***
- ***uint32_t IVR2_Reg***
- ***uint32_t IVR3_Reg***

Field Documentation

- ***uint32_t CRYP_ContextTypeDef::DataType***
  This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ContextTypeDef::KeySize***
  This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ContextTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ContextTypeDef::pInitVect***
  The initialization vector, counter with CBC and CTR Algorithm
- ***uint32_t CRYP_ContextTypeDef::Algorithm***
  This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t CRYP_ContextTypeDef::DataWidthUnit***
  This parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ContextTypeDef::KeyIVConfigSkip***
  This parameter can be a value of ***CRYP_Configuration_Skip***
- ***uint32_t CRYP_ContextTypeDef::KeyMode***
  This parameter can be value of ***CRYP_Key_Mode***
- ***uint32_t CRYP_ContextTypeDef::Phase***
  CRYP peripheral phase
- ***uint32_t CRYP_ContextTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag
- ***uint32_t CRYP_ContextTypeDef::CR_Reg***
  CRYP CR register
- ***uint32_t CRYP_ContextTypeDef::IER_Reg***
  CRYP IER register
- ***uint32_t CRYP_ContextTypeDef::IVR0_Reg***
  CRYP IVR0 register
- ***uint32_t CRYP_ContextTypeDef::IVR1_Reg***
  CRYP IVR1 register
- ***uint32_t CRYP_ContextTypeDef::IVR2_Reg***
  CRYP IVR2 register

- ***uint32_t CRYP_ContextTypeDef::IVR3_Reg***
  CRYP IVR3 register

15.1.3 __CRYP_HandleTypeDef

__CRYP_HandleTypeDef is defined in the stm32h5xx_hal_cryp.h

Data Fields

- ***AES_TypeDef * Instance***
- ***CRYP_ConfigTypeDef Init***
- ***uint32_t * pCrypInBuffPtr***
- ***uint32_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypHeaderCount***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***uint16_t Size***
- ***uint32_t Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t KeyIVConfig***
- ***uint32_t SizesSum***
- ***void(* InCpltCallback***
- ***void(* OutCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***AES_TypeDef* __CRYP_HandleTypeDef::Instance***
  AES Register base address
- ***CRYP_ConfigTypeDef __CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***uint32_t* __CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint32_t* __CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypHeaderCount***
  Counter of header data in words
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypInCount***
  Counter of input data in words
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypOutCount***
  Counter of output data in words
- ***uint16_t __CRYP_HandleTypeDef::Size***
  length of input data in word or in byte, according to DataWidthUnit
- ***uint32_t __CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters
- ***HAL_LockTypeDef __CRYP_HandleTypeDef::Lock***
  CRYP locking object

- ***__IO HAL_CRYP_STATETypeDef __CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t __CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t __CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped
- ***uint32_t __CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing
- ***void(* __CRYP_HandleTypeDef::InCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Input FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::OutCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Output FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::ErrorCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Error callback
- ***void(* __CRYP_HandleTypeDef::MspInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp Init callback
- ***void(* __CRYP_HandleTypeDef::MspDeInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp DeInit callback

15.2 CRYP Firmware driver API description

The following section lists the various functions of the CRYP library.

15.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()or __HAL_RCC_AES_CLK_ENABLE
   b. In case of using interrupts (e.g. HAL_CRYP_Encrypt_IT())
      - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
      - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
      - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
   c. In case of using DMA to control data transfer (e.g. HAL_CRYP_Encrypt_DMA())
      - Enable the DMAx interface clock using __RCC_DMAx_CLK_ENABLE()
      - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
      - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. The output channel should have higher priority than the input channel HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the CRYP according to the specified parameters :
   a. The data type: 1-bit, 8-bit, 16-bit or 32-bit.
   b. The key size: 128, 192 or 256.
   c. The AES Algorithm ECB/CBC/CTR/GCM or CCM.
   d. The initialization vector (counter). It is not used in ECB mode.
   e. The key buffer used for encryption/decryption.
      - In some specific configurations, the key is written by the application code out of the HAL scope. In that case, user can still resort to the HAL APIs as usual but must make sure that pKey pointer is set to NULL.
   f. The DataWidthUnit field. It specifies whether the data length (or the payload length for authentication algorithms) is in words or bytes.
   g. The Header used only in AES GCM and CCM Algorithm for authentication.
   h. The HeaderSize used to give size of header buffer in word or bytes, depending upon HeaderWidthUnit field.
   i. The HeaderWidthUnit field. It specifies whether the header length (for authentication algorithms) is in words or bytes.
   j. The B0 block is the first authentication block used only in AES CCM mode.
   k. The KeyIVConfigSkip used to process several messages in a row.
   l. The KeyMode used to special key operation modes (for SAES : wrapped key, shared key with AES peripheral).
   m. The KeySelect, Only for SAES, used to select key from different key source.
   n. The KeyProtection, Only for SAES, used for security context enforcement.
3. Three processing (encryption/decryption) functions are available:
   a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
   c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA
4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is configured and processes the buffer in input. At second call, no need to Initialize the CRYP, user have to get current configuration via HAL_CRYP_GetConfig() API, then only HAL_CRYP_SetConfig() is requested to set new parameters, finally user can start encryption/decryption.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.
6. To process a single message with consecutive calls to HAL_CRYP_Encrypt() or HAL_CRYP_Decrypt() without having to configure again the Key or the Initialization Vector between each API call, the field KeyIVConfigSkip of the initialization structure must be set to CRYP_KEYIVCONFIG_ONCE. Same is true for consecutive calls of HAL_CRYP_Encrypt_IT(), HAL_CRYP_Decrypt_IT(), HAL_CRYP_Encrypt_DMA() or HAL_CRYP_Decrypt_DMA().

The cryptographic processor supports following standards:

1. The advanced encryption standard (AES) supported:
   a. 128-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
      - Counter mode (CTR)
      - Galois/counter mode (GCM/GMAC)
      - Counter with Cipher Block Chaining-Message(CCM)
   c. keys length Supported : 128-bit and 256-bit

Note: *Specific care must be taken to format the key and the Initialization Vector IV!*

If the key is defined as a 128-bit long array key[127..0] = {b127 ... b0} where b127 is the MSB and b0 the LSB, the key must be stored in MCU memory

- as a sequence of words where the MSB word comes first (occupies the lowest memory address)
  - address n+0 : 0b b127 .. b120 b119 .. b112 b111 .. b104 b103 .. b96
  - address n+4 : 0b b95 .. b88 b87 .. b80 b79 .. b72 b71 .. b64
  - address n+8 : 0b b63 .. b56 b55 .. b48 b47 .. b40 b39 .. b32
  - address n+C : 0b b31 .. b24 b23 .. b16 b15 .. b8 b7 .. b0

Hereafter, another illustration when considering a 128-bit long key made of 16 bytes {B15..B0}. The 4 32-bit words that make the key must be stored as follows in MCU memory:

- address n+0 : 0x B15 B14 B13 B12
- address n+4 : 0x B11 B10 B9 B8
- address n+8 : 0x B7 B6 B5 B4
- address n+C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Same format must be applied for a 256-bit long key made of 32 bytes {B31..B0}. The 8 32-bit words that make the key must be stored as follows in MCU memory:

- address n+00 : 0x B31 B30 B29 B28
- address n+04 : 0x B27 B26 B25 B24
- address n+08 : 0x B23 B22 B21 B20
- address n+0C : 0x B19 B18 B17 B16
- address n+10 : 0x B15 B14 B13 B12
- address n+14 : 0x B11 B10 B9 B8
- address n+18 : 0x B7 B6 B5 B4
- address n+1C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR7 = 0x B31 B30 B29 B28
- AES_KEYR6 = 0x B27 B26 B25 B24
- AES_KEYR5 = 0x B23 B22 B21 B20
- AES_KEYR4 = 0x B19 B18 B17 B16
- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Initialization Vector IV (4 32-bit words) format must follow the same as that of a 128-bit long key.

Note that key and IV registers are not sensitive to swap mode selection.

This section describes the AES Galois/counter mode (GCM) supported by the peripherals:

1. Algorithm supported :
   a. Galois/counter mode (GCM)
   b. Galois message authentication code (GMAC) :is exactly the same as GCM algorithm composed only by an header.
2. Four phases are performed in GCM :
   a. Init phase: peripheral prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: peripheral processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: peripheral processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: peripheral generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESGCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits.

3. structure of message construction in GCM is defined as below :
   a. 16 bytes Initial Counter Block (ICB)composed of IV and counter
   b. The authenticated header A (also knows as Additional Authentication Data AAD) this part of the message is only authenticated, not encrypted.
   c. The plaintext message P is both authenticated and encrypted as ciphertext. GCM standard specifies that ciphertext has same bit length as the plaintext.
   d. The last block is composed of the length of A (on 64 bits) and the length of ciphertext (on 64 bits)

A more detailed description of the GCM message structure is available below.

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by the peripheral:

1. Specific parameters for CCM :
   a. B0 block : follows NIST Special Publication 800-38C,
   b. B1 block (header)
   c. CTRx block : control blocks

A detailed description of the CCM message structure is available below.

1. CCM in peripheral:
   a. To perform message payload encryption or decryption AES is configured in CTR mode.
   b. For authentication two phases are performed : - Header phase: peripheral processes the Additional Authenticated Data (AAD) first, then the cleartext message only cleartext payload (not the ciphertext payload) is used and no output.
   c. Final phase: peripheral generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESCCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits

Callback registration

The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_CRYP_RegisterCallback() or HAL_CRYP_RegisterXXXCallback() to register an interrupt callback.

Function @ref HAL_CRYP_RegisterCallback() allows to register following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_CRYP_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_CRYP_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit.

By default, after the @ref HAL_CRYP_Init() and when the state is HAL_CRYP_STATE_RESET all callbacks are set to the corresponding weak functions : examples @ref HAL_CRYP_InCpltCallback() , @ref HAL_CRYP_OutCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak function in the @ref HAL_CRYP_Init()/ @ref HAL_CRYP_DeInit() only when these callbacks are null (not registered beforehand). if not, MspInit or MspDeInit are not null, the @ref HAL_CRYP_Init() / @ref HAL_CRYP_DeInit() keep and use the user MspInit/MspDeInit functions (registered beforehand)

Callbacks can be registered/unregistered in HAL_CRYP_STATE_READY state only. Exception done MspInit/ MspDeInit callbacks that can be registered/unregistered in HAL_CRYP_STATE_READY or HAL_CRYP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_CRYP_RegisterCallback() before calling @ref HAL_CRYP_DeInit() or @ref HAL_CRYP_Init() function.

When The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Suspend/Resume feature

The compilation define USE_HAL_CRYP_SUSPEND_RESUME when set to 1 allows the user to resort to the suspend/resume feature. A low priority block processing can be suspended to process a high priority block instead. When the high priority block processing is over, the low priority block processing can be resumed, restarting from the point where it was suspended. This feature is applicable only in non-blocking interrupt mode.

User must resort to HAL_CRYP_Suspend() to suspend the low priority block processing. This API manages the hardware block processing suspension and saves all the internal data that will be needed to restart later on. Upon HAL_CRYP_Suspend() completion, the user can launch the processing of any other block (high priority block processing).

When the high priority block processing is over, user must invoke HAL_CRYP_Resume() to resume the low priority block processing. Ciphering (or deciphering) restarts from the suspension point and ends as usual.

HAL_CRYP_Suspend() reports an error when the suspension request is sent too late (i.e when the low priority block processing is about to end). There is no use to suspend the tag generation processing for authentication algorithms.

Note: If the key is written out of HAL scope (case pKey pointer set to NULL by the user), the block processing suspension/resumption mechanism is NOT applicable.

Note: If the Key and Initialization Vector are configured only once and configuration is skipped for consecutive processings (case KeyIVConfigSkip set to CRYP_KEYIVCONFIG_ONCE), the block processing suspension/ resumption mechanism is NOT applicable.

15.2.2 Initialization, de-initialization and Set and Get configuration functions

This section provides functions allowing to:

- Initialize the CRYP
- DeInitialize the CRYP
- Initialize the CRYP MSP
- DeInitialize the CRYP MSP
- configure CRYP (HAL_CRYP_SetConfig) with the specified parameters in the CRYP_ConfigTypeDef Parameters which are configured in This section are :
- Key size
- Data Type : 32,16, 8 or 1bit
- AlgoMode : ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard.
- Get CRYP configuration (HAL_CRYP_GetConfig) from the specified parameters in the CRYP_HandleTypeDef
- For interleave mode, API HAL_CRYP_SaveContext and HAL_CRYP_RestoreContext to be used to save then Restore CRYP configuration and parameters. CRYP_IVCONFIG_ONCE should be selected for KeyIVConfigSkip parameter. Only polling mode is supported, interleave mode should be used with HAL_CRYP_Encrypt and HAL_CRYP_Decrypt API.

This section contains the following APIs:

- ***HAL_CRYP_Init()***
- ***HAL_CRYP_DeInit()***
- ***HAL_CRYP_SetConfig()***
- ***HAL_CRYP_GetConfig()***
- ***HAL_CRYP_MspInit()***
- ***HAL_CRYP_MspDeInit()***
- ***HAL_CRYP_RegisterCallback()***
- ***HAL_CRYP_UnRegisterCallback()***
- ***HAL_CRYP_SaveContext()***
- ***HAL_CRYP_RestoreContext()***

15.2.3 Encrypt Decrypt functions

This section provides API allowing to Encrypt/Decrypt Data following

- Standard AES algorithms supported by the peripheral: - Electronic Code Book(ECB) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Galois/ counter mode (GCM) - Counter with Cipher Block Chaining-Message(CCM)

Three processing functions are available:

- Polling mode : HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
- Interrupt mode : HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
- DMA mode : HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA

This section contains the following APIs:

- ***HAL_CRYP_Encrypt()***
- ***HAL_CRYP_Decrypt()***
- ***HAL_CRYP_Encrypt_IT()***
- ***HAL_CRYP_Decrypt_IT()***
- ***HAL_CRYP_Encrypt_DMA()***
- ***HAL_CRYP_Decrypt_DMA()***

15.2.4 CRYP IRQ handler management

This section provides CRYP IRQ handler and callback functions.

- HAL_CRYP_IRQHandler CRYP interrupt request
- HAL_CRYP_InCpltCallback input data transfer complete callback
- HAL_CRYP_OutCpltCallback output data transfer complete callback
- HAL_CRYP_ErrorCallback CRYP error callback
- HAL_CRYP_GetState return the CRYP state
- HAL_CRYP_GetError return the CRYP error code

This section contains the following APIs:

- ***HAL_CRYP_IRQHandler()***
- ***HAL_CRYP_GetError()***
- ***HAL_CRYP_GetState()***
- ***HAL_CRYP_InCpltCallback()***
- ***HAL_CRYP_OutCpltCallback()***
- ***HAL_CRYP_ErrorCallback()***

15.2.5 Detailed description of functions

HAL_CRYP_Init

Function name

HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP according to the specified parameters in the CRYP_ConfigTypeDef and creates the associated handle.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_DeInit

Function name

HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)

Function description

De-Initializes the CRYP peripheral.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_MspInit

Function name

void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_MspDeInit

Function name

void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)

Function description

DeInitializes CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_SetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_SetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Configure the CRYP according to the specified parameters in the CRYP_ConfigTypeDef.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure
- **pConf:** pointer to a CRYP_ConfigTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_RegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID, pCRYP_CallbackTypeDef pCallback)

Function description

Register a User CRYP Callback To be used instead of the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_CRYP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_UnRegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID)

Function description

Unregister an CRYP Callback CRYP callback is redirected to the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_CRYP_SaveContext

Function name

HAL_StatusTypeDef HAL_CRYP_SaveContext (CRYP_HandleTypeDef * hcryp, CRYP_ContextTypeDef * pcont)

Function description

CRYP peripheral parameters storage when processing Interleaved mode .

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pcont:** pointer to a CRYP_ContextTypeDef structure where CRYP parameters will be stored.

Return values

- **HAL:** status

HAL_CRYP_RestoreContext

Function name

HAL_StatusTypeDef HAL_CRYP_RestoreContext (CRYP_HandleTypeDef * hcryp, CRYP_ContextTypeDef * pcont)

Function description

Restore CRYP parameters needed for Interleaved mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pcont:** pointer to a CRYP_ContextTypeDef structure that contains CRYP parameters stored.

Return values

- **HAL:** status

HAL_CRYP_Encrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput, uint32_t Timeout)

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput, uint32_t Timeout)

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Decryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

HAL_CRYP_STATETypeDef HAL_CRYP_GetState (const CRYP_HandleTypeDef * hcryp)

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (const CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP peripheral

Return values

- **CRYP:** error code

15.3 CRYP Firmware driver defines

The following section lists the various define and macros of the module.

15.3.1 CRYP

CRYP

CRYP Algorithm Mode

CRYP_AES_ECB

Electronic codebook chaining algorithm

CRYP_AES_CBC

Cipher block chaining algorithm

CRYP_AES_CTR

Counter mode chaining algorithm

CRYP_AES_GCM_GMAC

Galois counter mode - Galois message authentication code

CRYP_AES_CCM

Counter with Cipher Mode

CRYP CLEAR Flags

CRYP_CLEAR_CCF

Computation Complete Flag Clear

CRYP_CLEAR_RWEIF

Clear Error Flag : RWEIF in AES_ISR and both RDERR and WRERR flags in AES_SR

CRYP_CLEAR_KEIF

Clear Key Error Flag: KEIF in AES_ISR

CRYP_CLEAR_RNGEIF

Clear rng Error Flag: RNGEIF in AES_ISR

CRYP Key and IV Configuration Skip Mode

CRYP_KEYIVCONFIG_ALWAYS

Peripheral Key and IV configuration to do systematically

CRYP_KEYIVCONFIG_ONCE

Peripheral Key and IV configuration to do only once

CRYP_KEYNOCONFIG

Peripheral Key configuration to not do

CRYP_IVCONFIG_ONCE

Peripheral IV configuration do once for interleave mode

CRYP Data Type

CRYP_DATATYPE_32B

CRYP_DATATYPE_16B

CRYP_DATATYPE_8B

CRYP_DATATYPE_1B

CRYP_NO_SWAP

32-bit data type (no swapping)

CRYP_HALFWORD_SWAP

16-bit data type (half-word swapping)

CRYP_BYTE_SWAP

8-bit data type (byte swapping)

CRYP_BIT_SWAP

1-bit data type (bit swapping)

CRYP Data Width Unit

CRYP_DATAWIDTHUNIT_WORD

By default, size unit is word

CRYP_DATAWIDTHUNIT_BYTE

By default, size unit is byte

CRYP Error Definition

HAL_CRYP_ERROR_NONE

No error

HAL_CRYP_ERROR_WRITE

Write error

HAL_CRYP_ERROR_READ

Read error

HAL_CRYP_ERROR_DMA

DMA error

HAL_CRYP_ERROR_BUSY

Busy flag error

HAL_CRYP_ERROR_TIMEOUT

Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED

Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE

Sequence are not respected only for GCM or CCM

HAL_CRYP_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_CRYP_ERROR_KEY

Key error

HAL_CRYP_ERROR_RNG

Rng error

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

__HAL_CRYP_GET_FLAG

Description:

- Check whether the specified CRYP status flag is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values for TinyAES:
  - CRYP_FLAG_KEYVALID Key valid flag
  - CRYP_FLAG_BUSY GCM process suspension forbidden or transferring a shared key from SAES IP.
  - CRYP_FLAG_WRERR Write Error flag
  - CRYP_FLAG_RDERR Read Error flag
  - CRYP_FLAG_CCF Computation Complete flag
  - CRYP_FLAG_KEIF Key error flag
  - CRYP_FLAG_RWEIF Read/write Error flag

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the CRYP pending status flag.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP_CLEAR_RWEIF Read (RDERR), Write (WRERR) or Read/write (RWEIF) Error Flag Clear
  - CRYP_CLEAR_CCF Computation Complete Flag (CCF) Clear
  - CRYP_CLEAR_KEIF Key error interrupt flag clear

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

- Check whether the specified CRYP interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt source to check This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- None

CRYP Flags

CRYP_FLAG_BUSY

GCM process suspension forbidden also set when transferring a shared key from SAES peripheral

CRYP_FLAG_WRERR

Write Error flag

CRYP_FLAG_RDERR

Read error flag

CRYP_FLAG_CCF

Computation completed flag as AES_ISR_CCF

CRYP_FLAG_KEYVALID

Key Valid flag

CRYP_FLAG_KEIF

Key error interrupt flag

CRYP_FLAG_RWEIF

Read or write error Interrupt flag

CRYP_FLAG_RNGEIF

RNG error interrupt flag

CRYP Header Width Unit

CRYP_HEADERWIDTHUNIT_WORD

By default, header size unit is word

CRYP_HEADERWIDTHUNIT_BYTE

By default, header size unit is byte

CRYP Interrupt

CRYP_IT_CCFIE

Computation Complete interrupt enable

CRYP_IT_RWEIE

Read or write Error interrupt enable

CRYP_IT_KEIE

Key error interrupt enable

CRYP_IT_RNGEIE

Rng error interrupt enable

CRYP Key Mode

CRYP_KEYMODE_NORMAL

Normal key usage, Key registers are freely usable

CRYP_KEYMODE_SHARED

Shared key

CRYP_KEYMODE_WRAPPED

Only for SAES, Wrapped key: to encrypt or decrypt AES keys

CRYP Key Protection

CRYP_KEYPROT_ENABLE

Only for SAES, Key protection between 2 applications with different security contexts

CRYP_KEYPROT_DISABLE

Only for SAES, Key not protected between 2 applications with different security contexts

CRYP Key Select

CRYP_KEYSEL_NORMAL

Normal key, key registers SAES_KEYx or CRYP_KEYx

CRYP_KEYSEL_HW

Only for SAES, Hardware key : derived hardware unique key (DHUK 256-bit)

CRYP_KEYSEL_SW

Only for SAES, Software key : boot hardware key BHK (256-bit)

CRYP_KEYSEL_HSW

Only for SAES, DHUK XOR BHK Hardware unique key XOR software key

CRYP Key Share ID

CRYP_KSHAREID_AES

Share SAES Key with AES peripheral

CRYP Key Size

CRYP_KEYSIZE_128B

128-bit long key

CRYP_KEYSIZE_256B

256-bit long key

16 HAL CRYP Extension Driver

16.1 CRYPEx Firmware driver API description

The following section lists the various functions of the CRYPEx library.

16.1.1 Extended AES processing functions

This section provides functions allowing to generate the authentication TAG in Polling mode

1. HAL_CRYPEx_AESGCM_GenerateAuthTAG
2. HAL_CRYPEx_AESCCM_GenerateAuthTAG they should be used after Encrypt/Decrypt operation.

This section contains the following APIs:

- ***HAL_CRYPEx_AESGCM_GenerateAuthTAG()***
- ***HAL_CRYPEx_AESCCM_GenerateAuthTAG()***

16.1.2 Wrap and Unwrap key

This section provides API allowing to wrap (encrypt) and unwrap (decrypt) key using one of the following keys, and AES Algorithm. Key selection : - Derived hardware unique key (DHUK) - XOR of DHUK and BHK - Boot hardware key (BHK)

This section contains the following APIs:

- ***HAL_CRYPEx_WrapKey()***
- ***HAL_CRYPEx_UnwrapKey()***

16.1.3 Encrypt and Decrypt Shared key functions

This section provides API allowing to Encrypt and Decrypt Shared key

This section contains the following APIs:

- ***HAL_CRYPEx_EncryptSharedKey()***
- ***HAL_CRYPEx_DecryptSharedKey()***

16.1.4 Detailed description of functions

HAL_CRYPEx_AESGCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

generate the GCM authentication TAG.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pAuthTag:** Pointer to the authentication buffer the pAuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of pAuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

AES CCM Authentication TAG generation.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pAuthTag:** Pointer to the authentication buffer the pAuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of pAuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_UnwrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_UnwrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint32_t Timeout)

Function description

Unwrap (Decrypt) application keys.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the Key buffer to decrypt or generated key in case of CTR.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_WrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_WrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint32_t * pOutput, uint32_t Timeout)

Function description

Wrap (encrypt) application keys.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the Key buffer to encrypt in case of ECB or CBC
- **pOutput:** Pointer to the Key buffer encrypted in case of ECB or CBC
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_EncryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_EncryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * pKey, uint32_t * pOutput, uint32_t ID, uint32_t Timeout)

Function description

Encrypt Shared key.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pKey:** Pointer to the Key buffer to share
- **pOutput:** Pointer to the Key buffer encrypted
- **ID:** Key share identification
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_DecryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_DecryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * pKey, uint32_t ID, uint32_t Timeout)

Function description

Decrypt Shared key.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pKey:** Pointer to the Key buffer to share
- **ID:** Key share identification
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

17 HAL DAC Generic Driver

17.1 DAC Firmware driver registers structures

17.1.1 __DAC_HandleTypeDef

__DAC_HandleTypeDef is defined in the stm32h5xx_hal_dac.h

Data Fields

- ***DAC_TypeDef * Instance***
- ***__IO HAL_DAC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * DMA_Handle1***
- ***DMA_HandleTypeDef * DMA_Handle2***
- ***__IO uint32_t ErrorCode***
- ***void(* ConvCpltCallbackCh1***
- ***void(* ConvHalfCpltCallbackCh1***
- ***void(* ErrorCallbackCh1***
- ***void(* DMAUnderrunCallbackCh1***
- ***void(* ConvCpltCallbackCh2***
- ***void(* ConvHalfCpltCallbackCh2***
- ***void(* ErrorCallbackCh2***
- ***void(* DMAUnderrunCallbackCh2***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DAC_TypeDef* __DAC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_DAC_StateTypeDef __DAC_HandleTypeDef::State***
  DAC communication state
- ***HAL_LockTypeDef __DAC_HandleTypeDef::Lock***
  DAC locking object
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle1***
  Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle2***
  Pointer DMA handler for channel 2
- ***__IO uint32_t __DAC_HandleTypeDef::ErrorCode***
  DAC Error code
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::MspInitCallback)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::MspDeInitCallback)(struct __DAC_HandleTypeDef *hdac)***

17.1.2 DAC_SampleAndHoldConfTypeDef

DAC_SampleAndHoldConfTypeDef is defined in the stm32h5xx_hal_dac.h

Data Fields

- ***uint32_t DAC_SampleTime***
- ***uint32_t DAC_HoldTime***
- ***uint32_t DAC_RefreshTime***

Field Documentation

- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime***
  Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime***
  Specifies the hold time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime***
  Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

17.1.3 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32h5xx_hal_dac.h

Data Fields

- ***uint32_t DAC_HighFrequency***
- ***FunctionalState DAC_DMADoubleDataMode***
- ***FunctionalState DAC_SignedFormat***
- ***uint32_t DAC_SampleAndHold***
- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_OutputBuffer***
- ***uint32_t DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_UserTrimming***
- ***uint32_t DAC_TrimmingValue***
- ***DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig***

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_HighFrequency***
  Specifies the frequency interface mode This parameter can be a value of ***DAC_HighFrequency***
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_DMADoubleDataMode***
  Specifies if DMA double data mode should be enabled or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_SignedFormat***
  Specifies if signed format should be used or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold***
  Specifies whether the DAC mode. This parameter can be a value of ***DAC_SampleAndHold***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger***
  Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer***
  Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral***
  Specifies whether the DAC output is connected or not to on chip peripheral. This parameter can be a value of ***DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***DAC_UserTrimming*** DAC_UserTrimming is either factory or user trimming

- ***uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue***
  Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig***
  Sample and Hold settings

17.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

17.2.1 DAC Peripheral features

DAC Channels

STM32H5 devices integrate two 12-bit Digital Analog Converters The 2 converters (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output or connected to on-chip peripherals (ex. ADC).
2. DAC channel2 with DAC_OUT2 (PA5) as output or connected to on-chip peripherals (ex. ADC).

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM1, TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM8 and TIM15 (DAC_TRIGGER_T1_TRGO, DAC_TRIGGER_T2_TRGO...)
3. Low Power Timers CH1: LPTIM1 and LPTIM2 (DAC_TRIGGER_LPTIM1_CH1, DAC_TRIGGER_LPTIM2_CH1)
4. Software using DAC_TRIGGER_SOFTWARE

The trigger selection depends on the PWR mode: in stop0, stop1 and stop2 we should select DAC_TRIGGER_EXT_IT9, DAC_TRIGGER_LPTIM1_CH1 or DAC_TRIGGER_LPTIM2_CH1.The other triggers are not functional.

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note: Refer to the device datasheet for more details about output impedance value with and without output buffer.

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion. The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA). The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode, low power run, low power sleep mode & stop1 mode. Low power stop1 mode allows only static conversion. To enable Sample and Hold mode Enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE; & DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefreshTime;

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
   - Calibration aims at correcting some offset of output buffer.
   - The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
   - HAL_DACEx_SelfCalibrate:
     - Runs automatically the calibration.
     - Enables the user trimming mode
     - Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VREF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA request can be generated when an external trigger (but not a software trigger) occurs if DMA requests are enabled using HAL_DAC_Start_DMA(). DMA requests are mapped as following: GPDMA requests are mapped as following:

- DAC channel1 mapped on GPDMA request 2 (can be any GPDMA channel)

- DAC channel2 mapped on GPDMA request 3 (can be any GPDMA channel)

High frequency interface mode

The high frequency interface informs DAC instance about the bus frequency in use. It is mandatory information for DAC (as internal timing of DAC is bus frequency dependent) provided thanks to parameter DAC_HighFrequency handled in HAL_DAC_ConfigChannel () function. Use of DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC value of DAC_HighFrequency is recommended function figured out the correct setting. The high frequency mode is same for all converters of a same DAC instance. Either same parameter DAC_HighFrequency is used for all DAC converters or again self DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC detection parameter.

Note: For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description

17.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion First issued trigger will start the conversion of the value previously set by HAL_DAC_SetValue().
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

Callback registration

The compilation define USE_HAL_DAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_DAC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_DAC_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit.
- All Callbacks This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_DAC_Init and if the state is HAL_DAC_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_DAC_Init and HAL_DAC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_DAC_Init and HAL_DAC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DAC_RegisterCallback before calling HAL_DAC_DeInit or HAL_DAC_Init function. When The compilation define USE_HAL_DAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

Note: You can refer to the DAC HAL driver header file for more useful macros

17.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- ***HAL_DAC_Init()***
- ***HAL_DAC_DeInit()***
- ***HAL_DAC_MspInit()***

- *HAL_DAC_MspDeInit()*
- *HAL_DAC_RegisterCallback()*
- *HAL_DAC_UnRegisterCallback()*

17.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.

This section contains the following APIs:

- *HAL_DAC_Start()*
- *HAL_DAC_Stop()*
- *HAL_DAC_Start_DMA()*
- *HAL_DAC_Stop_DMA()*
- *HAL_DAC_IRQHandler()*
- *HAL_DAC_SetValue()*
- *HAL_DAC_ConvCpltCallbackCh1()*
- *HAL_DAC_ConvHalfCpltCallbackCh1()*
- *HAL_DAC_ErrorCallbackCh1()*
- *HAL_DAC_DMAUnderrunCallbackCh1()*
- *HAL_DAC_RegisterCallback()*
- *HAL_DAC_UnRegisterCallback()*

17.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DAC_GetValue()*
- *HAL_DAC_ConfigChannel()*

17.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*

17.2.7 Detailed description of functions

HAL_DAC_Init

Function name

HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DeInit

Function name

HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)

Function description

Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name

void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDeInit

Function name

void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)

Function description

DeInitialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name

HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Stop

Function name

HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, const uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DAC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_IRQHandler

Function name

void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)

Function description

Handles DAC interrupt request This function uses the interruption of DMA underrun.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_DMAUnderrunCallbackCh1

Function name

void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_RegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID, pDAC_CallbackTypeDef pCallback)

Function description

Register a User DAC Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DAC_ERROR_INVALID_CALLBACK DAC Error Callback ID
  - HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 Complete Callback ID
  - HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  - HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  - HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  - HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  - HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  - HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  - HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  - HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  - HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_DAC_RegisterCallback() may be called before HAL_DAC_Init() in HAL_DAC_STATE_RESET to register callbacks for HAL_DAC_MSPINIT_CB_ID and HAL_DAC_MSPDEINIT_CB_ID

HAL_DAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_UnRegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a User DAC Callback DAC Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 transfer Complete Callback ID
  - HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  - HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  - HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  - HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  - HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  - HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  - HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  - HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  - HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
  - HAL_DAC_ALL_CB_ID DAC All callbacks

Return values

- **status:**

Notes

- The HAL_DAC_UnRegisterCallback() may be called before HAL_DAC_Init() in HAL_DAC_STATE_RESET to un-register callbacks for HAL_DAC_MSPINIT_CB_ID and HAL_DAC_MSPDEINIT_CB_ID

HAL_DAC_GetValue

Function name

uint32_t HAL_DAC_GetValue (const DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Returns the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, const DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Configures the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

Notes

- By calling this function, the high frequency interface mode (HFSEL bits) will be set. This parameter scope is the DAC instance. As the function is called for each channel, the DAC high frequency interface mode of sConfig must be the same at each call. (or DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC self detect).

HAL_DAC_GetState

Function name

HAL_DAC_StateTypeDef HAL_DAC_GetState (const DAC_HandleTypeDef * hdac)

Function description

return the DAC handle state

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** state

HAL_DAC_GetError

Function name

uint32_t HAL_DAC_GetError (const DAC_HandleTypeDef * hdac)

Function description

Return the DAC error code.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **DAC:** Error Code

DAC_DMAConvCpltCh1

Function name

void DAC_DMAConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh1

Function name

void DAC_DMAErrorCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh1

Function name

void DAC_DMAHalfConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

17.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

17.3.1 DAC

DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_EXTERNAL

DAC_CHIPCONNECT_INTERNAL

DAC_CHIPCONNECT_BOTH

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERRUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

HAL_DAC_ERROR_TIMEOUT

Timeout error

HAL_DAC_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DAC_ERROR_INVALID_CONFIG

Invalid configuration error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG

Description:

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag
  - DAC_FLAG_DAC1RDY DAC channel 1 ready status flag
  - DAC_FLAG_DAC2RDY DAC channel 2 ready status flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC_FLAG_DAC1RDY

DAC_FLAG_DAC2RDY

DAC high frequency interface mode

DAC_HIGH_FREQUENCY_INTERFACE_MODE_DISABLE

High frequency interface mode disabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_80MHZ

High frequency interface mode compatible to AHB>80MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_160MHZ

High frequency interface mode compatible to AHB>160MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC

High frequency interface mode automatic

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC power mode

DAC_SAMPLEANDHOLD_DISABLE

DAC_SAMPLEANDHOLD_ENABLE

DAC trigger selection

DAC_TRIGGER_NONE

conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger

DAC_TRIGGER_SOFTWARE

conversion started by software trigger for DAC channel

DAC_TRIGGER_T1_TRGO

TIM1 TRGO selected as external conversion trigger for DAC channel.

DAC_TRIGGER_T2_TRGO

TIM2 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T6_TRGO

TIM6 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T7_TRGO

TIM7 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM1_CH1

LPTIM1 CH1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM2_CH1

LPTIM2 CH1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_EXT_IT9

EXTI Line9 event selected as external conversion trigger for DAC channel

DAC_TRIGGER_T4_TRGO

TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T5_TRGO

TIM5 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T8_TRGO

TIM8 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T15_TRGO

TIM15 TRGO selected as external conversion trigger for DAC channel

DAC User Trimming

DAC_TRIMMING_FACTORY

Factory trimming

DAC_TRIMMING_USER

User trimming

18 HAL DAC Extension Driver

18.1 DACEx Firmware driver API description

The following section lists the various functions of the DACEx library.

18.1.1 How to use this driver

Dual mode IO operation

- Use HAL_DACEx_DualStart() to enable both channel and start conversion for dual mode operation. If software trigger is selected, using HAL_DACEx_DualStart() will start the conversion of the value previously set by HAL_DACEx_DualSetValue().
- Use HAL_DACEx_DualStop() to disable both channel and stop conversion for dual mode operation.
- Use HAL_DACEx_DualStart_DMA() to enable both channel and start conversion for dual mode operation using DMA to feed DAC converters. First issued trigger will start the conversion of the value previously set by HAL_DACEx_DualSetValue(). The same callbacks that are used in single mode are called in dual mode to notify transfer completion (half complete or complete), errors or underrun.
- Use HAL_DACEx_DualStop_DMA() to disable both channel and stop conversion for dual mode operation using DMA to feed DAC converters.
- When Dual mode is enabled (i.e. DAC Channel1 and Channel2 are used simultaneously) : Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.

Signal generation operation

- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- HAL_DACEx_SelfCalibrate to calibrate one DAC channel.
- HAL_DACEx_SetUserTrimming to set user trimming value.
- HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming have been used at least one time after reset).

18.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- ***HAL_DACEx_DualStart()***
- ***HAL_DACEx_DualStop()***
- ***HAL_DACEx_DualStart_DMA()***
- ***HAL_DACEx_DualStop_DMA()***
- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_DualSetValue()***
- ***HAL_DACEx_ConvCpltCallbackCh2()***
- ***HAL_DACEx_ConvHalfCpltCallbackCh2()***
- ***HAL_DACEx_ErrorCallbackCh2()***

- *HAL_DACEx_DMAUnderrunCallbackCh2()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*
- *HAL_DACEx_DualGetValue()*

18.1.3 Peripheral Control functions

This section provides functions allowing to:

- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DACEx_DualGetValue()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*

18.1.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
  - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- **HAL:** status

HAL_DACEx_DualStart

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart (DAC_HandleTypeDef * hdac)

Function description

Enables DAC and starts conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStop

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop (DAC_HandleTypeDef * hdac)

Function description

Disables DAC and stop conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStart_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, const uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of both channel 1 and 2 of the same DAC.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that will request data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DACEx_DualStop_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion both channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that requests data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DACEx_DualSetValue

Function name

HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)

Function description

Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel1 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel2 to be loaded in the selected data holding register.

Return values

- **HAL:** status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_DualGetValue

Function name

uint32_t HAL_DACEx_DualGetValue (const DAC_HandleTypeDef * hdac)

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **The:** selected DAC channel data output value.

HAL_DACEx_ConvCpltCallbackCh2

Function name

void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name

void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name

void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name

void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Run the self calibration of one DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC channel configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming
- **HAL:** status

Notes

- Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name

HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)

Function description

Set the trimming mode and trimming value (user trimming mode applied).

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure updated with new DAC trimming value.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **NewTrimmingValue:** DAC new trimming value

Return values

- **HAL:** status

HAL_DACEx_GetTrimOffset

Function name

uint32_t HAL_DACEx_GetTrimOffset (const DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Return the DAC trimming value.

Parameters

- **hdac:** DAC handle
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Trimming:** value : range: 0->31

DAC_DMAConvCpltCh2

Function name

void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh2

Function name

void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh2

Function name

void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

18.2 DACEx Firmware driver defines

The following section lists the various define and macros of the module.

18.2.1 DACEx

DACEx

DACEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0

Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1_0

Unmask DAC channel LFSR bit[1:0] for noise wave generation

DAC_LFSRUNMASK_BITS2_0

Unmask DAC channel LFSR bit[2:0] for noise wave generation

DAC_LFSRUNMASK_BITS3_0

Unmask DAC channel LFSR bit[3:0] for noise wave generation

DAC_LFSRUNMASK_BITS4_0

Unmask DAC channel LFSR bit[4:0] for noise wave generation

DAC_LFSRUNMASK_BITS5_0

Unmask DAC channel LFSR bit[5:0] for noise wave generation

DAC_LFSRUNMASK_BITS6_0

Unmask DAC channel LFSR bit[6:0] for noise wave generation

DAC_LFSRUNMASK_BITS7_0

Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0

Unmask DAC channel LFSR bit[8:0] for noise wave generation

DAC_LFSRUNMASK_BITS9_0

Unmask DAC channel LFSR bit[9:0] for noise wave generation

DAC_LFSRUNMASK_BITS10_0

Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0

Unmask DAC channel LFSR bit[11:0] for noise wave generation

DAC_TRIANGLEAMPLITUDE_1

Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUDE_3

Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUDE_7

Select max triangle amplitude of 7

DAC_TRIANGLEAMPLITUDE_15

Select max triangle amplitude of 15

DAC_TRIANGLEAMPLITUDE_31

Select max triangle amplitude of 31

DAC_TRIANGLEAMPLITUDE_63

Select max triangle amplitude of 63

DAC_TRIANGLEAMPLITUDE_127

Select max triangle amplitude of 127

DAC_TRIANGLEAMPLITUDE_255

Select max triangle amplitude of 255

DAC_TRIANGLEAMPLITUDE_511

Select max triangle amplitude of 511

DAC_TRIANGLEAMPLITUDE_1023

Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047

Select max triangle amplitude of 2047

DAC_TRIANGLEAMPLITUDE_4095

Select max triangle amplitude of 4095

19 HAL DCACHE Generic Driver

19.1 DCACHE Firmware driver registers structures

19.1.1 DCACHE_InitTypeDef

DCACHE_InitTypeDef is defined in the stm32h5xx_hal_dcache.h

Data Fields

- ***uint32_t ReadBurstType***

Field Documentation

- ***uint32_t DCACHE_InitTypeDef::ReadBurstType***
  Burst type to be applied for Data Cache This parameter can be a value of ***DCACHE_Read_Burst_Type***

19.1.2 __DCACHE_HandleTypeDef

__DCACHE_HandleTypeDef is defined in the stm32h5xx_hal_dcache.h

Data Fields

- ***DCACHE_TypeDef * Instance***
- ***DCACHE_InitTypeDef Init***
- ***void(* ErrorCallback***
- ***void(* CleanByAddrCallback***
- ***void(* InvalidateByAddrCallback***
- ***void(* InvalidateCompleteCallback***
- ***void(* CleanAndInvalidateByAddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***__IO HAL_DCACHE_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DCACHE_TypeDef* __DCACHE_HandleTypeDef::Instance***
  DCACHE register base address.
- ***DCACHE_InitTypeDef __DCACHE_HandleTypeDef::Init***
  DCACHE Initialization Structure.
- ***void(* __DCACHE_HandleTypeDef::ErrorCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::CleanByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::InvalidateByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::InvalidateCompleteCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::CleanAndInvalidateByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::MspInitCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::MspDeInitCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***__IO HAL_DCACHE_StateTypeDef __DCACHE_HandleTypeDef::State***
- ***__IO uint32_t __DCACHE_HandleTypeDef::ErrorCode***

19.2 DCACHE Firmware driver API description

The following section lists the various functions of the DCACHE library.

19.2.1 How to use this driver

1. Configure and enable the MPU to override default config if needed, please refers to ARM manual for default memory attribute. Then enable DCACHE.

- Use HAL_DCACHE_Invalidate() to invalidate the full cache content:
  - Cache content is lost, and reloaded when needed.
  - Used for complete invalidate of the DCACHE in case.
  - Blocking call until operation is done.
- Use HAL_DCACHE_InvalidateByAddr() to invalidate cache content for specific range:
  - Cache content for specific range is lost, and reloaded when needed.
  - Used when excepting a buffer to be updated by a peripheral (typically DMA transfer)
  - Blocking call until operation is done.
- Use HAL_DCACHE_CleanByAddr() to clean cache content for a specific range:
  - Cache content for specific range is written back to memory.
  - Used when buffer is updated by CPU before usage by a peripheral (typically DMA transfer)
  - Blocking call until operation is done.
- Use HAL_DCACHE_CleanInvalidateByAddr() to clean and invalidate cache content for a specific range:
  - Cache content for specific range is written back to memory, and reloaded when needed.
  - Used when sharing buffer between CPU and other peripheral.
  - Recommended to use for MPU reprogramming.
  - Blocking call until operation is done.

Interrupt mode IO operation

- Configure the DCACHE interrupt priority using HAL_NVIC_SetPriority()
- Enable the DCACHE IRQ handler using HAL_NVIC_EnableIRQ()
- Override weak definition for following callbacks (if needed):
  - HAL_DCACHE_CleanAndInvalidateByAddrCallback()
  - HAL_DCACHE_InvalidateCompleteCallback()
  - HAL_DCACHE_InvalidateByAddrCallback()
  - HAL_DCACHE_CleanByAddrCallback()
  - HAL_DCACHE_ErrorCallback()
- Use HAL_DCACHE_<COMMAND>_IT() to start a DCACHE operation with IT enabled.
- Use HAL_DCACHE_IRQHandler() called under DCACHEx_IRQHandler() Interrupt subroutine

Use HAL_DCACHE_GetState() function to return the DCACHE state and HAL_DCACHE_GetError() in case of error detection.

DCACHE HAL driver macros list

Below the list of macros defined in the DCACHE HAL driver.

- __HAL_DCACHE_ENABLE_IT : Enable DCACHE interrupts.
- __HAL_DCACHE_DISABLE_IT : Disable DCACHE interrupts.
- __HAL_DCACHE_GET_IT_SOURCE: Check whether the specified DCACHE interrupt source is enabled or not.
- __HAL_DCACHE_GET_FLAG : Check whether the selected DCACHE flag is set or not.
- __HAL_DCACHE_CLEAR_FLAG : Clear the selected DCACHE flags.

Note: You can refer to the header file of the DCACHE HAL driver for more useful macros.

19.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the DCACHEx peripheral:

- User must implement HAL_DCACHE_MspInit() function in which he configures all related peripherals resources (CLOCK, MPU, IT and NVIC).

- Call the function HAL_DCACHE_Init() to configure the selected device with the selected configuration:
  - ReadBurstType
- Call the function HAL_DCACHE_DeInit() to restore the reset configuration of the selected DCACHEx peripheral.

This section contains the following APIs:

- ***HAL_DCACHE_Init()***
- ***HAL_DCACHE_DeInit()***
- ***HAL_DCACHE_MspInit()***
- ***HAL_DCACHE_MspDeInit()***

19.2.3 IO operation functions

This section provides functions allowing to:

- Enable the Data cache.
- Disable the Data cache.
- Set Read Burst Type.
- Invalidate the Data cache.
- Invalidate the Data cache with interrupt.
- Clean the Data cache by Addr.
- Invalidate the Data cache by Addr.
- Clean and Invalidate the Data cache by Addr.
- Clean the Data cache by Addr with interrupt.
- Invalidate the Data cache by Addr with interrupt.
- Clean and Invalidate the Data cache by Addr with interrupt.
- Start the Data Cache performance monitoring.
- Stop the Data Cache performance monitoring.
- Reset the Data Cache performance monitoring values.
- Get the Data Cache performance Read Hit monitoring value.
- Get the Data Cache performance Read Miss monitoring value.
- Get the Data Cache performance Write Hit monitoring value.
- Get the Data Cache performance Write Miss monitoring value.

This section contains the following APIs:

- ***HAL_DCACHE_Enable()***
- ***HAL_DCACHE_Disable()***
- ***HAL_DCACHE_IsEnabled()***
- ***HAL_DCACHE_SetReadBurstType()***
- ***HAL_DCACHE_Invalidate()***
- ***HAL_DCACHE_InvalidateByAddr()***
- ***HAL_DCACHE_CleanByAddr()***
- ***HAL_DCACHE_CleanInvalidByAddr()***
- ***HAL_DCACHE_Invalidate_IT()***
- ***HAL_DCACHE_InvalidateByAddr_IT()***
- ***HAL_DCACHE_CleanByAddr_IT()***
- ***HAL_DCACHE_CleanInvalidByAddr_IT()***
- ***HAL_DCACHE_Monitor_Start()***
- ***HAL_DCACHE_Monitor_Stop()***
- ***HAL_DCACHE_Monitor_Reset()***
- ***HAL_DCACHE_Monitor_GetReadHitValue()***
- ***HAL_DCACHE_Monitor_GetReadMissValue()***
- ***HAL_DCACHE_Monitor_GetWriteHitValue()***
- ***HAL_DCACHE_Monitor_GetWriteMissValue()***

- *HAL_DCACHE_IRQHandler()*
- *HAL_DCACHE_RegisterCallback()*
- *HAL_DCACHE_UnRegisterCallback()*
- *HAL_DCACHE_CleanByAddrCallback()*
- *HAL_DCACHE_InvalidateByAddrCallback()*
- *HAL_DCACHE_CleanAndInvalidateByAddrCallback()*
- *HAL_DCACHE_InvalidateCompleteCallback()*
- *HAL_DCACHE_ErrorCallback()*

19.2.4 Peripheral State

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_DCACHE_GetState()*
- *HAL_DCACHE_GetError()*

19.2.5 Detailed description of functions

HAL_DCACHE_Init

Function name

HAL_StatusTypeDef HAL_DCACHE_Init (DCACHE_HandleTypeDef * hdcache)

Function description

Initializes the DCACHE according to the specified parameters in the DCACHE_InitTypeDef and initialize the associated handle.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHE.

Return values

- **HAL:** status

HAL_DCACHE_DeInit

Function name

HAL_StatusTypeDef HAL_DCACHE_DeInit (DCACHE_HandleTypeDef * hdcache)

Function description

DeInitialize the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_MspInit

Function name

void HAL_DCACHE_MspInit (DCACHE_HandleTypeDef * hdcache)

Function description

Initialize the DCACHE MSP.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_MspDeInit

Function name

void HAL_DCACHE_MspDeInit (DCACHE_HandleTypeDef * hdcache)

Function description

DeInitialize the DCACHE MSP.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_Enable

Function name

HAL_StatusTypeDef HAL_DCACHE_Enable (DCACHE_HandleTypeDef * hdcache)

Function description

Enable the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_Disable

Function name

HAL_StatusTypeDef HAL_DCACHE_Disable (DCACHE_HandleTypeDef * hdcache)

Function description

Disable the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_IsEnabled

Function name

uint32_t HAL_DCACHE_IsEnabled (const DCACHE_HandleTypeDef * hdcache)

Function description

Check whether the Data Cache is enabled or not.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Status:** (0: disabled, 1: enabled)

HAL_DCACHE_SetReadBurstType

Function name

HAL_StatusTypeDef HAL_DCACHE_SetReadBurstType (DCACHE_HandleTypeDef * hdcache, uint32_t ReadBurstType)

Function description

Set Read Burst Type.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **ReadBurstType:** Burst type to be applied for Data Cache DCACHE_READ_BURST_WRAP, DCACHE_READ_BURST_INC.

Return values

- **HAL:** status

HAL_DCACHE_Invalidate

Function name

HAL_StatusTypeDef HAL_DCACHE_Invalidate (DCACHE_HandleTypeDef * hdcache)

Function description

Invalidate the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

Notes

- This function waits for end of full cache invalidation

HAL_DCACHE_InvalidateByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_InvalidateByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Invalidate the Data cache for a specific region.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Invalidated
- **dSize:** Size of the region to be Invalidated(in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Invalidation

HAL_DCACHE_CleanByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean the Data cache by Addr.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned
- **dSize:** Size of the region to be Cleaned (in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Clean

HAL_DCACHE_CleanInvalidByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanInvalidByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean and Invalidate the Data cache by Addr.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned and Invalidated
- **dSize:** Size of the region to be Cleaned and Invalidated (in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Clean and Invalidation

HAL_DCACHE_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_Invalidate_IT (DCACHE_HandleTypeDef * hdcache)

Function description

Invalidate the Data cache with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_InvalidateByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_InvalidateByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Invalidate the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Invalidated
- **dSize:** Size of the region to be Invalidated

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_CleanByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned
- **dSize:** Size of the region to be Cleaned

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_CleanInvalidByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanInvalidByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean and Invalidate the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned and Invalidated
- **dSize:** Size of the region to be Cleaned and Invalidated

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_IRQHandler

Function name

void HAL_DCACHE_IRQHandler (DCACHE_HandleTypeDef * hdcache)

Function description

Handle the Data Cache interrupt request.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

Notes

- This API should be called under the DCACHEx_IRQHandler().

HAL_DCACHE_ErrorCallback

Function name

void HAL_DCACHE_ErrorCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Error callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_CleanByAddrCallback

Function name

void HAL_DCACHE_CleanByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache clean command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_InvalidateByAddrCallback

Function name

void HAL_DCACHE_InvalidateByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache Invalidate command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_InvalidateCompleteCallback

Function name

void HAL_DCACHE_InvalidateCompleteCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache full invalidation complete callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_CleanAndInvalidateByAddrCallback

Function name

void HAL_DCACHE_CleanAndInvalidateByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache clean and Invalidate command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCACHE_RegisterCallback (DCACHE_HandleTypeDef * hdcache, HAL_DCACHE_CallbackIDTypeDef CallbackID, pDCACHE_CallbackTypeDef pCallback)

Function description

Register a User DCACHE Callback To be used instead of the weak predefined callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCACHE_CLEAN_BY_ADDRESS_CB_ID Clean By Addr callback ID
  - HAL_DCACHE_INVALIDATE_BY_ADDRESS_CB_ID Invalidate By Addr callback ID
  - HAL_DCACHE_CLEAN_AND_INVALIDATE_BY_ADDRESS_CB_ID Clean and Invalidate By Addr callback ID
  - HAL_DCACHE_INVALIDATE_COMPLETE_CB_ID Invalidate Complete ID
  - HAL_DCACHE_ERROR_CB_ID Error callback ID
  - HAL_DCACHE_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCACHE_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_DCACHE_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCACHE_UnRegisterCallback (DCACHE_HandleTypeDef * hdcache, HAL_DCACHE_CallbackIDTypeDef CallbackID)

Function description

Unregister an DCACHE Callback DCACHE callback is redirected to the weak predefined callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DCACHE_CLEAN_BY_ADDRESS_CB_ID Clean By Addr callback ID
  - HAL_DCACHE_INVALIDATE_BY_ADDRESS_CB_ID Invalidate By Addr callback ID
  - HAL_DCACHE_CLEAN_AND_INVALIDATE_BY_ADDRESS_CB_ID Clean and Invalidate By Addr callback ID
  - HAL_DCACHE_INVALIDATE_COMPLETE_CB_ID Invalidate Complete callback ID
  - HAL_DCACHE_ERROR_CB_ID Error callback ID
  - HAL_DCACHE_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCACHE_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_DCACHE_Monitor_GetReadHitValue

Function name

uint32_t HAL_DCACHE_Monitor_GetReadHitValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Read Hit monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Read:** Hit monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetReadMissValue

Function name

uint32_t HAL_DCACHE_Monitor_GetReadMissValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Read Miss monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Read:** Miss monitoring value

Notes

- Upon reaching the 16-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetWriteHitValue

Function name

uint32_t HAL_DCACHE_Monitor_GetWriteHitValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Write Hit monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Write:** Hit monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetWriteMissValue

Function name

uint32_t HAL_DCACHE_Monitor_GetWriteMissValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Write Miss monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Write:** Miss monitoring value

Notes

- Upon reaching the 16-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Reset (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Reset the Data Cache performance monitoring values.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  - DCACHE_MONITOR_READ_HIT
  - DCACHE_MONITOR_READ_MISS
  - DCACHE_MONITOR_WRITE_HIT
  - DCACHE_MONITOR_WRITE_MISS
  - DCACHE_MONITOR_ALL

Return values

- **HAL:** status

HAL_DCACHE_Monitor_Start

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Start (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Start the Data Cache performance monitoring.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  - DCACHE_MONITOR_READ_HIT
  - DCACHE_MONITOR_READ_MISS
  - DCACHE_MONITOR_WRITE_HIT
  - DCACHE_MONITOR_WRITE_MISS
  - DCACHE_MONITOR_ALL

Return values

- **HAL:** status

HAL_DCACHE_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Stop (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Stop the Data Cache performance monitoring.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  - DCACHE_MONITOR_READ_HIT
  - DCACHE_MONITOR_READ_MISS
  - DCACHE_MONITOR_WRITE_HIT
  - DCACHE_MONITOR_WRITE_MISS
  - DCACHE_MONITOR_ALL

Return values

- **HAL:** status

Notes

- Stopping the monitoring does not reset the values.

HAL_DCACHE_GetState

Function name

HAL_DCACHE_StateTypeDef HAL_DCACHE_GetState (const DCACHE_HandleTypeDef * hdcache)

Function description

Return the DCACHE handle state.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** state

HAL_DCACHE_GetError

Function name

uint32_t HAL_DCACHE_GetError (const DCACHE_HandleTypeDef * hdcache)

Function description

Return the DCACHE error code.

Parameters

- **hdcache:** pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHE.

Return values

- **DCACHE:** Error Code

19.3 DCACHE Firmware driver defines

The following section lists the various define and macros of the module.

19.3.1 DCACHE

DCACHE

Command Operation

LL_DCACHE_COMMAND_NO_OPERATION

LL_DCACHE_COMMAND_CLEAN_BY_ADDR

LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR

LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR

DCACHE Error Code

HAL_DCACHE_ERROR_NONE

No error

HAL_DCACHE_ERROR_TIMEOUT

Timeout error

HAL_DCACHE_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DCACHE_ERROR_EVICTION_CLEAN

Eviction or clean operation write-back error

HAL_DCACHE_ERROR_INVALID_OPERATION

Invalid operation

DCACHE Exported Macros

__HAL_DCACHE_ENABLE_IT

Description:

- Enable DCACHE interrupts.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCACHE_IT_BUSYEND Busy end interrupt
  - DCACHE_IT_ERROR Cache error interrupt
  - DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_DCACHE_DISABLE_IT

Description:

- Disable DCACHE interrupts.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DCACHE_IT_BUSYEND Busy end interrupt
  - DCACHE_IT_ERROR Cache error interrupt
  - DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_DCACHE_GET_IT_SOURCE

Description:

- Check whether the specified DCACHE interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt source to check. This parameter can be any combination of the following values:
  - DCACHE_IT_BUSYEND Busy end interrupt
  - DCACHE_IT_ERROR Cache error interrupt
  - DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- The: state of __INTERRUPT__ (SET or RESET).

__HAL_DCACHE_GET_FLAG

Description:

- Check whether the selected DCACHE flag is set or not.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - DCACHE_FLAG_BUSY Busy flag
  - DCACHE_FLAG_BUSYEND Busy end flag
  - DCACHE_FLAG_ERROR Cache error flag
  - DCACHE_FLAG_BUSYCMD Cache Busy command flag
  - DCACHE_FLAG_CMDEND Cache command end flag

Return value:

- The: state of __FLAG__ (0 or 1).

__HAL_DCACHE_CLEAR_FLAG

Description:

- Clear the selected DCACHE flags.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __FLAG__: specifies the DCACHE flags to clear. This parameter can be any combination of the following values:
  - DCACHE_FLAG_BUSYEND Busy end flag
  - DCACHE_FLAG_ERROR Cache error flag
  - DCACHE_FLAG_CMDEND Cache command end flag

Flags

DCACHE_FLAG_BUSY

Busy flag

DCACHE_FLAG_BUSYEND

Busy end flag

DCACHE_FLAG_ERROR

Cache error flag

DCACHE_FLAG_BUSYCMD

Busy command flag

DCACHE_FLAG_CMDEND

Command end flag

Interrupts

DCACHE_IT_BUSYEND

Busy end interrupt

DCACHE_IT_ERROR

Cache error interrupt

DCACHE_IT_CMDEND

Command end interrupt

Monitor type

DCACHE_MONITOR_READ_HIT

Read Hit monitoring

DCACHE_MONITOR_READ_MISS

Read Miss monitoring

DCACHE_MONITOR_WRITE_HIT

Write Hit monitoring

DCACHE_MONITOR_WRITE_MISS

Write Miss monitoring

DCACHE_MONITOR_ALL

LL_DCACHE_MONITOR_READ_HIT

Read Hit monitoring

LL_DCACHE_MONITOR_READ_MISS

Read Miss monitoring

LL_DCACHE_MONITOR_WRITE_HIT

Write Hit monitoring

LL_DCACHE_MONITOR_WRITE_MISS

Write Miss monitoring

LL_DCACHE_MONITOR_ALL

Remapped Output burst type

DCACHE_READ_BURST_WRAP

WRAP

DCACHE_READ_BURST_INCR

INCR

LL_DCACHE_READ_BURST_WRAP

WRAP

LL_DCACHE_READ_BURST_INCR

INCR

20 HAL DCMI Generic Driver

20.1 DCMI Firmware driver registers structures

20.1.1 DCMI_CodesInitTypeDef

DCMI_CodesInitTypeDef is defined in the stm32h5xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMI_CodesInitTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

20.1.2 DCMI_SyncUnmaskTypeDef

DCMI_SyncUnmaskTypeDef is defined in the stm32h5xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

20.1.3 DCMI_InitTypeDef

DCMI_InitTypeDef is defined in the stm32h5xx_hal_dcmi.h

Data Fields

- ***uint32_t SynchroMode***
- ***uint32_t PCKPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t CaptureRate***
- ***uint32_t ExtendedDataMode***
- ***DCMI_CodesInitTypeDef SyncroCode***
- ***uint32_t JPEGMode***
- ***uint32_t ByteSelectMode***

- ***uint32_t ByteSelectStart***
- ***uint32_t LineSelectMode***
- ***uint32_t LineSelectStart***

Field Documentation

- ***uint32_t DCMI_InitTypeDef::SynchroMode***
  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of ***DCMI_Synchronization_Mode***
- ***uint32_t DCMI_InitTypeDef::PCKPolarity***
  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of ***DCMI_PIXCK_Polarity***
- ***uint32_t DCMI_InitTypeDef::VSPolarity***
  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of ***DCMI_VSYNC_Polarity***
- ***uint32_t DCMI_InitTypeDef::HSPolarity***
  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of ***DCMI_HSYNC_Polarity***
- ***uint32_t DCMI_InitTypeDef::CaptureRate***
  Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of ***DCMI_Capture_Rate***
- ***uint32_t DCMI_InitTypeDef::ExtendedDataMode***
  Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of ***DCMI_Extended_Data_Mode***
- ***DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode***
  Specifies the code of the line/frame start delimiter and the line/frame end delimiter
- ***uint32_t DCMI_InitTypeDef::JPEGMode***
  Enable or Disable the JPEG mode. This parameter can be a value of ***DCMI_MODE_JPEG***
- ***uint32_t DCMI_InitTypeDef::ByteSelectMode***
  Specifies the data to be captured by the interface This parameter can be a value of ***DCMI_Byte_Select_Mode***
- ***uint32_t DCMI_InitTypeDef::ByteSelectStart***
  Specifies if the data to be captured by the interface is even or odd This parameter can be a value of ***DCMI_Byte_Select_Start***
- ***uint32_t DCMI_InitTypeDef::LineSelectMode***
  Specifies the line of data to be captured by the interface This parameter can be a value of ***DCMI_Line_Select_Mode***
- ***uint32_t DCMI_InitTypeDef::LineSelectStart***
  Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of ***DCMI_Line_Select_Start***

20.1.4 __DCMI_HandleTypeDef

__DCMI_HandleTypeDef is defined in the stm32h5xx_hal_dcmi.h

Data Fields

- ***DCMI_TypeDef * Instance***
- ***DCMI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DCMI_StateTypeDef State***
- ***__IO uint32_t XferCount***
- ***__IO uint32_t XferSize***
- ***uint32_t XferTransferNumber***
- ***uint32_t pBuffPtr***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***__IO uint32_t ErrorCode***
- ***void(* FrameEventCallback***
- ***void(* VsyncEventCallback***

- ***void(* LineEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DCMI_TypeDef* __DCMI_HandleTypeDef::Instance***
  DCMI Register base address
- ***DCMI_InitTypeDef __DCMI_HandleTypeDef::Init***
  DCMI parameters
- ***HAL_LockTypeDef __DCMI_HandleTypeDef::Lock***
  DCMI locking object
- ***__IO HAL_DCMI_StateTypeDef __DCMI_HandleTypeDef::State***
  DCMI state
- ***__IO uint32_t __DCMI_HandleTypeDef::XferCount***
  DMA transfer counter
- ***__IO uint32_t __DCMI_HandleTypeDef::XferSize***
  DMA transfer size
- ***uint32_t __DCMI_HandleTypeDef::XferTransferNumber***
  DMA transfer number
- ***uint32_t __DCMI_HandleTypeDef::pBuffPtr***
  Pointer to DMA output buffer
- ***DMA_HandleTypeDef* __DCMI_HandleTypeDef::DMA_Handle***
  Pointer to the DMA handler
- ***__IO uint32_t __DCMI_HandleTypeDef::ErrorCode***
  DCMI Error code
- ***void(* __DCMI_HandleTypeDef::FrameEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Frame Event Callback
- ***void(* __DCMI_HandleTypeDef::VsyncEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Vsync Event Callback
- ***void(* __DCMI_HandleTypeDef::LineEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Line Event Callback
- ***void(* __DCMI_HandleTypeDef::ErrorCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Error Callback
- ***void(* __DCMI_HandleTypeDef::MspInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp Init callback
- ***void(* __DCMI_HandleTypeDef::MspDeInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp DeInit callback

20.2 DCMI Firmware driver API description

The following section lists the various functions of the DCMI library.

20.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using HAL_DCMI_Init() function.
2. Configure the selected DMA stream to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using HAL_DCMI_Start_DMA() function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using HAL_DCMI_ConfigCrop() and HAL_DCMI_EnableCrop() functions

5. The capture can be stopped using HAL_DCMI_Stop() function.
6. To control DCMI state you can use the function HAL_DCMI_GetState().

DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- __HAL_DCMI_ENABLE: Enable the DCMI peripheral.
- __HAL_DCMI_DISABLE: Disable the DCMI peripheral.
- __HAL_DCMI_GET_FLAG: Get the DCMI pending flags.
- __HAL_DCMI_CLEAR_FLAG: Clear the DCMI pending flags.
- __HAL_DCMI_ENABLE_IT: Enable the specified DCMI interrupts.
- __HAL_DCMI_DISABLE_IT: Disable the specified DCMI interrupts.
- __HAL_DCMI_GET_IT_SOURCE: Check whether the specified DCMI interrupt has occurred or not.

Note: You can refer to the DCMI HAL driver header file for more useful macros

Callback registration

20.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- ***HAL_DCMI_Init()***
- ***HAL_DCMI_DeInit()***
- ***HAL_DCMI_MspInit()***
- ***HAL_DCMI_MspDeInit()***
- ***HAL_DCMI_RegisterCallback()***
- ***HAL_DCMI_UnRegisterCallback()***

20.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- ***HAL_DCMI_Start_DMA()***
- ***HAL_DCMI_Stop()***
- ***HAL_DCMI_Suspend()***
- ***HAL_DCMI_Resume()***
- ***HAL_DCMI_IRQHandler()***
- ***HAL_DCMI_ErrorCallback()***
- ***HAL_DCMI_LineEventCallback()***
- ***HAL_DCMI_VsyncEventCallback()***
- ***HAL_DCMI_FrameEventCallback()***

20.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.
- Set embedded synchronization delimiters unmasks.

This section contains the following APIs:

- *HAL_DCMI_ConfigCrop()*
- *HAL_DCMI_DisableCrop()*
- *HAL_DCMI_EnableCrop()*
- *HAL_DCMI_ConfigSyncUnmask()*

20.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- *HAL_DCMI_GetState()*
- *HAL_DCMI_GetError()*
- *HAL_DCMI_RegisterCallback()*
- *HAL_DCMI_UnRegisterCallback()*

20.2.6 Detailed description of functions

HAL_DCMI_Init

Function name

HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI according to the specified parameters in the DCMI_InitTypeDef and create the associated handle.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DeInit

Function name

HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)

Function description

Deinitializes the DCMI peripheral registers to their default reset values.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_MspInit

Function name

void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_MspDeInit

Function name

void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)

Function description

DeInitializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_RegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID, pDCMI_CallbackTypeDef pCallback)

Function description

Register a User DCMI Callback To be used instead of the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_DCMI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_UnRegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID)

Function description

Unregister a DCMI Callback DCMI callback is redirected to the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_DCMI_Start_DMA

Function name

HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)

Function description

Enables DCMI DMA request and enables DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **DCMI_Mode:** DCMI capture mode snapshot or continuous grab.
- **pData:** The destination memory Buffer address (LCD Frame buffer).
- **Length:** The length of capture to be transferred.

Return values

- **HAL:** status

HAL_DCMI_Stop

Function name

HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable DCMI DMA request and Disable DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Suspend

Function name

HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)

Function description

Suspend DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Resume

Function name

HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)

Function description

Resume DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ErrorCallback

Function name

void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Error DCMI callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_LineEventCallback

Function name

void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Line Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_FrameEventCallback

Function name

void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Frame Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_VsyncEventCallback

Function name

void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

VSYNC Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_IRQHandler

Function name

void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)

Function description

Handles DCMI interrupt request.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values

- **None:**

HAL_DCMI_ConfigCrop

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)

Function description

Configure the DCMI CROP coordinate.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **YSize:** DCMI Line number
- **XSize:** DCMI Pixel per line
- **X0:** DCMI window X offset
- **Y0:** DCMI window Y offset

Return values

- **HAL:** status

HAL_DCMI_EnableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Enable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ConfigSyncUnmask

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigSyncUnmask (DCMI_HandleTypeDef * hdcmi, DCMI_SyncUnmaskTypeDef * SyncUnmask)

Function description

Set embedded synchronization delimiters unmasks.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **SyncUnmask:** pointer to a DCMI_SyncUnmaskTypeDef structure that contains the embedded synchronization delimiters unmasks.

Return values

- **HAL:** status

HAL_DCMI_GetState

Function name

HAL_DCMI_StateTypeDef HAL_DCMI_GetState (const DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI state.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** state

HAL_DCMI_GetError

Function name

uint32_t HAL_DCMI_GetError (const DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI error code.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **DCMI:** Error Code

20.3 DCMI Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 DCMI

DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL

Interface captures all received data

DCMI_BSM_OTHER

Interface captures every other byte from the received data

DCMI_BSM_ALTERNATE_4

Interface captures one byte out of four

DCMI_BSM_ALTERNATE_2

Interface captures two bytes out of four

DCMI Byte Select Start

DCMI_OEBS_ODD

Interface captures first data from the frame/line start, second one being dropped

DCMI_OEBS_EVEN

Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode

DCMI_MODE_CONTINUOUS

The received data are transferred continuously into the destination memory through the DMA

DCMI_MODE_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate

DCMI_CR_ALL_FRAME

All frames are captured

DCMI_CR_ALTERNATE_2_FRAME

Every alternate frame captured

DCMI_CR_ALTERNATE_4_FRAME

One frame in 4 frames captured

DCMI Error Code

HAL_DCMI_ERROR_NONE

No error

HAL_DCMI_ERROR_OVR

Overrun error

HAL_DCMI_ERROR_SYNC

Synchronization error

HAL_DCMI_ERROR_TIMEOUT

Timeout error

HAL_DCMI_ERROR_DMA

DMA error

HAL_DCMI_ERROR_INVALID_CALLBACK

Invalid callback error

DCMI Exported Macros

__HAL_DCMI_RESET_HANDLE_STATE

Description:

- Reset DCMI handle state.

Parameters:

- __HANDLE__: specifies the DCMI handle.

Return value:

- None

__HAL_DCMI_ENABLE

Description:

- Enable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_DISABLE

Description:

- Disable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_GET_FLAG

Description:

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
  - DCMI_FLAG_VSYNC: VSYNC pin state (active frame / synchronization between frames)
  - DCMI_FLAG_FNE: FIFO empty flag
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVRRI: Overrun flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask
  - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
  - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
  - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
  - DCMI_FLAG_VSYNCMI: DCMI VSYNC masked interrupt status
  - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flags.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVFRI: Overflow flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_DISABLE_IT

Description:

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_GET_IT_SOURCE

Description:

- Check whether the specified DCMI interrupt has occurred or not.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- The: state of INTERRUPT.

DCMI Extended Data Mode

DCMI_EXTEND_DATA_8B

Interface captures 8-bit data on every pixel clock

DCMI_EXTEND_DATA_10B

Interface captures 10-bit data on every pixel clock

DCMI_EXTEND_DATA_12B

Interface captures 12-bit data on every pixel clock

DCMI_EXTEND_DATA_14B

Interface captures 14-bit data on every pixel clock

DCMI Flags

DCMI_FLAG_HSYNC

HSYNC pin state (active line / synchronization between lines)

DCMI_FLAG_VSYNC

VSYNC pin state (active frame / synchronization between frames)

DCMI_FLAG_FNE

FIFO not empty flag

DCMI_FLAG_FRAMERI

Frame capture complete interrupt flag

DCMI_FLAG_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

VSYNC interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

DCMI Frame capture complete masked interrupt status

DCMI_FLAG_OVRMI

DCMI Overrun masked interrupt status

DCMI_FLAG_ERRMI

DCMI Synchronization error masked interrupt status

DCMI_FLAG_VSYNCMI

DCMI VSYNC masked interrupt status

DCMI_FLAG_LINEMI

DCMI Line masked interrupt status

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW

Horizontal synchronization active Low

DCMI_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMI interrupt sources

DCMI_IT_FRAME

Capture complete interrupt

DCMI_IT_OVR

Overrun interrupt

DCMI_IT_ERR

Synchronization error interrupt

DCMI_IT_VSYNC

VSYNC interrupt

DCMI_IT_LINE

Line interrupt

DCMI Line Select Mode

DCMI_LSM_ALL

Interface captures all received lines

DCMI_LSM_ALTERNATE_2

Interface captures one line out of two

DCMI Line Select Start

DCMI_OELS_ODD

Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN

Interface captures second line from the frame start, first one being dropped

DCMI Mis Index

DCMI_MIS_INDEX

DCMI MIS register index

DCMI MODE JPEG

DCMI_JPEG_DISABLE

Mode JPEG Disabled

DCMI_JPEG_ENABLE

Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMI SR Index

DCMI_SR_INDEX

DCMI SR register index

DCMI Stop Time Out

HAL_TIMEOUT_DCMI_STOP

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW

Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH

Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE

Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT

Window Height

21 HAL DMA Generic Driver

21.1 DMA Firmware driver registers structures

21.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32h5xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t SrcInc***
- ***uint32_t DestInc***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t Priority***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t TransferAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the DMA channel request. This parameter can be a value of ***DMA_Request_Selection***
- ***uint32_t DMA_InitTypeDef::BlkHWRequest***
  Specifies the Block hardware request mode for DMA channel. Block Hardware request feature can be used only with dedicated peripherals. This parameter can be a value of ***DMA_Block_Request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies the transfer direction for DMA channel. This parameter can be a value of ***DMA_Transfer_Direction***
- ***uint32_t DMA_InitTypeDef::SrcInc***
  Specifies the source increment mode for the DMA channel. This parameter can be a value of ***DMA_Source_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::DestInc***
  Specifies the destination increment mode for the DMA channel. This parameter can be a value of ***DMA_Destination_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::SrcDataWidth***
  Specifies the source data width for the DMA channel. This parameter can be a value of ***DMA_Source_Data_Width***
- ***uint32_t DMA_InitTypeDef::DestDataWidth***
  Specifies the destination data width for the DMA channel. This parameter can be a value of ***DMA_Destination_Data_Width***
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitTypeDef::SrcBurstLength***
  Specifies the source burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::DestBurstLength***
  Specifies the destination burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::TransferAllocatedPort***
  Specifies the transfer allocated ports. This parameter can be a combination of ***DMA_Transfer_Allocated_Port***

- ***uint32_t DMA_InitTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the transfer mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Mode***

21.1.2 DMA_InitLinkedListTypeDef

DMA_InitLinkedListTypeDef is defined in the stm32h5xx_hal_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t LinkedListMode***

Field Documentation

- ***uint32_t DMA_InitLinkedListTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkStepMode***
  Specifies the link step mode for the DMA channel. This parameter can be a value of ***DMAEx_Link_Step_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  Specifies the linked-list allocated port for the DMA channel. This parameter can be a value of ***DMAEx_Link_Allocated_Port***
- ***uint32_t DMA_InitLinkedListTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkedListMode***
  Specifies linked-list transfer mode for the DMA channel. This parameter can be a value of ***DMAEx_LinkedList_Mode***

21.1.3 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32h5xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***DMA_InitLinkedListTypeDef InitLinkedList***
- ***HAL_LockTypeDef Lock***
- ***uint32_t Mode***
- ***__IO HAL_DMA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***void(* XferSuspendCallback***
- ***struct __DMA_QListTypeDef * LinkedListQueue***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register the DMA channel base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA channel init parameters

- ***DMA_InitLinkedListTypeDef __DMA_HandleTypeDef::InitLinkedList***
  DMA channel linked-list init parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***uint32_t __DMA_HandleTypeDef::Mode***
  DMA transfer mode
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA error code
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Abort callback
- ***void(* __DMA_HandleTypeDef::XferSuspendCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Suspend callback
- ***struct __DMA_QListTypeDef* __DMA_HandleTypeDef::LinkedListQueue***
  DMA linked-list queue

21.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

21.2.1

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.2 How to use this driver

21.2.3

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.4

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.5 Initialization and de-initialization functions

21.2.6

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.

- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.7

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.8 IO operation functions

21.2.9

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.10

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.11 State and Errors functions

21.2.12

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.13

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.

- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.14 DMA Attributes functions

21.2.15

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- ***HAL_DMA_ConfigChannelAttributes()***
- ***HAL_DMA_GetConfigChannelAttributes()***
- ***HAL_DMA_LockChannelAttributes()***
- ***HAL_DMA_GetLockChannelAttributes()***

21.2.16 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in normal mode according to the specified parameters in the DMA_InitTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_DeInit

Function name

HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in normal mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_Start

Function name

HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Start the DMA channel transfer in normal mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Starts the DMA channel transfer in normal mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a wait until the DMA channel is effectively stopped is added. If a channel is suspended while a data transfer is on-going, the current data will be transferred and the channel will be effectively suspended only after the transfer of any on-going data is finished.

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer in interrupt mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef *const hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer status (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CompleteLevel:** : Specifies the DMA level complete.
- **Timeout:** : Timeout duration.

Return values

- **HAL:** status

HAL_DMA_IRQHandler

Function name

void HAL_DMA_IRQHandler (DMA_HandleTypeDef *const hdma)

Function description

Handle DMA interrupt request (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **None.:**

HAL_DMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *const _hdma) pCallback)

Function description

Register callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enumeration.
- **pCallback:** : Pointer to private callback function.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_RegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

Unregister callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enum.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_UnRegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to un-register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef const *const hdma)

Function description

Returns the DMA channel state.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** state.

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (DMA_HandleTypeDef const *const hdma)

Function description

Return the DMA channel error code.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** Error Code.

HAL_DMA_ConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_ConfigChannelAttributes (DMA_HandleTypeDef *const hdma, uint32_t ChannelAttributes)

Function description

Configure the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **ChannelAttributes:** : Specifies the DMA channel secure/privilege attributes. This parameter can be a one or a combination of DMA Channel Attributes.

Return values

- **HAL:** Status.

Notes

- These attributes cannot be modified when the corresponding lock state is enabled.

HAL_DMA_GetConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetConfigChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pChannelAttributes)

Function description

Get the DMA channel security and privilege attributes.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pChannelAttributes:** : Pointer to the returned attributes.

Return values

- **HAL:** Status.

HAL_DMA_LockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_LockChannelAttributes (DMA_HandleTypeDef const *const hdma)

Function description

Lock the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** Status.

HAL_DMA_GetLockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetLockChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pLockState)

Function description

Get the security and privilege attribute lock state of a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pLockState:** : Pointer to lock state (returned value can be DMA_CHANNEL_ATTRIBUTE_UNLOCKED or DMA_CHANNEL_ATTRIBUTE_LOCKED).

Return values

- **HAL:** status.

21.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 DMA

DMA

DMA Block Request

DMA_BREQ_SINGLE_BURST

Hardware request protocol at a single / burst level

DMA_BREQ_BLOCK

Hardware request protocol at a block level

DMA Channel Attributes

DMA_CHANNEL_PRIV

Channel is privileged

DMA_CHANNEL_NPRIV

Channel is unprivileged

DMA_CHANNEL_SEC

Channel is secure

DMA_CHANNEL_NSEC

Channel is non-secure

DMA_CHANNEL_SRC_SEC

Channel source is secure

DMA_CHANNEL_SRC_NSEC

Channel source is non-secure

DMA_CHANNEL_DEST_SEC

Channel destination is secure

DMA_CHANNEL_DEST_NSEC

Channel destination is non-secure

DMA_CHANNEL_ATTRIBUTE_UNLOCKED

Channel attribute is unlocked

DMA_CHANNEL_ATTRIBUTE_LOCKED

Channel attribute is locked

DMA destination Data Width

DMA_DEST_DATAWIDTH_BYTE

Destination data width : Byte

DMA_DEST_DATAWIDTH_HALFWORD

Destination data width : HalfWord

DMA_DEST_DATAWIDTH_WORD

Destination data width : Word

DMA Destination Increment Mode

DMA_DINC_FIXED

Destination fixed single / burst

DMA_DINC_INCREMENTED

Destination incremented single / burst

DMA Error Codes

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_DTE

Data transfer error

HAL_DMA_ERROR_ULE

Update linked-list item error

HAL_DMA_ERROR_USE

User setting error

HAL_DMA_ERROR_TO

Trigger overrun error

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_NO_XFER

No transfer ongoing error

HAL_DMA_ERROR_BUSY

Busy error

HAL_DMA_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None.

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_GET_FLAG

Description:

- Get the DMA channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update linked-list Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.
  - DMA_FLAG_IDLEF : Idle flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update Linked-List Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : Specifies the DMA interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Checks whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt source to check.
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_SET_COUNTER

Description:

- Writes the block number of bytes to be transferred from the source on the DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.
- __COUNTER__: : Number of data bytes to be transferred from the source (from 0 to 65535).

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data bytes in the current DMA Channel transfer.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- The: number of remaining data units in the current DMA Stream transfer.

DMA Flag Definition

DMA_FLAG_IDLE

Idle flag

DMA_FLAG_TC

Transfer complete flag

DMA_FLAG_HT

Half transfer complete flag

DMA_FLAG_DTE

Data transfer error flag

DMA_FLAG_ULE

Update linked-list item error flag

DMA_FLAG_USE

User setting error flag

DMA_FLAG_SUSP

Completed suspension flag

DMA_FLAG_TO

Trigger overrun flag

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION

Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER

Index of the DMA handle used for Trigger DMA requests

DMA Interrupt Enable Definition

DMA_IT_TC

Transfer complete interrupt

DMA_IT_HT

Half transfer complete interrupt

DMA_IT_DTE

Data transfer error interrupt

DMA_IT_ULE

Update linked-list item error interrupt

DMA_IT_USE

User eetting error interrupt

DMA_IT_SUSP

Completed suspension interrupt

DMA_IT_TO

Trigger overrun interrupt

DMA Priority Level

DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low weight

DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid weight

DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High weight

DMA_HIGH_PRIORITY

Priority level : HIGH Priority

DMA Request Selection

GPDMA1_REQUEST_ADC1

GPDMA1 HW request is ADC1

GPDMA1_REQUEST_ADC2

GPDMA1 HW request is ADC2

GPDMA1_REQUEST_DAC1_CH1

GPDMA1 HW request is DAC1_CH1

GPDMA1_REQUEST_DAC1_CH2

GPDMA1 HW request is DAC1_CH2

GPDMA1_REQUEST_TIM6_UP

GPDMA1 HW request is TIM6_UP

GPDMA1_REQUEST_TIM7_UP

GPDMA1 HW request is TIM7_UP

GPDMA1_REQUEST_SPI1_RX

GPDMA1 HW request is SPI1_RX

GPDMA1_REQUEST_SPI1_TX

GPDMA1 HW request is SPI1_TX

GPDMA1_REQUEST_SPI2_RX

GPDMA1 HW request is SPI2_RX

GPDMA1_REQUEST_SPI2_TX

GPDMA1 HW request is SPI2_TX

GPDMA1_REQUEST_SPI3_RX

GPDMA1 HW request is SPI3_RX

GPDMA1_REQUEST_SPI3_TX

GPDMA1 HW request is SPI3_TX

GPDMA1_REQUEST_I2C1_RX

GPDMA1 HW request is I2C1_RX

GPDMA1_REQUEST_I2C1_TX

GPDMA1 HW request is I2C1_TX

GPDMA1_REQUEST_I2C2_RX

GPDMA1 HW request is I2C2_RX

GPDMA1_REQUEST_I2C2_TX

GPDMA1 HW request is I2C2_TX

GPDMA1_REQUEST_I2C3_RX

GPDMA1 HW request is I2C3_RX

GPDMA1_REQUEST_I2C3_TX

GPDMA1 HW request is I2C3_TX

GPDMA1_REQUEST_USART1_RX

GPDMA1 HW request is USART1_RX

GPDMA1_REQUEST_USART1_TX

GPDMA1 HW request is USART1_TX

GPDMA1_REQUEST_USART2_RX

GPDMA1 HW request is USART2_RX

GPDMA1_REQUEST_USART2_TX

GPDMA1 HW request is USART2_TX

GPDMA1_REQUEST_USART3_RX

GPDMA1 HW request is USART3_RX

GPDMA1_REQUEST_USART3_TX

GPDMA1 HW request is USART3_TX

GPDMA1_REQUEST_UART4_RX

GPDMA1 HW request is UART4_RX

GPDMA1_REQUEST_UART4_TX

GPDMA1 HW request is UART4_TX

GPDMA1_REQUEST_UART5_RX

GPDMA1 HW request is UART5_RX

GPDMA1_REQUEST_UART5_TX

GPDMA1 HW request is UART5_TX

GPDMA1_REQUEST_USART6_RX

GPDMA1 HW request is USART6_RX

GPDMA1_REQUEST_USART6_TX

GPDMA1 HW request is USART6_TX

GPDMA1_REQUEST_UART7_RX

GPDMA1 HW request is UART7_RX

GPDMA1_REQUEST_UART7_TX

GPDMA1 HW request is UART7_TX

GPDMA1_REQUEST_UART8_RX

GPDMA1 HW request is UART8_RX

GPDMA1_REQUEST_UART8_TX

GPDMA1 HW request is UART8_TX

GPDMA1_REQUEST_UART9_RX

GPDMA1 HW request is UART9_RX

GPDMA1_REQUEST_UART9_TX

GPDMA1 HW request is UART9_TX

GPDMA1_REQUEST_USART10_RX

GPDMA1 HW request is USART10_RX

GPDMA1_REQUEST_USART10_TX

GPDMA1 HW request is USART10_TX

GPDMA1_REQUEST_USART11_RX

GPDMA1 HW request is USART11_RX

GPDMA1_REQUEST_USART11_TX

GPDMA1 HW request is USART11_TX

GPDMA1_REQUEST_UART12_RX

GPDMA1 HW request is UART12_RX

GPDMA1_REQUEST_UART12_TX

GPDMA1 HW request is UART12_TX

GPDMA1_REQUEST_LPUART1_RX

GPDMA1 HW request is LPUART1_RX

GPDMA1_REQUEST_LPUART1_TX

GPDMA1 HW request is LPUART1_TX

GPDMA1_REQUEST_SPI4_RX

GPDMA1 HW request is SPI4_RX

GPDMA1_REQUEST_SPI4_TX

GPDMA1 HW request is SPI4_TX

GPDMA1_REQUEST_SPI5_RX

GPDMA1 HW request is SPI5_RX

GPDMA1_REQUEST_SPI5_TX

GPDMA1 HW request is SPI5_TX

GPDMA1_REQUEST_SPI6_RX

GPDMA1 HW request is SPI6_RX

GPDMA1_REQUEST_SPI6_TX

GPDMA1 HW request is SPI6_TX

GPDMA1_REQUEST_SAI1_A

GPDMA1 HW request is SAI1_A

GPDMA1_REQUEST_SAI1_B

GPDMA1 HW request is SAI1_B

GPDMA1_REQUEST_SAI2_A

GPDMA1 HW request is SAI2_A

GPDMA1_REQUEST_SAI2_B

GPDMA1 HW request is SAI2_B

GPDMA1_REQUEST_OCTOSPI1

GPDMA1 HW request is OCTOSPI1

GPDMA1_REQUEST_TIM1_CH1

GPDMA1 HW request is TIM1_CH1

GPDMA1_REQUEST_TIM1_CH2

GPDMA1 HW request is TIM1_CH2

GPDMA1_REQUEST_TIM1_CH3

GPDMA1 HW request is TIM1_CH3

GPDMA1_REQUEST_TIM1_CH4

GPDMA1 HW request is TIM1_CH4

GPDMA1_REQUEST_TIM1_UP

GPDMA1 HW request is TIM1_UP

GPDMA1_REQUEST_TIM1_TRIG

GPDMA1 HW request is TIM1_TRIG

GPDMA1_REQUEST_TIM1_COM

GPDMA1 HW request is TIM1_COM

GPDMA1_REQUEST_TIM8_CH1

GPDMA1 HW request is TIM8_CH1

GPDMA1_REQUEST_TIM8_CH2

GPDMA1 HW request is TIM8_CH2

GPDMA1_REQUEST_TIM8_CH3

GPDMA1 HW request is TIM8_CH3

GPDMA1_REQUEST_TIM8_CH4

GPDMA1 HW request is TIM8_CH4

GPDMA1_REQUEST_TIM8_UP

GPDMA1 HW request is TIM8_UP

GPDMA1_REQUEST_TIM8_TRIG

GPDMA1 HW request is TIM8_TRIG

GPDMA1_REQUEST_TIM8_COM

GPDMA1 HW request is TIM8_COM

GPDMA1_REQUEST_TIM2_CH1

GPDMA1 HW request is TIM2_CH1

GPDMA1_REQUEST_TIM2_CH2

GPDMA1 HW request is TIM2_CH2

GPDMA1_REQUEST_TIM2_CH3

GPDMA1 HW request is TIM2_CH3

GPDMA1_REQUEST_TIM2_CH4

GPDMA1 HW request is TIM2_CH4

GPDMA1_REQUEST_TIM2_UP

GPDMA1 HW request is TIM2_UP

GPDMA1_REQUEST_TIM3_CH1

GPDMA1 HW request is TIM3_CH1

GPDMA1_REQUEST_TIM3_CH2

GPDMA1 HW request is TIM3_CH2

GPDMA1_REQUEST_TIM3_CH3

GPDMA1 HW request is TIM3_CH3

GPDMA1_REQUEST_TIM3_CH4

GPDMA1 HW request is TIM3_CH4

GPDMA1_REQUEST_TIM3_UP

GPDMA1 HW request is TIM3_UP

GPDMA1_REQUEST_TIM3_TRIG

GPDMA1 HW request is TIM3_TRIG

GPDMA1_REQUEST_TIM4_CH1

GPDMA1 HW request is TIM4_CH1

GPDMA1_REQUEST_TIM4_CH2

GPDMA1 HW request is TIM4_CH2

GPDMA1_REQUEST_TIM4_CH3

GPDMA1 HW request is TIM4_CH3

GPDMA1_REQUEST_TIM4_CH4

GPDMA1 HW request is TIM4_CH4

GPDMA1_REQUEST_TIM4_UP

GPDMA1 HW request is TIM4_UP

GPDMA1_REQUEST_TIM5_CH1

GPDMA1 HW request is TIM5_CH1

GPDMA1_REQUEST_TIM5_CH2

GPDMA1 HW request is TIM5_CH2

GPDMA1_REQUEST_TIM5_CH3

GPDMA1 HW request is TIM5_CH3

GPDMA1_REQUEST_TIM5_CH4

GPDMA1 HW request is TIM5_CH4

GPDMA1_REQUEST_TIM5_UP

GPDMA1 HW request is TIM5_UP

GPDMA1_REQUEST_TIM5_TRIG

GPDMA1 HW request is TIM5_TRIG

GPDMA1_REQUEST_TIM15_CH1

GPDMA1 HW request is TIM15_CH1

GPDMA1_REQUEST_TIM15_UP

GPDMA1 HW request is TIM15_UP

GPDMA1_REQUEST_TIM15_TRIG

GPDMA1 HW request is TIM15_TRIG

GPDMA1_REQUEST_TIM15_COM

GPDMA1 HW request is TIM15_COM

GPDMA1_REQUEST_TIM16_CH1

GPDMA1 HW request is TIM16_CH1

GPDMA1_REQUEST_TIM16_UP

GPDMA1 HW request is TIM16_UP

GPDMA1_REQUEST_TIM17_CH1

GPDMA1 HW request is TIM17_CH1

GPDMA1_REQUEST_TIM17_UP

GPDMA1 HW request is TIM17_UP

GPDMA1_REQUEST_LPTIM1_IC1

GPDMA1 HW request is LPTIM1_IC1

GPDMA1_REQUEST_LPTIM1_IC2

GPDMA1 HW request is LPTIM1_IC2

GPDMA1_REQUEST_LPTIM1_UE

GPDMA1 HW request is LPTIM1_UE

GPDMA1_REQUEST_LPTIM2_IC1

GPDMA1 HW request is LPTIM2_IC1

GPDMA1_REQUEST_LPTIM2_IC2

GPDMA1 HW request is LPTIM2_IC2

GPDMA1_REQUEST_LPTIM2_UE

GPDMA1 HW request is LPTIM2_UE

GPDMA1_REQUEST_DCMI

GPDMA1 HW request is DCMI

GPDMA1_REQUEST_AES_OUT

GPDMA1 HW request is AES_OUT

GPDMA1_REQUEST_AES_IN

GPDMA1 HW request is AES_IN

GPDMA1_REQUEST_HASH_IN

GPDMA1 HW request is HASH_IN

GPDMA1_REQUEST_UCPD1_RX

GPDMA1 HW request is UCPD1_RX

GPDMA1_REQUEST_UCPD1_TX

GPDMA1 HW request is UCPD1_TX

GPDMA1_REQUEST_CORDIC_READ

GPDMA1 HW request is CORDIC_READ

GPDMA1_REQUEST_CORDIC_WRITE

GPDMA1 HW request is CORDIC_WRITE

GPDMA1_REQUEST_FMAC_READ

GPDMA1 HW request is FMAC_READ

GPDMA1_REQUEST_FMAC_WRITE

GPDMA1 HW request is FMAC_WRITE

GPDMA1_REQUEST_SAES_OUT

GPDMA1 HW request is SAES_OUT

GPDMA1_REQUEST_SAES_IN

GPDMA1 HW request is SAES_IN

GPDMA1_REQUEST_I3C1_RX

GPDMA1 HW request is I3C1_RX

GPDMA1_REQUEST_I3C1_TX

GPDMA1 HW request is I3C1_TX

GPDMA1_REQUEST_I3C1_TC

GPDMA1 HW request is I3C1_TC

GPDMA1_REQUEST_I3C1_RS

GPDMA1 HW request is I3C1_RS

GPDMA1_REQUEST_I2C4_RX

GPDMA1 HW request is I2C4_RX

GPDMA1_REQUEST_I2C4_TX

GPDMA1 HW request is I2C4_TX

GPDMA1_REQUEST_LPTIM3_IC1

GPDMA1 HW request is LPTIM3_IC1

GPDMA1_REQUEST_LPTIM3_IC2

GPDMA1 HW request is LPTIM3_IC2

GPDMA1_REQUEST_LPTIM3_UE

GPDMA1 HW request is LPTIM3_UE

GPDMA1_REQUEST_LPTIM5_IC1

GPDMA1 HW request is LPTIM5_IC1

GPDMA1_REQUEST_LPTIM5_IC2

GPDMA1 HW request is LPTIM5_IC2

GPDMA1_REQUEST_LPTIM5_UE

GPDMA1 HW request is LPTIM5_UE

GPDMA1_REQUEST_LPTIM6_IC1

GPDMA1 HW request is LPTIM6_IC1

GPDMA1_REQUEST_LPTIM6_IC2

GPDMA1 HW request is LPTIM6_IC2

GPDMA1_REQUEST_LPTIM6_UE

GPDMA1 HW request is LPTIM6_UE

GPDMA2_REQUEST_ADC1

GPDMA2 HW request is ADC1

GPDMA2_REQUEST_ADC2

GPDMA2 HW request is ADC2

GPDMA2_REQUEST_DAC1_CH1

GPDMA2 HW request is DAC1_CH1

GPDMA2_REQUEST_DAC1_CH2

GPDMA2 HW request is DAC1_CH2

GPDMA2_REQUEST_TIM6_UP

GPDMA2 HW request is TIM6_UP

GPDMA2_REQUEST_TIM7_UP

GPDMA2 HW request is TIM7_UP

GPDMA2_REQUEST_SPI1_RX

GPDMA2 HW request is SPI1_RX

GPDMA2_REQUEST_SPI1_TX

GPDMA2 HW request is SPI1_TX

GPDMA2_REQUEST_SPI2_RX

GPDMA2 HW request is SPI2_RX

GPDMA2_REQUEST_SPI2_TX

GPDMA2 HW request is SPI2_TX

GPDMA2_REQUEST_SPI3_RX

GPDMA2 HW request is SPI3_RX

GPDMA2_REQUEST_SPI3_TX

GPDMA2 HW request is SPI3_TX

GPDMA2_REQUEST_I2C1_RX

GPDMA2 HW request is I2C1_RX

GPDMA2_REQUEST_I2C1_TX

GPDMA2 HW request is I2C1_TX

GPDMA2_REQUEST_I2C2_RX

GPDMA2 HW request is I2C2_RX

GPDMA2_REQUEST_I2C2_TX

GPDMA2 HW request is I2C2_TX

GPDMA2_REQUEST_I2C3_RX

GPDMA2 HW request is I2C3_RX

GPDMA2_REQUEST_I2C3_TX

GPDMA2 HW request is I2C3_TX

GPDMA2_REQUEST_USART1_RX

GPDMA2 HW request is USART1_RX

GPDMA2_REQUEST_USART1_TX

GPDMA2 HW request is USART1_TX

GPDMA2_REQUEST_USART2_RX

GPDMA2 HW request is USART2_RX

GPDMA2_REQUEST_USART2_TX

GPDMA2 HW request is USART2_TX

GPDMA2_REQUEST_USART3_RX

GPDMA2 HW request is USART3_RX

GPDMA2_REQUEST_USART3_TX

GPDMA2 HW request is USART3_TX

GPDMA2_REQUEST_UART4_RX

GPDMA2 HW request is UART4_RX

GPDMA2_REQUEST_UART4_TX

GPDMA2 HW request is UART4_TX

GPDMA2_REQUEST_UART5_RX

GPDMA2 HW request is UART5_RX

GPDMA2_REQUEST_UART5_TX

GPDMA2 HW request is UART5_TX

GPDMA2_REQUEST_USART6_RX

GPDMA2 HW request is USART6_RX

GPDMA2_REQUEST_USART6_TX

GPDMA2 HW request is USART6_TX

GPDMA2_REQUEST_UART7_RX

GPDMA2 HW request is UART7_RX

GPDMA2_REQUEST_UART7_TX

GPDMA2 HW request is UART7_TX

GPDMA2_REQUEST_UART8_RX

GPDMA2 HW request is UART8_RX

GPDMA2_REQUEST_UART8_TX

GPDMA2 HW request is UART8_TX

GPDMA2_REQUEST_UART9_RX

GPDMA2 HW request is UART9_RX

GPDMA2_REQUEST_UART9_TX

GPDMA2 HW request is UART9_TX

GPDMA2_REQUEST_USART10_RX

GPDMA2 HW request is USART10_RX

GPDMA2_REQUEST_USART10_TX

GPDMA2 HW request is USART10_TX

GPDMA2_REQUEST_USART11_RX

GPDMA2 HW request is USART11_RX

GPDMA2_REQUEST_USART11_TX

GPDMA2 HW request is USART11_TX

GPDMA2_REQUEST_UART12_RX

GPDMA2 HW request is UART12_RX

GPDMA2_REQUEST_UART12_TX

GPDMA2 HW request is UART12_TX

GPDMA2_REQUEST_LPUART1_RX

GPDMA2 HW request is LPUART1_RX

GPDMA2_REQUEST_LPUART1_TX

GPDMA2 HW request is LPUART1_TX

GPDMA2_REQUEST_SPI4_RX

GPDMA2 HW request is SPI4_RX

GPDMA2_REQUEST_SPI4_TX

GPDMA2 HW request is SPI4_TX

GPDMA2_REQUEST_SPI5_RX

GPDMA2 HW request is SPI5_RX

GPDMA2_REQUEST_SPI5_TX

GPDMA2 HW request is SPI5_TX

GPDMA2_REQUEST_SPI6_RX

GPDMA2 HW request is SPI6_RX

GPDMA2_REQUEST_SPI6_TX

GPDMA2 HW request is SPI6_TX

GPDMA2_REQUEST_SAI1_A

GPDMA2 HW request is SAI1_A

GPDMA2_REQUEST_SAI1_B

GPDMA2 HW request is SAI1_B

GPDMA2_REQUEST_SAI2_A

GPDMA2 HW request is SAI2_A

GPDMA2_REQUEST_SAI2_B

GPDMA2 HW request is SAI2_B

GPDMA2_REQUEST_OCTOSPI1

GPDMA2 HW request is OCTOSPI1

GPDMA2_REQUEST_TIM1_CH1

GPDMA2 HW request is TIM1_CH1

GPDMA2_REQUEST_TIM1_CH2

GPDMA2 HW request is TIM1_CH2

GPDMA2_REQUEST_TIM1_CH3

GPDMA2 HW request is TIM1_CH3

GPDMA2_REQUEST_TIM1_CH4

GPDMA2 HW request is TIM1_CH4

GPDMA2_REQUEST_TIM1_UP

GPDMA2 HW request is TIM1_UP

GPDMA2_REQUEST_TIM1_TRIG

GPDMA2 HW request is TIM1_TRIG

GPDMA2_REQUEST_TIM1_COM

GPDMA2 HW request is TIM1_COM

GPDMA2_REQUEST_TIM8_CH1

GPDMA2 HW request is TIM8_CH1

GPDMA2_REQUEST_TIM8_CH2

GPDMA2 HW request is TIM8_CH2

GPDMA2_REQUEST_TIM8_CH3

GPDMA2 HW request is TIM8_CH3

GPDMA2_REQUEST_TIM8_CH4

GPDMA2 HW request is TIM8_CH4

GPDMA2_REQUEST_TIM8_UP

GPDMA2 HW request is TIM8_UP

GPDMA2_REQUEST_TIM8_TRIG

GPDMA2 HW request is TIM8_TRIG

GPDMA2_REQUEST_TIM8_COM

GPDMA2 HW request is TIM8_COM

GPDMA2_REQUEST_TIM2_CH1

GPDMA2 HW request is TIM2_CH1

GPDMA2_REQUEST_TIM2_CH2

GPDMA2 HW request is TIM2_CH2

GPDMA2_REQUEST_TIM2_CH3

GPDMA2 HW request is TIM2_CH3

GPDMA2_REQUEST_TIM2_CH4

GPDMA2 HW request is TIM2_CH4

GPDMA2_REQUEST_TIM2_UP

GPDMA2 HW request is TIM2_UP

GPDMA2_REQUEST_TIM3_CH1

GPDMA2 HW request is TIM3_CH1

GPDMA2_REQUEST_TIM3_CH2

GPDMA2 HW request is TIM3_CH2

GPDMA2_REQUEST_TIM3_CH3

GPDMA2 HW request is TIM3_CH3

GPDMA2_REQUEST_TIM3_CH4

GPDMA2 HW request is TIM3_CH4

GPDMA2_REQUEST_TIM3_UP

GPDMA2 HW request is TIM3_UP

GPDMA2_REQUEST_TIM3_TRIG

GPDMA2 HW request is TIM3_TRIG

GPDMA2_REQUEST_TIM4_CH1

GPDMA2 HW request is TIM4_CH1

GPDMA2_REQUEST_TIM4_CH2

GPDMA2 HW request is TIM4_CH2

GPDMA2_REQUEST_TIM4_CH3

GPDMA2 HW request is TIM4_CH3

GPDMA2_REQUEST_TIM4_CH4

GPDMA2 HW request is TIM4_CH4

GPDMA2_REQUEST_TIM4_UP

GPDMA2 HW request is TIM4_UP

GPDMA2_REQUEST_TIM5_CH1

GPDMA2 HW request is TIM5_CH1

GPDMA2_REQUEST_TIM5_CH2

GPDMA2 HW request is TIM5_CH2

GPDMA2_REQUEST_TIM5_CH3

GPDMA2 HW request is TIM5_CH3

GPDMA2_REQUEST_TIM5_CH4

GPDMA2 HW request is TIM5_CH4

GPDMA2_REQUEST_TIM5_UP

GPDMA2 HW request is TIM5_UP

GPDMA2_REQUEST_TIM5_TRIG

GPDMA2 HW request is TIM5_TRIG

GPDMA2_REQUEST_TIM15_CH1

GPDMA2 HW request is TIM15_CH1

GPDMA2_REQUEST_TIM15_UP

GPDMA2 HW request is TIM15_UP

GPDMA2_REQUEST_TIM15_TRIG

GPDMA2 HW request is TIM15_TRIG

GPDMA2_REQUEST_TIM15_COM

GPDMA2 HW request is TIM15_COM

GPDMA2_REQUEST_TIM16_CH1

GPDMA2 HW request is TIM16_CH1

GPDMA2_REQUEST_TIM16_UP

GPDMA2 HW request is TIM16_UP

GPDMA2_REQUEST_TIM17_CH1

GPDMA2 HW request is TIM17_CH1

GPDMA2_REQUEST_TIM17_UP

GPDMA2 HW request is TIM17_UP

GPDMA2_REQUEST_LPTIM1_IC1

GPDMA2 HW request is LPTIM1_IC1

GPDMA2_REQUEST_LPTIM1_IC2

GPDMA2 HW request is LPTIM1_IC2

GPDMA2_REQUEST_LPTIM1_UE

GPDMA2 HW request is LPTIM1_UE

GPDMA2_REQUEST_LPTIM2_IC1

GPDMA2 HW request is LPTIM2_IC1

GPDMA2_REQUEST_LPTIM2_IC2

GPDMA2 HW request is LPTIM2_IC2

GPDMA2_REQUEST_LPTIM2_UE

GPDMA2 HW request is LPTIM2_UE

GPDMA2_REQUEST_DCMI

GPDMA2 HW request is DCMI

GPDMA2_REQUEST_AES_OUT

GPDMA2 HW request is AES_OUT

GPDMA2_REQUEST_AES_IN

GPDMA2 HW request is AES_IN

GPDMA2_REQUEST_HASH_IN

GPDMA2 HW request is HASH_IN

GPDMA2_REQUEST_UCPD1_RX

GPDMA2 HW request is UCPD1_RX

GPDMA2_REQUEST_UCPD1_TX

GPDMA2 HW request is UCPD1_TX

GPDMA2_REQUEST_CORDIC_READ

GPDMA2 HW request is CORDIC_READ

GPDMA2_REQUEST_CORDIC_WRITE

GPDMA2 HW request is CORDIC_WRITE

GPDMA2_REQUEST_FMAC_READ

GPDMA2 HW request is FMAC_READ

GPDMA2_REQUEST_FMAC_WRITE

GPDMA2 HW request is FMAC_WRITE

GPDMA2_REQUEST_SAES_OUT

GPDMA2 HW request is SAES_OUT

GPDMA2_REQUEST_SAES_IN

GPDMA2 HW request is SAES_IN

GPDMA2_REQUEST_I3C1_RX

GPDMA2 HW request is I3C1_RX

GPDMA2_REQUEST_I3C1_TX

GPDMA2 HW request is I3C1_TX

GPDMA2_REQUEST_I3C1_TC

GPDMA2 HW request is I3C1_TC

GPDMA2_REQUEST_I3C1_RS

GPDMA2 HW request is I3C1_RS

GPDMA2_REQUEST_I2C4_RX

GPDMA2 HW request is I2C4_RX

GPDMA2_REQUEST_I2C4_TX

GPDMA2 HW request is I2C4_TX

GPDMA2_REQUEST_LPTIM3_IC1

GPDMA2 HW request is LPTIM3_IC1

GPDMA2_REQUEST_LPTIM3_IC2

GPDMA2 HW request is LPTIM3_IC2

GPDMA2_REQUEST_LPTIM3_UE

GPDMA2 HW request is LPTIM3_UE

GPDMA2_REQUEST_LPTIM5_IC1

GPDMA2 HW request is LPTIM5_IC1

GPDMA2_REQUEST_LPTIM5_IC2

GPDMA2 HW request is LPTIM5_IC2

GPDMA2_REQUEST_LPTIM5_UE

GPDMA2 HW request is LPTIM5_UE

GPDMA2_REQUEST_LPTIM6_IC1

GPDMA2 HW request is LPTIM6_IC1

GPDMA2_REQUEST_LPTIM6_IC2

GPDMA2 HW request is LPTIM6_IC2

GPDMA2_REQUEST_LPTIM6_UE

GPDMA2 HW request is LPTIM6_UE

DMA_REQUEST_SW

DMA SW request

DMA Source Data Width

DMA_SRC_DATAWIDTH_BYTE

Source data width : Byte

DMA_SRC_DATAWIDTH_HALFWORD

Source data width : HalfWord

DMA_SRC_DATAWIDTH_WORD

Source data width : Word

DMA Source Increment Mode

DMA_SINC_FIXED

Source fixed single / burst

DMA_SINC_INCREMENTED

Source incremented single / burst

DMA Transfer Allocated Port

DMA_SRC_ALLOCATED_PORT0

Source allocated Port 0

DMA_SRC_ALLOCATED_PORT1

Source allocated Port 1

DMA_DEST_ALLOCATED_PORT0

Destination allocated Port 0

DMA_DEST_ALLOCATED_PORT1

Destination allocated Port 1

DMA Transfer Direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Transfer Event Mode

DMA_TCEM_BLOCK_TRANSFER

The TC event is generated at the end of each block and the HT event is generated at the half of each block

DMA_TCEM_REPEATED_BLOCK_TRANSFER

The TC event is generated at the end of the repeated block and the HT event is generated at the half of the repeated block

DMA_TCEM_EACH_LL_ITEM_TRANSFER

The TC event is generated at the end of each linked-list item and the HT event is generated at the half of each linked-list item

DMA_TCEM_LAST_LL_ITEM_TRANSFER

The TC event is generated at the end of the last linked-list item and the HT event is generated at the half of the last linked-list item

DMA Transfer Mode

DMA_NORMAL

Normal DMA transfer

DMA_PFCTRL

HW request peripheral flow control mode

22 HAL DMA Extension Driver

22.1 DMAEx Firmware driver registers structures

22.1.1 DMA_DataHandlingConfTypeDef

DMA_DataHandlingConfTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t DataExchange***
- ***uint32_t DataAlignment***

Field Documentation

- ***uint32_t DMA_DataHandlingConfTypeDef::DataExchange***
  Specifies the DMA channel data exchange mode. This parameter can be a value of ***DMAEx_Data_Exchange***
- ***uint32_t DMA_DataHandlingConfTypeDef::DataAlignment***
  Specifies the DMA channel data padding and alignment mode This parameter can be a value of ***DMAEx_Data_Alignment***

22.1.2 DMA_TriggerConfTypeDef

DMA_TriggerConfTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***

Field Documentation

- ***uint32_t DMA_TriggerConfTypeDef::TriggerMode***
  Specifies the DMA channel trigger mode. This parameter can be a value of ***DMAEx_Trigger_Mode***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerPolarity***
  Specifies the DMA channel trigger event polarity. This parameter can be a value of ***DMAEx_Trigger_Polarity***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerSelection***
  Specifies the DMA channel trigger event selection. This parameter can be a value of ***DMAEx_Trigger_Selection***

22.1.3 DMA_RepeatBlockConfTypeDef

DMA_RepeatBlockConfTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t RepeatCount***
- ***int32_t SrcAddrOffset***
- ***int32_t DestAddrOffset***
- ***int32_t BlkSrcAddrOffset***
- ***int32_t BlkDestAddrOffset***

Field Documentation

- ***uint32_t DMA_RepeatBlockConfTypeDef::RepeatCount***
  Specifies the DMA channel repeat count (the number of repetitions of block). This parameter can be a value between 1 and 2048
- ***int32_t DMA_RepeatBlockConfTypeDef::SrcAddrOffset***
  Specifies the DMA channel single/burst source address offset : This parameter can be a value between -8191 and 8191. If source address offset > 0 => Increment the source address by offset from where the last single/burst transfer ends. If source address offset < 0 => Decrement the source address by offset from where the last single/burst transfer ends. If source address offset == 0 => The next single/burst source address starts from where the last transfer ends

- ***int32_t DMA_RepeatBlockConfTypeDef::DestAddrOffset***
  Specifies the DMA channel single/burst destination address offset signed value : This parameter can be a value between -8191 and 8191. If destination address offset > 0 => Increment the destination address by offset from where the last single/burst transfer ends. If destination address offset < 0 => Decrement the destination address by offset from where the last single/burst transfer ends. If destination address offset == 0 => The next single/burst destination address starts from where the last transfer ends.
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkSrcAddrOffset***
  Specifies the DMA channel block source address offset signed value : This parameter can be a value between -65535 and 65535. If block source address offset > 0 => Increment the block source address by offset from where the last block ends. If block source address offset < 0 => Decrement the next block source address by offset from where the last block ends. If block source address offset == 0 => the next block source address starts from where the last block ends
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkDestAddrOffset***
  Specifies the DMA channel block destination address offset signed value : This parameter can be a value between -65535 and 65535. If block destination address offset > 0 => Increment the block destination address by offset from where the last block ends. If block destination address offset < 0 => Decrement the next block destination address by offset from where the last block ends. If block destination address offset == 0 => the next block destination address starts from where the last block ends

22.1.4 DMA_NodeConfTypeDef

DMA_NodeConfTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t NodeType***
- ***DMA_InitTypeDef Init***
- ***DMA_DataHandlingConfTypeDef DataHandlingConfig***
- ***DMA_TriggerConfTypeDef TriggerConfig***
- ***DMA_RepeatBlockConfTypeDef RepeatBlockConfig***
- ***uint32_t SrcAddress***
- ***uint32_t DstAddress***
- ***uint32_t DataSize***
- ***uint32_t SrcSecure***
- ***uint32_t DestSecure***

Field Documentation

- ***uint32_t DMA_NodeConfTypeDef::NodeType***
  Specifies the DMA channel node type. This parameter can be a value of ***DMAEx_Node_Type***
- ***DMA_InitTypeDef DMA_NodeConfTypeDef::Init***
  Specifies the DMA channel basic configuration
- ***DMA_DataHandlingConfTypeDef DMA_NodeConfTypeDef::DataHandlingConfig***
  Specifies the DMA channel data handling channel configuration
- ***DMA_TriggerConfTypeDef DMA_NodeConfTypeDef::TriggerConfig***
  Specifies the DMA channel trigger configuration
- ***DMA_RepeatBlockConfTypeDef DMA_NodeConfTypeDef::RepeatBlockConfig***
  Specifies the DMA channel repeated block configuration
- ***uint32_t DMA_NodeConfTypeDef::SrcAddress***
  Specifies the source memory address
- ***uint32_t DMA_NodeConfTypeDef::DstAddress***
  Specifies the destination memory address
- ***uint32_t DMA_NodeConfTypeDef::DataSize***
  Specifies the source data size in bytes
- ***uint32_t DMA_NodeConfTypeDef::SrcSecure***
  Specifies the source security attribute
- ***uint32_t DMA_NodeConfTypeDef::DestSecure***
  Specifies the destination security attribute

22.1.5 DMA_NodeTypeDef

DMA_NodeTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t LinkRegisters***
- ***uint32_t NodeInfo***

Field Documentation

- ***uint32_t DMA_NodeTypeDef::LinkRegisters[8U]***
  Physical Node register description
- ***uint32_t DMA_NodeTypeDef::NodeInfo***
  Node information

22.1.6 __DMA_QListTypeDef

__DMA_QListTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***DMA_NodeTypeDef * Head***
- ***DMA_NodeTypeDef * FirstCircularNode***
- ***uint32_t NodeNumber***
- ***__IO HAL_DMA_QStateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Type***

Field Documentation

- ***DMA_NodeTypeDef* __DMA_QListTypeDef::Head***
  Specifies the queue head node
- ***DMA_NodeTypeDef* __DMA_QListTypeDef::FirstCircularNode***
  Specifies the queue first circular node
- ***uint32_t __DMA_QListTypeDef::NodeNumber***
  Specifies the queue node number
- ***__IO HAL_DMA_QStateTypeDef __DMA_QListTypeDef::State***
  Specifies the queue state
- ***__IO uint32_t __DMA_QListTypeDef::ErrorCode***
  Specifies the queue error code
- ***__IO uint32_t __DMA_QListTypeDef::Type***
  Specifies whether the queue is static or dynamic

22.1.7 DMA_NodeInQInfoTypeDef

DMA_NodeInQInfoTypeDef is defined in the stm32h5xx_hal_dma_ex.h

Data Fields

- ***uint32_t cllr_offset***
- ***uint32_t previousnode_addr***
- ***uint32_t currentnode_pos***
- ***uint32_t currentnode_addr***
- ***uint32_t nextnode_addr***

Field Documentation

- ***uint32_t DMA_NodeInQInfoTypeDef::cllr_offset***
- ***uint32_t DMA_NodeInQInfoTypeDef::previousnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_pos***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::nextnode_addr***

22.2 DMAEx Firmware driver API description

The following section lists the various functions of the DMAEx library.

22.2.1

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.2 How to use this driver

22.2.3

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.4

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.5 Linked-List Initialization and De-Initialization Functions

22.2.6

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.7

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.8 Linked-List IO Operation Functions

22.2.9

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.10

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.11 Linked-List Management Functions

22.2.12

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.13

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.14 Data handling, repeated block and trigger configuration functions

22.2.15

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.16

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.17 Suspend and resume operation functions

22.2.18

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.19

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.20 Fifo status function

22.2.21

This section provides function allowing to get DMA channel FIFO level.

- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  - This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:

- ***HAL_DMAEx_GetFifoLevel()***

22.2.22 Detailed description of functions

HAL_DMAEx_List_Init

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in linked-list mode according to the specified parameters in the DMA_InitLinkedListTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_DeInit

Function name

HAL_StatusTypeDef HAL_DMAEx_List_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in linked-list mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start (DMA_HandleTypeDef *const hdma)

Function description

Start the DMA channel transfer in linked-list mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start_IT (DMA_HandleTypeDef *const hdma)

Function description

Starts the DMA channel transfer in linked-list mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_BuildNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_BuildNode (DMA_NodeConfTypeDef const *const pNodeConfig, DMA_NodeTypeDef *const pNode)

Function description

Build a DMA channel node according to the specified parameters in the DMA_NodeConfTypeDef.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

Notes

- The DMA linked-list node parameter address should be 32bit aligned and should not exceed the 64 KByte addressable space.

HAL_DMAEx_List_GetNodeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_GetNodeConfig (DMA_NodeConfTypeDef *const pNodeConfig, DMA_NodeTypeDef const *const pNode)

Function description

Get a DMA channel node configuration.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pPrevNode, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node in any queue position of linked-list queue according to selecting previous node.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the head of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the tail of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNode)

Function description

Remove node from any linked-list queue position.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Head (DMA_QListTypeDef *const pQList)

Function description

Remove the head node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Tail (DMA_QListTypeDef *const pQList)

Function description

Remove the tail node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pOldNode, DMA_NodeTypeDef *const pNewNode)

Function description

Replace node in linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pOldNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list old node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the head node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the tail node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ResetQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ResetQ (DMA_QListTypeDef *const pQList)

Function description

Reset the linked-list queue and unlink queue nodes.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ (DMA_QListTypeDef *const pSrcQList, DMA_NodeTypeDef const *const pPrevNode, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue to a destination linked-list queue according to selecting previous node.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Head (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the head of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Tail (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the tail of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularModeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularModeConfig (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pFirstCircularNode)

Function description

Set circular mode configuration for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pFirstCircularNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list first circular node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularMode (DMA_QListTypeDef *const pQList)

Function description

Set circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ClearCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ClearCircularMode (DMA_QListTypeDef *const pQList)

Function description

Clear circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToDynamic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToDynamic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to dynamic (Optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToStatic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToStatic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to static (Not optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_LinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_LinkQ (DMA_HandleTypeDef *const hdma, DMA_QListTypeDef *const pQList)

Function description

Link linked-list queue to a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_UnLinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_UnLinkQ (DMA_HandleTypeDef *const hdma)

Function description

Unlink linked-list queue from a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigDataHandling

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigDataHandling (DMA_HandleTypeDef *const hdma, DMA_DataHandlingConfTypeDef const *const pConfigDataHandling)

Function description

Configure the DMA channel data handling according to the specified parameters in the DMA_DataHandlingConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigDataHandling:** : Pointer to a DMA_DataHandlingConfTypeDef structure that contains the data handling configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigTrigger

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigTrigger (DMA_HandleTypeDef *const hdma, DMA_TriggerConfTypeDef const *const pConfigTrigger)

Function description

Configure the DMA channel trigger according to the specified parameters in the DMA_TriggerConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigTrigger:** : Pointer to a DMA_TriggerConfTypeDef structure that contains the trigger configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigRepeatBlock

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigRepeatBlock (DMA_HandleTypeDef *const hdma, DMA_RepeatBlockConfTypeDef const *const pConfigRepeatBlock)

Function description

Configure the DMA channel repeated block according to the specified parameters in the DMA_RepeatBlockConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigRepeatBlock:** : Pointer to a DMA_RepeatBlockConfTypeDef structure that contains the repeated block configuration.

Return values

- **HAL:** status.

HAL_DMAEx_Suspend

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a check for wait until the DMA channel is effectively suspended is added. If a channel is suspended while a data transfer is ongoing, the current data will be transferred and the channel will be effectively suspended only after the transfer of this single/burst data is finished.

HAL_DMAEx_Suspend_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend_IT (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_Resume

Function name

HAL_StatusTypeDef HAL_DMAEx_Resume (DMA_HandleTypeDef *const hdma)

Function description

Resume any suspended DMA channel transfer.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_GetFifoLevel

Function name

uint32_t HAL_DMAEx_GetFifoLevel (DMA_HandleTypeDef const *const hdma)

Function description

Get and returns the DMA channel FIFO level.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **Returns:** the number of available beats in FIFO.

22.3 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 DMAEx

DMAEx

DMAEx Data Alignment

DMA_DATA_RIGHTALIGN_ZEROPADDED

If source data width < destination data width => Right aligned padded with 0 up to destination data width

DMA_DATA_RIGHTALIGN_LEFTTRUNC

If source data width > destination data width => Right aligned left Truncated down to destination data width

DMA_DATA_RIGHTALIGN_SIGNEXT

If source data width < destination data width => Right Aligned padded with sign extended up to destination data width

DMA_DATA_LEFTALIGN_RIGHTTRUNC

If source data width > destination data width => Left Aligned Right Truncated down to the destination data width

DMA_DATA_PACK

If source data width < destination data width => Packed at the destination data width (Available only for GPDMA)

DMA_DATA_UNPACK

If source data width > destination data width => Unpacked at the destination data width (Available only for GPDMA)

DMAEx Data Exchange

DMA_EXCHANGE_NONE

No data exchange

DMA_EXCHANGE_DEST_BYTE

Destination Byte exchange when destination data width is > Byte

DMA_EXCHANGE_DEST_HALFWORD

Destination Half-Word exchange when destination data width is > Half-Word

DMA_EXCHANGE_SRC_BYTE

Source Byte endianness exchange when source data width is word

DMAEx LinkedList Mode

DMA_LINKEDLIST_NORMAL

Linear linked-list DMA channel transfer

DMA_LINKEDLIST_CIRCULAR

Circular linked-list DMA channel transfer

DMAEx Linked-List Allocated Port

DMA_LINK_ALLOCATED_PORT0

Link allocated port 0

DMA_LINK_ALLOCATED_PORT1

Link allocated port 1

DMAEx Link Step Mode

DMA_LSM_FULL_EXECUTION

Channel is executed for the full linked-list

DMA_LSM_1LINK_EXECUTION

Channel is executed once for the current LLI

DMAEx Node Type

DMA_GPDMA_LINEAR_NODE

Defines the GPDMA linear addressing node type

DMA_GPDMA_2D_NODE

Defines the GPDMA 2 dimension addressing node type

DMAEx Trigger Mode

DMA_TRIGM_BLOCK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_REPEATED_BLOCK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_SINGLE_BURST_TRANSFER

A single/burst transfer is conditioned by (at least) one hit trigger

DMAEx Trigger Polarity

DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

DMAEx Trigger Selection

GPDMA1_TRIGGER_EXTI_LINE0

GPDMA1 HW Trigger signal is EXTI_LINE0

GPDMA1_TRIGGER_EXTI_LINE1

GPDMA1 HW Trigger signal is EXTI_LINE1

GPDMA1_TRIGGER_EXTI_LINE2

GPDMA1 HW Trigger signal is EXTI_LINE2

GPDMA1_TRIGGER_EXTI_LINE3

GPDMA1 HW Trigger signal is EXTI_LINE3

GPDMA1_TRIGGER_EXTI_LINE4

GPDMA1 HW Trigger signal is EXTI_LINE4

GPDMA1_TRIGGER_EXTI_LINE5

GPDMA1 HW Trigger signal is EXTI_LINE5

GPDMA1_TRIGGER_EXTI_LINE6

GPDMA1 HW Trigger signal is EXTI_LINE6

GPDMA1_TRIGGER_EXTI_LINE7

GPDMA1 HW Trigger signal is EXTI_LINE7

GPDMA1_TRIGGER_TAMP_TRG1

GPDMA1 HW Trigger signal is TAMP_TRG1

GPDMA1_TRIGGER_TAMP_TRG2

GPDMA1 HW Trigger signal is TAMP_TRG2

GPDMA1_TRIGGER_TAMP_TRG3

GPDMA1 HW Trigger signal is TAMP_TRG3

GPDMA1_TRIGGER_LPTIM1_CH1

GPDMA1 HW Trigger signal is LPTIM1_CH1

GPDMA1_TRIGGER_LPTIM1_CH2

GPDMA1 HW Trigger signal is LPTIM1_CH2

GPDMA1_TRIGGER_LPTIM2_CH1

GPDMA1 HW Trigger signal is LPTIM2_CH1

GPDMA1_TRIGGER_LPTIM2_CH2

GPDMA1 HW Trigger signal is LPTIM2_CH2

GPDMA1_TRIGGER_RTC_ALRA_TRG

GPDMA1 HW Trigger signal is RTC_ALRA_TRG

GPDMA1_TRIGGER_RTC_ALRB_TRG

GPDMA1 HW Trigger signal is RTC_ALRB_TRG

GPDMA1_TRIGGER_RTC_WUT_TRG

GPDMA1 HW Trigger signal is RTC_WUT_TRG

GPDMA1_TRIGGER_GPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

GPDMA1_TRIGGER_GPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

GPDMA1_TRIGGER_GPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

GPDMA1_TRIGGER_GPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

GPDMA1_TRIGGER_GPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

GPDMA1_TRIGGER_GPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

GPDMA1_TRIGGER_GPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

GPDMA1_TRIGGER_GPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

GPDMA1_TRIGGER_GPDMA2_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH0_TCF

GPDMA1_TRIGGER_GPDMA2_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH1_TCF

GPDMA1_TRIGGER_GPDMA2_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH2_TCF

GPDMA1_TRIGGER_GPDMA2_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH3_TCF

GPDMA1_TRIGGER_GPDMA2_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH4_TCF

GPDMA1_TRIGGER_GPDMA2_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH5_TCF

GPDMA1_TRIGGER_GPDMA2_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH6_TCF

GPDMA1_TRIGGER_GPDMA2_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH7_TCF

GPDMA1_TRIGGER_TIM2_TRGO

GPDMA1 HW Trigger signal is TIM2_TRGO

GPDMA1_TRIGGER_TIM15_TRGO

GPDMA1 HW Trigger signal is TIM15_TRGO

GPDMA1_TRIGGER_TIM12_TRGO

GPDMA1 HW Trigger signal is TIM12_TRGO

GPDMA1_TRIGGER_LPTIM3_CH1

GPDMA1 HW Trigger signal is LPTIM3_CH1

GPDMA1_TRIGGER_LPTIM3_CH2

GPDMA1 HW Trigger signal is LPTIM3_CH2

GPDMA1_TRIGGER_LPTIM4_AIT

GPDMA1 HW Trigger signal is LPTIM4_AIT

GPDMA1_TRIGGER_LPTIM5_CH1

GPDMA1 HW Trigger signal is LPTIM5_CH1

GPDMA1_TRIGGER_LPTIM5_CH2

GPDMA1 HW Trigger signal is LPTIM5_CH2

GPDMA1_TRIGGER_LPTIM6_CH1

GPDMA1 HW Trigger signal is LPTIM6_CH1

GPDMA1_TRIGGER_LPTIM6_CH2

GPDMA1 HW Trigger signal is LPTIM6_CH2

GPDMA2_TRIGGER_EXTI_LINE0

GPDMA2 HW Trigger signal is EXTI_LINE0

GPDMA2_TRIGGER_EXTI_LINE1

GPDMA2 HW Trigger signal is EXTI_LINE1

GPDMA2_TRIGGER_EXTI_LINE2

GPDMA2 HW Trigger signal is EXTI_LINE2

GPDMA2_TRIGGER_EXTI_LINE3

GPDMA2 HW Trigger signal is EXTI_LINE3

GPDMA2_TRIGGER_EXTI_LINE4

GPDMA2 HW Trigger signal is EXTI_LINE4

GPDMA2_TRIGGER_EXTI_LINE5

GPDMA2 HW Trigger signal is EXTI_LINE5

GPDMA2_TRIGGER_EXTI_LINE6

GPDMA2 HW Trigger signal is EXTI_LINE6

GPDMA2_TRIGGER_EXTI_LINE7

GPDMA2 HW Trigger signal is EXTI_LINE7

GPDMA2_TRIGGER_TAMP_TRG1

GPDMA2 HW Trigger signal is TAMP_TRG1

GPDMA2_TRIGGER_TAMP_TRG2

GPDMA2 HW Trigger signal is TAMP_TRG2

GPDMA2_TRIGGER_TAMP_TRG3

GPDMA2 HW Trigger signal is TAMP_TRG3

GPDMA2_TRIGGER_LPTIM1_CH1

GPDMA2 HW Trigger signal is LPTIM1_CH1

GPDMA2_TRIGGER_LPTIM1_CH2

GPDMA2 HW Trigger signal is LPTIM1_CH2

GPDMA2_TRIGGER_LPTIM2_CH1

GPDMA2 HW Trigger signal is LPTIM2_CH1

GPDMA2_TRIGGER_LPTIM2_CH2

GPDMA2 HW Trigger signal is LPTIM2_CH2

GPDMA2_TRIGGER_RTC_ALRA_TRG

GPDMA2 HW Trigger signal is RTC_ALRA_TRG

GPDMA2_TRIGGER_RTC_ALRB_TRG

GPDMA2 HW Trigger signal is RTC_ALRB_TRG

GPDMA2_TRIGGER_RTC_WUT_TRG

GPDMA2 HW Trigger signal is RTC_WUT_TRG

GPDMA2_TRIGGER_GPDMA1_CH0_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH0_TCF

GPDMA2_TRIGGER_GPDMA1_CH1_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH1_TCF

GPDMA2_TRIGGER_GPDMA1_CH2_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH2_TCF

GPDMA2_TRIGGER_GPDMA1_CH3_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH3_TCF

GPDMA2_TRIGGER_GPDMA1_CH4_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH4_TCF

GPDMA2_TRIGGER_GPDMA1_CH5_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH5_TCF

GPDMA2_TRIGGER_GPDMA1_CH6_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH6_TCF

GPDMA2_TRIGGER_GPDMA1_CH7_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH7_TCF

GPDMA2_TRIGGER_GPDMA2_CH0_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH0_TCF

GPDMA2_TRIGGER_GPDMA2_CH1_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH1_TCF

GPDMA2_TRIGGER_GPDMA2_CH2_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH2_TCF

GPDMA2_TRIGGER_GPDMA2_CH3_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH3_TCF

GPDMA2_TRIGGER_GPDMA2_CH4_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH4_TCF

GPDMA2_TRIGGER_GPDMA2_CH5_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH5_TCF

GPDMA2_TRIGGER_GPDMA2_CH6_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH6_TCF

GPDMA2_TRIGGER_GPDMA2_CH7_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH7_TCF

GPDMA2_TRIGGER_TIM2_TRGO

GPDMA2 HW Trigger signal is TIM2_TRGO

GPDMA2_TRIGGER_TIM15_TRGO

GPDMA2 HW Trigger signal is TIM15_TRGO

GPDMA2_TRIGGER_TIM12_TRGO

GPDMA2 HW Trigger signal is TIM12_TRGO

GPDMA2_TRIGGER_LPTIM3_CH1

GPDMA2 HW Trigger signal is LPTIM3_CH1

GPDMA2_TRIGGER_LPTIM3_CH2

GPDMA2 HW Trigger signal is LPTIM3_CH2

GPDMA2_TRIGGER_LPTIM4_AIT

GPDMA2 HW Trigger signal is LPTIM4_AIT

GPDMA2_TRIGGER_LPTIM5_CH1

GPDMA2 HW Trigger signal is LPTIM5_CH1

GPDMA2_TRIGGER_LPTIM5_CH2

GPDMA2 HW Trigger signal is LPTIM5_CH2

GPDMA2_TRIGGER_LPTIM6_CH1

GPDMA2 HW Trigger signal is LPTIM6_CH1

GPDMA2_TRIGGER_LPTIM6_CH2

GPDMA2 HW Trigger signal is LPTIM6_CH2

23 HAL DTS Generic Driver

23.1 DTS Firmware driver registers structures

23.1.1 DTS_InitTypeDef

DTS_InitTypeDef is defined in the stm32h5xx_hal_dts.h

Data Fields

- ***uint32_t QuickMeasure***
- ***uint32_t RefClock***
- ***uint32_t TriggerInput***
- ***uint32_t SamplingTime***
- ***uint32_t Divider***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***

Field Documentation

- ***uint32_t DTS_InitTypeDef::QuickMeasure***
  Specifies the quick measure option selection of the DTS sensor. This parameter can be a value of ***DTS_Quick_Measurement***
- ***uint32_t DTS_InitTypeDef::RefClock***
  Specifies the reference clock selection of the DTS sensor. This parameter can be a value of ***DTS_Reference_Clock_Selection***
- ***uint32_t DTS_InitTypeDef::TriggerInput***
  Specifies the trigger input of the DTS sensor. This parameter can be a value of ***DTS_TriggerConfig***
- ***uint32_t DTS_InitTypeDef::SamplingTime***
  Specifies the sampling time configuration. This parameter can be a value of ***DTS_Sampling_Time***
- ***uint32_t DTS_InitTypeDef::Divider***
  Specifies the high speed clock divider ratio. This parameter can be a value from 0 to 127
- ***uint32_t DTS_InitTypeDef::HighThreshold***
  Specifies the high threshold of the DTS sensor
- ***uint32_t DTS_InitTypeDef::LowThreshold***
  Specifies the low threshold of the DTS sensor

23.1.2 __DTS_HandleTypeDef

__DTS_HandleTypeDef is defined in the stm32h5xx_hal_dts.h

Data Fields

- ***DTS_TypeDef * Instance***
- ***DTS_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DTS_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* EndCallback***
- ***void(* LowCallback***
- ***void(* HighCallback***
- ***void(* AsyncEndCallback***
- ***void(* AsyncLowCallback***
- ***void(* AsyncHighCallback***

Field Documentation

- ***DTS_TypeDef* __DTS_HandleTypeDef::Instance***
  Register base address

- ***DTS_InitTypeDef __DTS_HandleTypeDef::Init***
  DTS required parameters
- ***HAL_LockTypeDef __DTS_HandleTypeDef::Lock***
  DTS Locking object
- ***__IO HAL_DTS_StateTypeDef __DTS_HandleTypeDef::State***
  DTS peripheral state
- ***void(* __DTS_HandleTypeDef::MspInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS Base Msp Init Callback
- ***void(* __DTS_HandleTypeDef::MspDeInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS Base Msp DeInit Callback
- ***void(* __DTS_HandleTypeDef::EndCallback)(struct __DTS_HandleTypeDef *hdts)***
  End measure Callback
- ***void(* __DTS_HandleTypeDef::LowCallback)(struct __DTS_HandleTypeDef *hdts)***
  low threshold Callback
- ***void(* __DTS_HandleTypeDef::HighCallback)(struct __DTS_HandleTypeDef *hdts)***
  high threshold Callback
- ***void(* __DTS_HandleTypeDef::AsyncEndCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous end of measure Callback
- ***void(* __DTS_HandleTypeDef::AsyncLowCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous low threshold Callback
- ***void(* __DTS_HandleTypeDef::AsyncHighCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous high threshold Callback

23.2 DTS Firmware driver API description

The following section lists the various functions of the DTS library.

23.2.1 DTS Peripheral features

The STM32h5xx device family integrate one DTS sensor interface that converts the temperature into a square wave which frequency is proportional to the temperature.

- Temperature window comparator feature: The DTS allows defining the high and low threshold that will be used for temperature comparison. If the temperature data is equal or higher than the high threshold, or equal or lower than the low threshold, an interrupt is generated and the corresponding flag.
- Synchronous interrupts: A global interrupt output line is available on the DTS block. The interrupt can be generated at the end of measurement and/or when the measurement result is equal/higher or equal/lower than a predefined threshold.
- Asynchronous wakeup: The DTS block provides an asynchronous interrupt line. It is used only when the LSE is selected as reference clock. This line can generate a signal that wakes up the system from Sleep and stop mode at the end of measurement and/or when the measurement result is equal/higher or equal/lower than a predefined threshold.
- Trigger inputs: The temperature measurement can be triggered either by software or by an external event as well as lowpower timer.
- Measurement modes:
  - Quick measure : measure without a calibration when a high precision is not required.
  - Slow measure : measure with a calibration used when a high precision is required.
- Sampling time: The sampling time can be used to increase temperature measurement precision.
- Prescaler: When a calibration is ongoing, the counter clock must be slower than 1 MHz. This is achieved by the PCLK clock prescaler embedded in the temperature sensor.
- Operating Modes:
  - PCLK only : The temperature sensor registers can be accessed. The interface can consequently be reconfigured and the measurement sequence is performed using PCLK clock.
  - PCLK and LSE : The temperature sensor registers can be accessed. The interface can consequently be reconfigured and the measurement sequence is performed using the LSE clock.
  - LSE only : The registers cannot be accessed. The measurement can be performed using the LSE clock This mode is used to exit from Sleep and stop mode by using hardware triggers and the asynchronous interrupt line.

- Calibration: The temperature sensor must run the calibration prior to any frequency measurement. The calibration is performed automatically when the temperature measurement is triggered except for quick measurement mode.
- Low power modes:
  - Sleep mode: only works in LSE mode.
  - Stop mode: only works in LSE mode.

  ==========

23.2.2 How to use this driver

This driver provides functions to configure and program the DTS instances of STM32H5xx devices. To use the DTS, perform the following steps:

- Initialize the DTS low level resources by implementing the HAL_DTS_MspInit():
  - If required enable the DTS interrupt by configuring and enabling EXTI line in Interrupt mode After that enable the DTS interrupt vector using HAL_NVIC_EnableIRQ() function.
- Configure the DTS using HAL_DTS_Init() function:
  - Select the quick measure option
  - Select the reference clock.
  - Select the trigger input.
  - Select the sampling time.
  - Select the high speed clock ratio divider.
  - Select the low threshold
  - Select the high threshold
- Use HAL_DTS_Start() to enable and start the DTS sensor in polling mode.
- Use HAL_DTS_Start_IT() to enable and start the DTS sensor in interrupt mode.
- Use HAL_DTS_GetTemperature() to get temperature from DTS.
- Use HAL_DTS_Stop() to disable and stop the DTS sensor in polling mode.
- Use HAL_DTS_Stop_IT() to disable and stop the DTS sensor in interrupt mode.
- De-initialize the DTS using HAL_DTS_DeInit() function.

Callback and interrupts

When the compilation flag USE_HAL_DTS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

- Implement the weak function HAL_DTS_EndCallback() to get a callback at the end of the temperature measurement.
- Implement the weak function HAL_DTS_HighCallback() to get a callback when the temperature exceed the high threshold.
- Implement the weak function HAL_DTS_LowCallback() to get a callback when the temperature go down the low threshold.
- Implement the weak function HAL_DTS_AsyncEndCallback() to get a callback at the end of an asynchronous temperature measurement.
- Implement the weak function HAL_DTS_AsyncHighCallback() to get a callback when the temperature exceed the high threshold in an asynchronous measurement.
- Implement the weak function HAL_DTS_AsyncLowCallback() to get a callback when the temperature go down the low threshold in an asynchronous measurement.

The compilation flag USE_HAL_DTS_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks.

State

The driver permits to get in run time the status of the peripheral.

- Use HAL_DTS_GetState() to get the state of the DTS.

23.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize DTS

This section contains the following APIs:

- *HAL_DTS_Init()*
- *HAL_DTS_DeInit()*
- *HAL_DTS_MspInit()*
- *HAL_DTS_MspDeInit()*
- *HAL_DTS_RegisterCallback()*
- *HAL_DTS_UnRegisterCallback()*

23.2.4 DTS Start Stop operation functions

This section provides functions allowing to:

- Start a DTS Sensor without interrupt.
- Stop a DTS Sensor without interrupt.
- Start a DTS Sensor with interrupt generation.
- Stop a DTS Sensor with interrupt generation.

This section contains the following APIs:

- *HAL_DTS_Start()*
- *HAL_DTS_Stop()*
- *HAL_DTS_Start_IT()*
- *HAL_DTS_Stop_IT()*
- *HAL_DTS_GetTemperature()*
- *HAL_DTS_IRQHandler()*
- *HAL_DTS_EndCallback()*
- *HAL_DTS_LowCallback()*
- *HAL_DTS_HighCallback()*
- *HAL_DTS_AsyncEndCallback()*
- *HAL_DTS_AsyncLowCallback()*
- *HAL_DTS_AsyncHighCallback()*
- *HAL_DTS_GetState()*

23.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_DTS_GetState()*

23.2.6 Detailed description of functions

HAL_DTS_Init

Function name

HAL_StatusTypeDef HAL_DTS_Init (DTS_HandleTypeDef * hdts)

Function description

Initialize the DTS according to the specified parameters in the DTS_InitTypeDef and initialize the associated handle.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_DeInit

Function name

HAL_StatusTypeDef HAL_DTS_DeInit (DTS_HandleTypeDef * hdts)

Function description

DeInitialize the DTS peripheral.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the DTS configuration is locked. To unlock the configuration, perform a system reset.

HAL_DTS_MspInit

Function name

void HAL_DTS_MspInit (DTS_HandleTypeDef * hdts)

Function description

Initialize the DTS MSP.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_MspDeInit

Function name

void HAL_DTS_MspDeInit (DTS_HandleTypeDef * hdts)

Function description

DeInitialize the DTS MSP.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DTS_RegisterCallback (DTS_HandleTypeDef * hdts, HAL_DTS_CallbackIDTypeDef CallbackID, pDTS_CallbackTypeDef pCallback)

Function description

Register a user DTS callback to be used instead of the weak predefined callback.

Parameters

- **hdts:** DTS handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_DTS_MEAS_COMPLETE_CB_ID measure complete callback ID.
  - HAL_DTS_ASYNC_MEAS_COMPLETE_CB_ID asynchronous measure complete callback ID.
  - HAL_DTS_LOW_THRESHOLD_CB_ID low threshold detection callback ID.
  - HAL_DTS_ASYNC_LOW_THRESHOLD_CB_ID asynchronous low threshold detection callback ID.
  - HAL_DTS_HIGH_THRESHOLD_CB_ID high threshold detection callback ID.
  - HAL_DTS_ASYNC_HIGH_THRESHOLD_CB_ID asynchronous high threshold detection callback ID.
  - HAL_DTS_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DTS_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_DTS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DTS_UnRegisterCallback (DTS_HandleTypeDef * hdts, HAL_DTS_CallbackIDTypeDef CallbackID)

Function description

Unregister a user DTS callback.

Parameters

- **hdts:** DTS handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_DTS_MEAS_COMPLETE_CB_ID measure complete callback ID.
  - HAL_DTS_ASYNC_MEAS_COMPLETE_CB_ID asynchronous measure complete callback ID.
  - HAL_DTS_LOW_THRESHOLD_CB_ID low threshold detection callback ID.
  - HAL_DTS_ASYNC_LOW_THRESHOLD_CB_ID asynchronous low threshold detection callback ID.
  - HAL_DTS_HIGH_THRESHOLD_CB_ID high threshold detection callback ID.
  - HAL_DTS_ASYNC_HIGH_THRESHOLD_CB_ID asynchronous high threshold detection callback ID.
  - HAL_DTS_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DTS_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_DTS_Start

Function name

HAL_StatusTypeDef HAL_DTS_Start (DTS_HandleTypeDef * hdts)

Function description

Start the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_Stop

Function name

HAL_StatusTypeDef HAL_DTS_Stop (DTS_HandleTypeDef * hdts)

Function description

Stop the DTS Sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_GetTemperature

Function name

HAL_StatusTypeDef HAL_DTS_GetTemperature (DTS_HandleTypeDef * hdts, int32_t * Temperature)

Function description

Get temperature from DTS.

Parameters

- **hdts:** DTS handle
- **Temperature:** Temperature in deg C

Return values

- **HAL:** status

Notes

- This function retrieves latest available measure

HAL_DTS_Start_IT

Function name

HAL_StatusTypeDef HAL_DTS_Start_IT (DTS_HandleTypeDef * hdts)

Function description

Enable the interrupt(s) and start the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_Stop_IT

Function name

HAL_StatusTypeDef HAL_DTS_Stop_IT (DTS_HandleTypeDef * hdts)

Function description

Disable the interrupt(s) and stop the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_IRQHandler

Function name

void HAL_DTS_IRQHandler (DTS_HandleTypeDef * hdts)

Function description

DTS sensor IRQ Handler.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_GetState

Function name

HAL_DTS_StateTypeDef HAL_DTS_GetState (const DTS_HandleTypeDef * hdts)

Function description

Return the DTS handle state.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** state

HAL_DTS_EndCallback

Function name

void HAL_DTS_EndCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor End measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_LowCallback

Function name

void HAL_DTS_LowCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor low threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_HighCallback

Function name

void HAL_DTS_HighCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor high threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncEndCallback

Function name

void HAL_DTS_AsyncEndCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous end measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncLowCallback

Function name

void HAL_DTS_AsyncLowCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous low threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncHighCallback

Function name

void HAL_DTS_AsyncHighCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous high threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

23.3 DTS Firmware driver defines

The following section lists the various define and macros of the module.

23.3.1 DTS

DTS

DTS Exported Macros

__HAL_DTS_RESET_HANDLE_STATE

Description:

- Reset DTS handle state.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_ENABLE

Description:

- Enable the specified DTS sensor.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_DISABLE

Description:

- Disable the specified DTS sensor.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_ENABLE_IT

Description:

- Enable the DTS EXTI line in interrupt mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_DISABLE_IT

Description:

- Disable the DTS EXTI line in interrupt mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_ENABLE_EVENT

Description:

- Enable the DTS EXTI Line in event mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_DISABLE_EVENT

Description:

- Disable the DTS EXTI Line in event mode.

Return value:

- None

__HAL_DTS_GET_FLAG

Description:

- Checks whether the specified DTS flag is set or not.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - DTS_FLAG_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_ITL : interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_ITH : interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_AITE: asynchronous interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_AITL: asynchronous interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_AITH: asynchronous interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_RDY : Ready flag for DTS1

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_DTS_CLEAR_FLAG

Description:

- Clears the specified DTS pending flag.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - DTS_FLAG_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_ITL : interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_ITH : interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_AITE: asynchronous interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_AITL: asynchronous interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_AITH: asynchronous interrupt flag for high threshold for DTS1

Return value:

- None

__HAL_DTS_ENABLE_IT

Description:

- Enable the specified DTS interrupt.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __INTERRUPT__: specifies the DTS interrupt source to enable. This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- None

__HAL_DTS_DISABLE_IT

Description:

- Disable the specified DTS interrupt.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __INTERRUPT__: specifies the DTS interrupt source to enable. This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- None

__HAL_DTS_GET_IT_SOURCE

Description:

- Check whether the specified DTS interrupt source is enabled or not.

Parameters:

- __HANDLE__: DTS handle.
- __INTERRUPT__: DTS interrupt source to check This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- State: of interruption (SET or RESET)

__HAL_DTS_GET_REFCLK

Description:

- Check whether the specified DTS REFCLK is selected.

Parameters:

- __HANDLE__: DTS handle.
- __REFCLK__: DTS reference clock to check This parameter can be one of the following values:
  - DTS_REFCLKSEL_LSE: Low speed REF clock
  - DTS_REFCLKSEL_PCLK: High speed REF clock

Return value:

- State: of the REF clock tested (SET or RESET)

__HAL_DTS_GET_TRIGGER

Description:

- Get Trigger.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- One: of the following trigger DTS_TRIGGER_HW_NONE : No HW trigger (SW trigger)
  DTS_TRIGGER_LPTIMER1: LPTIMER1 trigger DTS_TRIGGER_LPTIMER2: LPTIMER2 trigger
  DTS_TRIGGER_LPTIMER3: LPTIMER3 trigger DTS_TRIGGER_EXTI13 : EXTI13 trigger

DTS EXTI Lines

DTS_EXTI_LINE_DTS1

EXTI line 50 connected to DTS1 output

DTS Flag Definitions

DTS_FLAG_TS1_ITE

Interrupt flag for end of measure for DTS1

DTS_FLAG_TS1_ITL

Interrupt flag for low threshold for DTS1

DTS_FLAG_TS1_ITH

Interrupt flag for high threshold for DTS1

DTS_FLAG_TS1_AITE

Asynchronous Interrupt flag for end of measure for DTS1

DTS_FLAG_TS1_AITL

Asynchronous Interrupt flag for low threshold for DTS1

DTS_FLAG_TS1_AITH

Asynchronous Interrupt flag for high threshold for DTS1

DTS_FLAG_TS1_RDY

Ready flag for DTS1

DTS Interrupts Definitions

DTS_IT_TS1_ITE

Enable interrupt flag for end of measure for DTS1

DTS_IT_TS1_ITL

Enable interrupt flag for low threshold for DTS1

DTS_IT_TS1_ITH

Enable interrupt flag for high threshold for DTS1

DTS_IT_TS1_AITE

Enable asynchronous interrupt flag for end of measure for DTS1

DTS_IT_TS1_AITL

Enable asynchronous interrupt flag for low threshold for DTS1

DTS_IT_TS1_AITH

Enable asynchronous interrupt flag for high threshold for DTS1

DTS Private macros to check input parameters

IS_DTS_QUICKMEAS

IS_DTS_REFCLK

IS_DTS_TRIGGERINPUT

IS_DTS_THRESHOLD

IS_DTS_DIVIDER_RATIO_NUMBER

IS_DTS_SAMPLINGTIME

DTS Quick Measurement

DTS_QUICKMEAS_ENABLE

Enable the Quick Measure (Measure without calibration)

DTS_QUICKMEAS_DISABLE

Disable the Quick Measure (Measure with calibration)

DTS Reference Clock Selection

DTS_REFCLKSEL_LSE

Low speed REF clock (LSE)

DTS_REFCLKSEL_PCLK

High speed REF clock (PCLK)

DTS Sampling Time

DTS_SMP_TIME_1_CYCLE

1 clock cycle for the sampling time

DTS_SMP_TIME_2_CYCLE

2 clock cycle for the sampling time

DTS_SMP_TIME_3_CYCLE

3 clock cycle for the sampling time

DTS_SMP_TIME_4_CYCLE

4 clock cycle for the sampling time

DTS_SMP_TIME_5_CYCLE

5 clock cycle for the sampling time

DTS_SMP_TIME_6_CYCLE

6 clock cycle for the sampling time

DTS_SMP_TIME_7_CYCLE

7 clock cycle for the sampling time

DTS_SMP_TIME_8_CYCLE

8 clock cycle for the sampling time

DTS_SMP_TIME_9_CYCLE

9 clock cycle for the sampling time

DTS_SMP_TIME_10_CYCLE

10 clock cycle for the sampling time

DTS_SMP_TIME_11_CYCLE

11 clock cycle for the sampling time

DTS_SMP_TIME_12_CYCLE

12 clock cycle for the sampling time

DTS_SMP_TIME_13_CYCLE

13 clock cycle for the sampling time

DTS_SMP_TIME_14_CYCLE

14 clock cycle for the sampling time

DTS_SMP_TIME_15_CYCLE

15 clock cycle for the sampling time

DTS Trigger Configuration

DTS_TRIGGER_HW_NONE

DTS_TRIGGER_LPTIMER1

DTS_TRIGGER_LPTIMER2

DTS_TRIGGER_LPTIMER3

DTS_TRIGGER_EXTI13

24 HAL ETH Generic Driver

24.1 ETH Firmware driver registers structures

24.1.1 ETH_DMADescTypeDef

ETH_DMADescTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***__IO uint32_t DESC0***
- ***__IO uint32_t DESC1***
- ***__IO uint32_t DESC2***
- ***__IO uint32_t DESC3***
- ***uint32_t BackupAddr0***
- ***uint32_t BackupAddr1***

Field Documentation

- ***__IO uint32_t ETH_DMADescTypeDef::DESC0***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC1***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC2***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC3***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr0***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr1***

24.1.2 __ETH_BufferTypeDef

__ETH_BufferTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint8_t * buffer***
- ***uint32_t len***
- ***struct __ETH_BufferTypeDef * next***

Field Documentation

- ***uint8_t* __ETH_BufferTypeDef::buffer***
- ***uint32_t __ETH_BufferTypeDef::len***
- ***struct __ETH_BufferTypeDef* __ETH_BufferTypeDef::next***

24.1.3 ETH_TxDescListTypeDef

ETH_TxDescListTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t TxDesc***
- ***uint32_t CurTxDesc***
- ***uint32_t * PacketAddress***
- ***uint32_t * CurrentPacketAddress***
- ***uint32_t BuffersInUse***
- ***uint32_t releaseIndex***

Field Documentation

- ***uint32_t ETH_TxDescListTypeDef::TxDesc[ETH_TX_DESC_CNT]***
- ***uint32_t ETH_TxDescListTypeDef::CurTxDesc***
- ***uint32_t* ETH_TxDescListTypeDef::PacketAddress[ETH_TX_DESC_CNT]***
- ***uint32_t* ETH_TxDescListTypeDef::CurrentPacketAddress***
- ***uint32_t ETH_TxDescListTypeDef::BuffersInUse***
- ***uint32_t ETH_TxDescListTypeDef::releaseIndex***

24.1.4 ETH_TxPacketConfig

ETH_TxPacketConfig is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t Attributes***
- ***uint32_t Length***
- ***ETH_BufferTypeDef * TxBuffer***
- ***uint32_t SrcAddrCtrl***
- ***uint32_t CRCPadCtrl***
- ***uint32_t ChecksumCtrl***
- ***uint32_t MaxSegmentSize***
- ***uint32_t PayloadLen***
- ***uint32_t TCPHeaderLen***
- ***uint32_t VlanTag***
- ***uint32_t VlanCtrl***
- ***uint32_t InnerVlanTag***
- ***uint32_t InnerVlanCtrl***
- ***void * pData***

Field Documentation

- ***uint32_t ETH_TxPacketConfig::Attributes***
  Tx packet HW features capabilities. This parameter can be a combination of ***ETH_Tx_Packet_Attributes***
- ***uint32_t ETH_TxPacketConfig::Length***
  Total packet length
- ***ETH_BufferTypeDef* ETH_TxPacketConfig::TxBuffer***
  Tx buffers pointers
- ***uint32_t ETH_TxPacketConfig::SrcAddrCtrl***
  Specifies the source address insertion control. This parameter can be a value of ***ETH_Tx_Packet_Source_Addr_Control***
- ***uint32_t ETH_TxPacketConfig::CRCPadCtrl***
  Specifies the CRC and Pad insertion and replacement control. This parameter can be a value of ***ETH_Tx_Packet_CRC_Pad_Control***
- ***uint32_t ETH_TxPacketConfig::ChecksumCtrl***
  Specifies the checksum insertion control. This parameter can be a value of ***ETH_Tx_Packet_Checksum_Control***
- ***uint32_t ETH_TxPacketConfig::MaxSegmentSize***
  Sets TCP maximum segment size only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFF
- ***uint32_t ETH_TxPacketConfig::PayloadLen***
  Sets Total payload length only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFFF
- ***uint32_t ETH_TxPacketConfig::TCPHeaderLen***
  Sets TCP header length only when TCP segmentation is enabled. This parameter can be a value from 0x5 to 0xF
- ***uint32_t ETH_TxPacketConfig::VlanTag***
  Sets VLAN Tag only when VLAN is enabled. This parameter can be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_TxPacketConfig::VlanCtrl***
  Specifies VLAN Tag insertion control only when VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_VLAN_Control***
- ***uint32_t ETH_TxPacketConfig::InnerVlanTag***
  Sets Inner VLAN Tag only when Inner VLAN is enabled. This parameter can be a value from 0x0 to 0x3FFFF
- ***uint32_t ETH_TxPacketConfig::InnerVlanCtrl***
  Specifies Inner VLAN Tag insertion control only when Inner VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_Inner_VLAN_Control***

- ***void* ETH_TxPacketConfig::pData***
  Specifies Application packet pointer to save

24.1.5 ETH_TimeStampTypeDef

ETH_TimeStampTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t TimeStampLow***
- ***uint32_t TimeStampHigh***

Field Documentation

- ***uint32_t ETH_TimeStampTypeDef::TimeStampLow***
- ***uint32_t ETH_TimeStampTypeDef::TimeStampHigh***

24.1.6 ETH_RxDescListTypeDef

ETH_RxDescListTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t RxDesc***
- ***uint32_t ItMode***
- ***uint32_t RxDescIdx***
- ***uint32_t RxDescCnt***
- ***uint32_t RxDataLength***
- ***uint32_t RxBuildDescIdx***
- ***uint32_t RxBuildDescCnt***
- ***uint32_t pRxLastRxDesc***
- ***ETH_TimeStampTypeDef TimeStamp***
- ***void * pRxStart***
- ***void * pRxEnd***

Field Documentation

- ***uint32_t ETH_RxDescListTypeDef::RxDesc[ETH_RX_DESC_CNT]***
- ***uint32_t ETH_RxDescListTypeDef::ItMode***
- ***uint32_t ETH_RxDescListTypeDef::RxDescIdx***
- ***uint32_t ETH_RxDescListTypeDef::RxDescCnt***
- ***uint32_t ETH_RxDescListTypeDef::RxDataLength***
- ***uint32_t ETH_RxDescListTypeDef::RxBuildDescIdx***
- ***uint32_t ETH_RxDescListTypeDef::RxBuildDescCnt***
- ***uint32_t ETH_RxDescListTypeDef::pRxLastRxDesc***
- ***ETH_TimeStampTypeDef ETH_RxDescListTypeDef::TimeStamp***
- ***void* ETH_RxDescListTypeDef::pRxStart***
- ***void* ETH_RxDescListTypeDef::pRxEnd***

24.1.7 ETH_MACConfigTypeDef

ETH_MACConfigTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t SourceAddrControl***
- ***FunctionalState ChecksumOffload***
- ***uint32_t InterPacketGapVal***
- ***FunctionalState GiantPacketSizeLimitControl***
- ***FunctionalState Support2KPacket***
- ***FunctionalState CRCStripTypePacket***
- ***FunctionalState AutomaticPadCRCStrip***
- ***FunctionalState Watchdog***

- ***FunctionalState Jabber***
- ***FunctionalState JumboPacket***
- ***uint32_t Speed***
- ***uint32_t DuplexMode***
- ***FunctionalState LoopbackMode***
- ***FunctionalState CarrierSenseBeforeTransmit***
- ***FunctionalState ReceiveOwn***
- ***FunctionalState CarrierSenseDuringTransmit***
- ***FunctionalState RetryTransmission***
- ***uint32_t BackOffLimit***
- ***FunctionalState DeferralCheck***
- ***uint32_t PreambleLength***
- ***FunctionalState UnicastSlowProtocolPacketDetect***
- ***FunctionalState SlowProtocolDetect***
- ***FunctionalState CRCCheckingRxPackets***
- ***uint32_t GiantPacketSizeLimit***
- ***FunctionalState ExtendedInterPacketGap***
- ***uint32_t ExtendedInterPacketGapVal***
- ***FunctionalState ProgrammableWatchdog***
- ***uint32_t WatchdogTimeout***
- ***uint32_t PauseTime***
- ***FunctionalState ZeroQuantaPause***
- ***uint32_t PauseLowThreshold***
- ***FunctionalState TransmitFlowControl***
- ***FunctionalState UnicastPausePacketDetect***
- ***FunctionalState ReceiveFlowControl***
- ***uint32_t TransmitQueueMode***
- ***uint32_t ReceiveQueueMode***
- ***FunctionalState DropTCPIPChecksumErrorPacket***
- ***FunctionalState ForwardRxErrorPacket***
- ***FunctionalState ForwardRxUndersizedGoodPacket***

Field Documentation

- ***uint32_t ETH_MACConfigTypeDef::SourceAddrControl***
  Selects the Source Address Insertion or Replacement Control. This parameter can be a value of ***ETH_Source_Addr_Control***
- ***FunctionalState ETH_MACConfigTypeDef::ChecksumOffload***
  Enables or Disable the checksum checking for received packet payloads TCP, UDP or ICMP headers
- ***uint32_t ETH_MACConfigTypeDef::InterPacketGapVal***
  Sets the minimum IPG between Packet during transmission. This parameter can be a value of ***ETH_Inter_Packet_Gap***
- ***FunctionalState ETH_MACConfigTypeDef::GiantPacketSizeLimitControl***
  Enables or disables the Giant Packet Size Limit Control.
- ***FunctionalState ETH_MACConfigTypeDef::Support2KPacket***
  Enables or disables the IEEE 802.3as Support for 2K length Packets
- ***FunctionalState ETH_MACConfigTypeDef::CRCStripTypePacket***
  Enables or disables the CRC stripping for Type packets.
- ***FunctionalState ETH_MACConfigTypeDef::AutomaticPadCRCStrip***
  Enables or disables the Automatic MAC Pad/CRC Stripping.
- ***FunctionalState ETH_MACConfigTypeDef::Watchdog***
  Enables or disables the Watchdog timer on Rx path.
- ***FunctionalState ETH_MACConfigTypeDef::Jabber***
  Enables or disables Jabber timer on Tx path.

- ***FunctionalState ETH_MACConfigTypeDef::JumboPacket***
  Enables or disables receiving Jumbo Packet When enabled, the MAC allows jumbo packets of 9,018 bytes without reporting a giant packet error
- ***uint32_t ETH_MACConfigTypeDef::Speed***
  Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of ***ETH_Speed***
- ***uint32_t ETH_MACConfigTypeDef::DuplexMode***
  Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of ***ETH_Duplex_Mode***
- ***FunctionalState ETH_MACConfigTypeDef::LoopbackMode***
  Enables or disables the loopback mode
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseBeforeTransmit***
  Enables or disables the Carrier Sense Before Transmission in Full Duplex Mode.
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveOwn***
  Enables or disables the Receive Own in Half Duplex mode.
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseDuringTransmit***
  Enables or disables the Carrier Sense During Transmission in the Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::RetryTransmission***
  Enables or disables the MAC retry transmission, when a collision occurs in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::BackOffLimit***
  Selects the BackOff limit value. This parameter can be a value of ***ETH_Back_Off_Limit***
- ***FunctionalState ETH_MACConfigTypeDef::DeferralCheck***
  Enables or disables the deferral check function in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::PreambleLength***
  Selects or not the Preamble Length for Transmit packets (Full Duplex mode). This parameter can be a value of ***ETH_Preamble_Length***
- ***FunctionalState ETH_MACConfigTypeDef::UnicastSlowProtocolPacketDetect***
  Enable or disables the Detection of Slow Protocol Packets with unicast address.
- ***FunctionalState ETH_MACConfigTypeDef::SlowProtocolDetect***
  Enable or disables the Slow Protocol Detection.
- ***FunctionalState ETH_MACConfigTypeDef::CRCCheckingRxPackets***
  Enable or disables the CRC Checking for Received Packets.
- ***uint32_t ETH_MACConfigTypeDef::GiantPacketSizeLimit***
  Specifies the packet size that the MAC will declare it as Giant, If it's size is greater than the value programmed in this field in units of bytes This parameter must be a number between Min_Data = 0x618 (1518 byte) and Max_Data = 0x3FFF (32 Kbyte).
- ***FunctionalState ETH_MACConfigTypeDef::ExtendedInterPacketGap***
  Enable or disables the extended inter packet gap.
- ***uint32_t ETH_MACConfigTypeDef::ExtendedInterPacketGapVal***
  Sets the Extended IPG between Packet during transmission. This parameter can be a value from 0x0 to 0xFF
- ***FunctionalState ETH_MACConfigTypeDef::ProgrammableWatchdog***
  Enable or disables the Programmable Watchdog.
- ***uint32_t ETH_MACConfigTypeDef::WatchdogTimeout***
  This field is used as watchdog timeout for a received packet This parameter can be a value of ***ETH_Watchdog_Timeout***
- ***uint32_t ETH_MACConfigTypeDef::PauseTime***
  This field holds the value to be used in the Pause Time field in the transmit control packet. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF.
- ***FunctionalState ETH_MACConfigTypeDef::ZeroQuantaPause***
  Enable or disables the automatic generation of Zero Quanta Pause Control packets.
- ***uint32_t ETH_MACConfigTypeDef::PauseLowThreshold***
  This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Packet. This parameter can be a value of ***ETH_Pause_Low_Threshold***
- ***FunctionalState ETH_MACConfigTypeDef::TransmitFlowControl***
  Enables or disables the MAC to transmit Pause packets in Full Duplex mode or the MAC back pressure operation in Half Duplex mode

- ***FunctionalState ETH_MACConfigTypeDef::UnicastPausePacketDetect***
  Enables or disables the MAC to detect Pause packets with unicast address of the station
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveFlowControl***
  Enables or disables the MAC to decodes the received Pause packet and disables its transmitter for a specified (Pause) time
- ***uint32_t ETH_MACConfigTypeDef::TransmitQueueMode***
  Specifies the Transmit Queue operating mode. This parameter can be a value of ***ETH_Transmit_Mode***
- ***uint32_t ETH_MACConfigTypeDef::ReceiveQueueMode***
  Specifies the Receive Queue operating mode. This parameter can be a value of ***ETH_Receive_Mode***
- ***FunctionalState ETH_MACConfigTypeDef::DropTCPIPChecksumErrorPacket***
  Enables or disables Dropping of TCPIP Checksum Error Packets.
- ***FunctionalState ETH_MACConfigTypeDef::ForwardRxErrorPacket***
  Enables or disables forwarding Error Packets.
- ***FunctionalState ETH_MACConfigTypeDef::ForwardRxUndersizedGoodPacket***
  Enables or disables forwarding Undersized Good Packets.

24.1.8 ETH_DMAConfigTypeDef

ETH_DMAConfigTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint32_t DMAArbitration***
- ***FunctionalState AddressAlignedBeats***
- ***uint32_t BurstMode***
- ***FunctionalState RebuildINCRxBurst***
- ***FunctionalState PBLx8Mode***
- ***uint32_t TxDMABurstLength***
- ***FunctionalState SecondPacketOperate***
- ***uint32_t RxDMABurstLength***
- ***FunctionalState FlushRxPacket***
- ***FunctionalState TCPSegmentation***
- ***uint32_t MaximumSegmentSize***

Field Documentation

- ***uint32_t ETH_DMAConfigTypeDef::DMAArbitration***
  Sets the arbitration scheme between DMA Tx and Rx This parameter can be a value of ***ETH_DMA_Arbitration***
- ***FunctionalState ETH_DMAConfigTypeDef::AddressAlignedBeats***
  Enables or disables the AHB Master interface address aligned burst transfers on Read and Write channels
- ***uint32_t ETH_DMAConfigTypeDef::BurstMode***
  Sets the AHB Master interface burst transfers. This parameter can be a value of ***ETH_Burst_Mode***
- ***FunctionalState ETH_DMAConfigTypeDef::RebuildINCRxBurst***
  Enables or disables the AHB Master to rebuild the pending beats of any initiated burst transfer with INCRx and SINGLE transfers.
- ***FunctionalState ETH_DMAConfigTypeDef::PBLx8Mode***
  Enables or disables the PBL multiplication by eight.
- ***uint32_t ETH_DMAConfigTypeDef::TxDMABurstLength***
  Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of ***ETH_Tx_DMA_Burst_Length***
- ***FunctionalState ETH_DMAConfigTypeDef::SecondPacketOperate***
  Enables or disables the Operate on second Packet mode, which allows the DMA to process a second Packet of Transmit data even before obtaining the status for the first one.
- ***uint32_t ETH_DMAConfigTypeDef::RxDMABurstLength***
  Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of ***ETH_Rx_DMA_Burst_Length***
- ***FunctionalState ETH_DMAConfigTypeDef::FlushRxPacket***
  Enables or disables the Rx Packet Flush

- ***FunctionalState ETH_DMAConfigTypeDef::TCPSegmentation***
  Enables or disables the TCP Segmentation
- ***uint32_t ETH_DMAConfigTypeDef::MaximumSegmentSize***
  Sets the maximum segment size that should be used while segmenting the packet This parameter can be a value from 0x40 to 0x3FFF

24.1.9 ETH_InitTypeDef

ETH_InitTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***uint8_t * MACAddr***
- ***ETH_MediaInterfaceTypeDef MediaInterface***
- ***ETH_DMADescTypeDef * TxDesc***
- ***ETH_DMADescTypeDef * RxDesc***
- ***uint32_t RxBuffLen***

Field Documentation

- ***uint8_t* ETH_InitTypeDef::MACAddr***
  MAC Address of used Hardware: must be pointer on an array of 6 bytes
- ***ETH_MediaInterfaceTypeDef ETH_InitTypeDef::MediaInterface***
  Selects the MII interface or the RMII interface.
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::TxDesc***
  Provides the address of the first DMA Tx descriptor in the list
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::RxDesc***
  Provides the address of the first DMA Rx descriptor in the list
- ***uint32_t ETH_InitTypeDef::RxBuffLen***
  Provides the length of Rx buffers size

24.1.10 __ETH_HandleTypeDef

__ETH_HandleTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***ETH_TypeDef * Instance***
- ***ETH_InitTypeDef Init***
- ***ETH_TxDescListTypeDef TxDescList***
- ***ETH_RxDescListTypeDef RxDescList***
- ***__IO HAL_ETH_StateTypeDef gState***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t DMAErrorCode***
- ***__IO uint32_t MACErrorCode***
- ***__IO uint32_t MACWakeUpEvent***
- ***__IO uint32_t MACLPIEvent***
- ***__IO uint32_t IsPtpConfigured***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* PMTCallback***
- ***void(* EEECallback***
- ***void(* WakeUpCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***pETH_rxAllocateCallbackTypeDef rxAllocateCallback***
- ***pETH_rxLinkCallbackTypeDef rxLinkCallback***
- ***pETH_txFreeCallbackTypeDef txFreeCallback***
- ***pETH_txPtpCallbackTypeDef txPtpCallback***

Field Documentation

- ***ETH_TypeDef* __ETH_HandleTypeDef::Instance***
  Register base address
- ***ETH_InitTypeDef __ETH_HandleTypeDef::Init***
  Ethernet Init Configuration
- ***ETH_TxDescListTypeDef __ETH_HandleTypeDef::TxDescList***
  Tx descriptor wrapper: holds all Tx descriptors list addresses and current descriptor index
- ***ETH_RxDescListTypeDef __ETH_HandleTypeDef::RxDescList***
  Rx descriptor wrapper: holds all Rx descriptors list addresses and current descriptor index
- ***__IO HAL_ETH_StateTypeDef __ETH_HandleTypeDef::gState***
  ETH state information related to global Handle management and also related to Tx operations. This parameter can be a value of ***ETH_State_Codes***
- ***__IO uint32_t __ETH_HandleTypeDef::ErrorCode***
  Holds the global Error code of the ETH HAL status machine This parameter can be a value of ***ETH_Error_Code***.
- ***__IO uint32_t __ETH_HandleTypeDef::DMAErrorCode***
  Holds the DMA Rx Tx Error code when a DMA AIS interrupt occurs This parameter can be a combination of ***ETH_DMA_Status_Flags***
- ***__IO uint32_t __ETH_HandleTypeDef::MACErrorCode***
  Holds the MAC Rx Tx Error code when a MAC Rx or Tx status interrupt occurs This parameter can be a combination of ***ETH_MAC_Rx_Tx_Status***
- ***__IO uint32_t __ETH_HandleTypeDef::MACWakeUpEvent***
  Holds the Wake Up event when the MAC exit the power down mode This parameter can be a value of ***ETH_MAC_Wake_Up_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::MACLPIEvent***
  Holds the LPI event when the an LPI status interrupt occurs. This parameter can be a value of ***ETHEx_LPI_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::IsPtpConfigured***
  Holds the PTP configuration status. This parameter can be a value of ***ETH_PTP_Config_Status***
- ***void(* __ETH_HandleTypeDef::TxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Tx Complete Callback
- ***void(* __ETH_HandleTypeDef::RxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Rx Complete Callback
- ***void(* __ETH_HandleTypeDef::ErrorCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Error Callback
- ***void(* __ETH_HandleTypeDef::PMTCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Power Management Callback
- ***void(* __ETH_HandleTypeDef::EEECallback)(struct __ETH_HandleTypeDef *heth)***
  ETH EEE Callback
- ***void(* __ETH_HandleTypeDef::WakeUpCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Wake UP Callback
- ***void(* __ETH_HandleTypeDef::MspInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp Init callback
- ***void(* __ETH_HandleTypeDef::MspDeInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp DeInit callback
- ***pETH_rxAllocateCallbackTypeDef __ETH_HandleTypeDef::rxAllocateCallback***
  ETH Rx Get Buffer Function
- ***pETH_rxLinkCallbackTypeDef __ETH_HandleTypeDef::rxLinkCallback***
  ETH Rx Set App Data Function
- ***pETH_txFreeCallbackTypeDef __ETH_HandleTypeDef::txFreeCallback***
  ETH Tx Free Function
- ***pETH_txPtpCallbackTypeDef __ETH_HandleTypeDef::txPtpCallback***
  ETH Tx Handle Ptp Function

24.1.11 ETH_MACFilterConfigTypeDef

ETH_MACFilterConfigTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***FunctionalState PromiscuousMode***
- ***FunctionalState ReceiveAllMode***
- ***FunctionalState HachOrPerfectFilter***
- ***FunctionalState HashUnicast***
- ***FunctionalState HashMulticast***
- ***FunctionalState PassAllMulticast***
- ***FunctionalState SrcAddrFiltering***
- ***FunctionalState SrcAddrInverseFiltering***
- ***FunctionalState DestAddrInverseFiltering***
- ***FunctionalState BroadcastFilter***
- ***uint32_t ControlPacketsFilter***

Field Documentation

- ***FunctionalState ETH_MACFilterConfigTypeDef::PromiscuousMode***
  Enable or Disable Promiscuous Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::ReceiveAllMode***
  Enable or Disable Receive All Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::HachOrPerfectFilter***
  Enable or Disable Perfect filtering in addition to Hash filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::HashUnicast***
  Enable or Disable Hash filtering on unicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::HashMulticast***
  Enable or Disable Hash filtering on multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::PassAllMulticast***
  Enable or Disable passing all multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrFiltering***
  Enable or Disable source address filtering module
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrInverseFiltering***
  Enable or Disable source address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::DestAddrInverseFiltering***
  Enable or Disable destination address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::BroadcastFilter***
  Enable or Disable broadcast filter
- ***uint32_t ETH_MACFilterConfigTypeDef::ControlPacketsFilter***
  Set the control packets filter This parameter can be a value of ***ETH_Control_Packets_Filter***

24.1.12 ETH_PowerDownConfigTypeDef

ETH_PowerDownConfigTypeDef is defined in the stm32h5xx_hal_eth.h

Data Fields

- ***FunctionalState WakeUpPacket***
- ***FunctionalState MagicPacket***
- ***FunctionalState GlobalUnicast***
- ***FunctionalState WakeUpForward***

Field Documentation

- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpPacket***
  Enable or Disable Wake up packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::MagicPacket***
  Enable or Disable Magic packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::GlobalUnicast***
  Enable or Disable Global unicast packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpForward***
  Enable or Disable Forwarding Wake up packets

24.2 ETH Firmware driver API description

The following section lists the various functions of the ETH library.

24.2.1 How to use this driver

The ETH HAL driver can be used as follows:

1. Declare a ETH_HandleTypeDef handle structure, for example: ETH_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL_ETH_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL_ETH_MspInit() API:
   a. Enable the Ethernet interface clock using
      - __HAL_RCC_ETH1MAC_CLK_ENABLE()
      - __HAL_RCC_ETH1TX_CLK_ENABLE()
      - __HAL_RCC_ETH1RX_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure Ethernet pinout
   d. Configure Ethernet NVIC interrupt (in Interrupt mode)
5. Ethernet data reception is asynchronous, so call the following API to start the listening mode:
   a. HAL_ETH_Start(): This API starts the MAC and DMA transmission and reception process, without enabling end of transfer interrupts, in this mode user has to poll for data reception by calling HAL_ETH_ReadData()
   b. HAL_ETH_Start_IT(): This API starts the MAC and DMA transmission and reception process, end of transfer interrupts are enabled in this mode, HAL_ETH_RxCpltCallback() will be executed when an Ethernet packet is received
6. When data is received user can call the following API to get received data:
   a. HAL_ETH_ReadData(): Read a received packet
7. For transmission path, two APIs are available:
   a. HAL_ETH_Transmit(): Transmit an ETH frame in blocking mode
   b. HAL_ETH_Transmit_IT(): Transmit an ETH frame in interrupt mode, HAL_ETH_TxCpltCallback() will be executed when end of transfer occur
8. Communication with an external PHY device:
   a. HAL_ETH_ReadPHYRegister(): Read a register from an external PHY
   b. HAL_ETH_WritePHYRegister(): Write data to an external RHY register
9. Configure the Ethernet MAC after ETH peripheral initialization
   a. HAL_ETH_GetMACConfig(): Get MAC actual configuration into ETH_MACConfigTypeDef
   b. HAL_ETH_SetMACConfig(): Set MAC configuration based on ETH_MACConfigTypeDef
10. Configure the Ethernet DMA after ETH peripheral initialization
    a. HAL_ETH_GetDMAConfig(): Get DMA actual configuration into ETH_DMAConfigTypeDef
    b. HAL_ETH_SetDMAConfig(): Set DMA configuration based on ETH_DMAConfigTypeDef
11. Configure the Ethernet PTP after ETH peripheral initialization
    a. Define HAL_ETH_USE_PTP to use PTP APIs.
    b. HAL_ETH_PTP_GetConfig(): Get PTP actual configuration into ETH_PTP_ConfigTypeDef
    c. HAL_ETH_PTP_SetConfig(): Set PTP configuration based on ETH_PTP_ConfigTypeDef
    d. HAL_ETH_PTP_GetTime(): Get Seconds and Nanoseconds for the Ethernet PTP registers
    e. HAL_ETH_PTP_SetTime(): Set Seconds and Nanoseconds for the Ethernet PTP registers
    f. HAL_ETH_PTP_AddTimeOffset(): Add Seconds and Nanoseconds offset for the Ethernet PTP registers
    g. HAL_ETH_PTP_InsertTxTimestamp(): Insert Timestamp in transmission
    h. HAL_ETH_PTP_GetTxTimestamp(): Get transmission timestamp
    i. HAL_ETH_PTP_GetRxTimestamp(): Get reception timestamp -@- The ARP offload feature is not supported in this driver. -@- The PTP offload feature is not supported in this driver.

Callback registration

24.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize and deinitialize the ETH peripheral:

- User must Implement HAL_ETH_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO and NVIC).
- Call the function HAL_ETH_Init() to configure the selected device with the selected configuration:
  - MAC address
  - Media interface (MII or RMII)
  - Rx DMA Descriptors Tab
  - Tx DMA Descriptors Tab
  - Length of Rx Buffers
- Call the function HAL_ETH_DeInit() to restore the default configuration of the selected ETH peripheral.

This section contains the following APIs:

- ***HAL_ETH_Init()***
- ***HAL_ETH_DeInit()***
- ***HAL_ETH_MspInit()***
- ***HAL_ETH_MspDeInit()***
- ***HAL_ETH_RegisterCallback()***
- ***HAL_ETH_UnRegisterCallback()***

24.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the ETH data transfer.

This section contains the following APIs:

- ***HAL_ETH_Start()***
- ***HAL_ETH_Start_IT()***
- ***HAL_ETH_Stop()***
- ***HAL_ETH_Stop_IT()***
- ***HAL_ETH_Transmit()***
- ***HAL_ETH_Transmit_IT()***
- ***HAL_ETH_ReadData()***
- ***HAL_ETH_RegisterRxAllocateCallback()***
- ***HAL_ETH_UnRegisterRxAllocateCallback()***
- ***HAL_ETH_RxAllocateCallback()***
- ***HAL_ETH_RxLinkCallback()***
- ***HAL_ETH_RegisterRxLinkCallback()***
- ***HAL_ETH_UnRegisterRxLinkCallback()***
- ***HAL_ETH_GetRxDataErrorCode()***
- ***HAL_ETH_RegisterTxFreeCallback()***
- ***HAL_ETH_UnRegisterTxFreeCallback()***
- ***HAL_ETH_TxFreeCallback()***
- ***HAL_ETH_ReleaseTxPacket()***
- ***HAL_ETH_IRQHandler()***
- ***HAL_ETH_TxCpltCallback()***
- ***HAL_ETH_RxCpltCallback()***
- ***HAL_ETH_ErrorCallback()***
- ***HAL_ETH_PMTCallback()***
- ***HAL_ETH_EEECallback()***
- ***HAL_ETH_WakeUpCallback()***

- *HAL_ETH_ReadPHYRegister()*
- *HAL_ETH_WritePHYRegister()*
- *HAL_ETH_TxPtpCallback()*

24.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the ETH peripheral.

This section contains the following APIs:

- *HAL_ETH_GetMACConfig()*
- *HAL_ETH_GetDMAConfig()*
- *HAL_ETH_SetMACConfig()*
- *HAL_ETH_SetDMAConfig()*
- *HAL_ETH_SetMDIOClockRange()*
- *HAL_ETH_SetMACFilterConfig()*
- *HAL_ETH_GetMACFilterConfig()*
- *HAL_ETH_SetSourceMACAddrMatch()*
- *HAL_ETH_SetHashTable()*
- *HAL_ETH_SetRxVLANIdentifier()*
- *HAL_ETH_EnterPowerDownMode()*
- *HAL_ETH_ExitPowerDownMode()*
- *HAL_ETH_SetWakeUpFilter()*

24.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of ETH communication process, return Peripheral Errors occurred during communication process

This section contains the following APIs:

- *HAL_ETH_GetState()*
- *HAL_ETH_GetError()*
- *HAL_ETH_GetDMAError()*
- *HAL_ETH_GetMACError()*
- *HAL_ETH_GetMACWakeUpSource()*

24.2.6 Detailed description of functions

HAL_ETH_Init

Function name

HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)

Function description

Initialize the Ethernet peripheral registers.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_DeInit

Function name

HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)

Function description

DeInitializes the ETH peripheral.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_MspInit

Function name

void HAL_ETH_MspInit (ETH_HandleTypeDef * heth)

Function description

Initializes the ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_MspDeInit

Function name

void HAL_ETH_MspDeInit (ETH_HandleTypeDef * heth)

Function description

DeInitializes ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID, pETH_CallbackTypeDef pCallback)

Function description

Register a User ETH Callback To be used instead of the weak predefined callback.

Parameters

- **heth:** eth handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_ETH_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_ETH_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_ETH_ERROR_CB_ID Error Callback ID
  - HAL_ETH_PMT_CB_ID Power Management Callback ID
  - HAL_ETH_EEE_CB_ID EEE Callback ID
  - HAL_ETH_WAKEUP_CB_ID Wake UP Callback ID
  - HAL_ETH_MSPINIT_CB_ID MspInit callback ID
  - HAL_ETH_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_ETH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID)

Function description

Unregister an ETH Callback ETH callback is redirected to the weak predefined callback.

Parameters

- **heth:** eth handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_ETH_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_ETH_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_ETH_ERROR_CB_ID Error Callback ID
  - HAL_ETH_PMT_CB_ID Power Management Callback ID
  - HAL_ETH_EEE_CB_ID EEE Callback ID
  - HAL_ETH_WAKEUP_CB_ID Wake UP Callback ID
  - HAL_ETH_MSPINIT_CB_ID MspInit callback ID
  - HAL_ETH_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_ETH_Start

Function name

HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception and transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Start_IT

Function name

HAL_StatusTypeDef HAL_ETH_Start_IT (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop

Function name

HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)

Function description

Stop Ethernet MAC and DMA reception/transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop_IT

Function name

HAL_StatusTypeDef HAL_ETH_Stop_IT (ETH_HandleTypeDef * heth)

Function description

Stop Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_ReadData

Function name

HAL_StatusTypeDef HAL_ETH_ReadData (ETH_HandleTypeDef * heth, void ** pAppBuff)

Function description

Read a received packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pAppBuff:** Pointer to an application buffer to receive the packet.

Return values

- **HAL:** status

HAL_ETH_RegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxAllocateCallback (ETH_HandleTypeDef * heth, pETH_rxAllocateCallbackTypeDef rxAllocateCallback)

Function description

Register the Rx alloc callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **rxAllocateCallback:** pointer to function to alloc buffer

Return values

- **HAL:** status

HAL_ETH_UnRegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxAllocateCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Rx alloc callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_RegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxLinkCallback (ETH_HandleTypeDef * heth, pETH_rxLinkCallbackTypeDef rxLinkCallback)

Function description

Set the Rx link data function.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **rxLinkCallback:** pointer to function to link data

Return values

- **HAL:** status

HAL_ETH_UnRegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxLinkCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Rx link callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_GetRxDataErrorCode

Function name

HAL_StatusTypeDef HAL_ETH_GetRxDataErrorCode (ETH_HandleTypeDef * heth, uint32_t * pErrorCode)

Function description

Get the error state of the last received packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pErrorCode:** pointer to uint32_t to hold the error code

Return values

- **HAL:** status

HAL_ETH_RegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterTxFreeCallback (ETH_HandleTypeDef * heth, pETH_txFreeCallbackTypeDef txFreeCallback)

Function description

Set the Tx free function.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **txFreeCallback:** pointer to function to release the packet

Return values

- **HAL:** status

HAL_ETH_UnRegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterTxFreeCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Tx free callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_ReleaseTxPacket

Function name

HAL_StatusTypeDef HAL_ETH_ReleaseTxPacket (ETH_HandleTypeDef * heth)

Function description

Release transmitted Tx packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Transmit

Function name

HAL_StatusTypeDef HAL_ETH_Transmit (ETH_HandleTypeDef * heth, ETH_TxPacketConfig * pTxConfig, uint32_t Timeout)

Function description

Sends an Ethernet Packet in polling mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted
- **Timeout:** timeout value

Return values

- **HAL:** status

HAL_ETH_Transmit_IT

Function name

HAL_StatusTypeDef HAL_ETH_Transmit_IT (ETH_HandleTypeDef * heth, ETH_TxPacketConfig * pTxConfig)

Function description

Sends an Ethernet Packet in interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted

Return values

- **HAL:** status

HAL_ETH_WritePHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_WritePHYRegister (const ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t RegValue)

Function description

Writes to a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **RegValue:** the value to write

Return values

- **HAL:** status

HAL_ETH_ReadPHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t * pRegValue)

Function description

Read a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **pRegValue:** parameter to hold read value

Return values

- **HAL:** status

HAL_ETH_IRQHandler

Function name

void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)

Function description

This function handles ETH interrupt request.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_TxCpltCallback

Function name

void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)

Function description

Tx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RxCpltCallback

Function name

void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)

Function description

Rx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_ErrorCallback

Function name

void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)

Function description

Ethernet transfer error callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_PMTCallback

Function name

void HAL_ETH_PMTCallback (ETH_HandleTypeDef * heth)

Function description

Ethernet Power Management module IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_EEECallback

Function name

void HAL_ETH_EEECallback (ETH_HandleTypeDef * heth)

Function description

Energy Efficient Etherent IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_WakeUpCallback

Function name

void HAL_ETH_WakeUpCallback (ETH_HandleTypeDef * heth)

Function description

ETH WAKEUP interrupt callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RxAllocateCallback

Function name

void HAL_ETH_RxAllocateCallback (uint8_t ** buff)

Function description

Rx Allocate callback.

Parameters

- **buff:** pointer to allocated buffer

Return values

- **None:**

HAL_ETH_RxLinkCallback

Function name

void HAL_ETH_RxLinkCallback (void ** pStart, void ** pEnd, uint8_t * buff, uint16_t Length)

Function description

Rx Link callback.

Parameters

- **pStart:** pointer to packet start
- **pEnd:** pointer to packet end
- **buff:** pointer to received data
- **Length:** received data length

Return values

- **None:**

HAL_ETH_TxFreeCallback

Function name

void HAL_ETH_TxFreeCallback (uint32_t * buff)

Function description

Tx Free callback.

Parameters

- **buff:** pointer to buffer to free

Return values

- **None:**

HAL_ETH_TxPtpCallback

Function name

void HAL_ETH_TxPtpCallback (uint32_t * buff, ETH_TimeStampTypeDef * timestamp)

Function description

HAL_ETH_GetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

Get the configuration of the MAC and MTL subsystems.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that will hold the configuration of the MAC.

Return values

- **HAL:** Status

HAL_ETH_GetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

Get the configuration of the DMA.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** Status

HAL_ETH_SetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

Set the MAC configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that contains the configuration of the MAC.

Return values

- **HAL:** status

HAL_ETH_SetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

Set the ETH DMA configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** status

HAL_ETH_SetMDIOClockRange

Function name

void HAL_ETH_SetMDIOClockRange (ETH_HandleTypeDef * heth)

Function description

Configures the Clock range of ETH MDIO interface.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_SetRxVLANIdentifier

Function name

void HAL_ETH_SetRxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t ComparisonBits, uint32_t VLANIdentifier)

Function description

Set the VLAN Identifier for Rx packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **ComparisonBits:** 12 or 16 bit comparison mode must be a value of ETH VLAN Tag Comparison
- **VLANIdentifier:** VLAN Identifier value

Return values

- **None:**

HAL_ETH_GetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACFilterConfig (ETH_HandleTypeDef * heth, ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

Get the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that will hold the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACFilterConfig (ETH_HandleTypeDef * heth, const ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

Set the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that contains the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetHashTable

Function name

HAL_StatusTypeDef HAL_ETH_SetHashTable (ETH_HandleTypeDef * heth, uint32_t * pHashTable)

Function description

Set the ETH Hash Table Value.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pHashTable:** pointer to a table of two 32 bit values, that contains the 64 bits of the hash table.

Return values

- **HAL:** status

HAL_ETH_SetSourceMACAddrMatch

Function name

HAL_StatusTypeDef HAL_ETH_SetSourceMACAddrMatch (const ETH_HandleTypeDef * heth, uint32_t AddrNbr, const uint8_t * pMACAddr)

Function description

Set the source MAC Address to be matched.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **AddrNbr:** The MAC address to configure This parameter must be a value of the following: ETH_MAC_ADDRESS1 ETH_MAC_ADDRESS2 ETH_MAC_ADDRESS3
- **pMACAddr:** Pointer to MAC address buffer data (6 bytes)

Return values

- **HAL:** status

HAL_ETH_EnterPowerDownMode

Function name

void HAL_ETH_EnterPowerDownMode (ETH_HandleTypeDef * heth, const ETH_PowerDownConfigTypeDef * pPowerDownConfig)

Function description

Enters the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pPowerDownConfig:** a pointer to ETH_PowerDownConfigTypeDef structure that contains the Power Down configuration

Return values

- **None.:**

HAL_ETH_ExitPowerDownMode

Function name

void HAL_ETH_ExitPowerDownMode (ETH_HandleTypeDef * heth)

Function description

Exits from the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETH_SetWakeUpFilter

Function name

HAL_StatusTypeDef HAL_ETH_SetWakeUpFilter (ETH_HandleTypeDef * heth, uint32_t * pFilter, uint32_t Count)

Function description

Set the WakeUp filter.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilter:** pointer to filter registers values
- **Count:** number of filter registers, must be from 1 to 8.

Return values

- **None.:**

HAL_ETH_GetState

Function name

HAL_ETH_StateTypeDef HAL_ETH_GetState (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH state.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** state

HAL_ETH_GetError

Function name

uint32_t HAL_ETH_GetError (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** Error Code

HAL_ETH_GetDMAError

Function name

uint32_t HAL_ETH_GetDMAError (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH DMA error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** DMA Error Code

HAL_ETH_GetMACError

Function name

uint32_t HAL_ETH_GetMACError (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC Error Code

HAL_ETH_GetMACWakeUpSource

Function name

uint32_t HAL_ETH_GetMACWakeUpSource (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC WakeUp event source.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event source

24.3 ETH Firmware driver defines

The following section lists the various define and macros of the module.

24.3.1 ETH

ETH

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Burst Mode

ETH_BURSTLENGTH_FIXED

ETH_BURSTLENGTH_MIXED

ETH_BURSTLENGTH_UNSPECIFIED

ETH Control Packets Filter

ETH_CTRLPACKETS_BLOCK_ALL

ETH_CTRLPACKETS_FORWARD_ALL_EXCEPT_PA

ETH_CTRLPACKETS_FORWARD_ALL

ETH_CTRLPACKETS_FORWARD_PASSED_ADDR_FILTER

ETH DMA Arbitration

ETH_DMAARBITRATION_RX

ETH_DMAARBITRATION_RX1_TX1

ETH_DMAARBITRATION_RX2_TX1

ETH_DMAARBITRATION_RX3_TX1

ETH_DMAARBITRATION_RX4_TX1

ETH_DMAARBITRATION_RX5_TX1

ETH_DMAARBITRATION_RX6_TX1

ETH_DMAARBITRATION_RX7_TX1

ETH_DMAARBITRATION_RX8_TX1

ETH_DMAARBITRATION_TX

ETH_DMAARBITRATION_TX1_RX1

ETH_DMAARBITRATION_TX2_RX1

ETH_DMAARBITRATION_TX3_RX1

ETH_DMAARBITRATION_TX4_RX1

ETH_DMAARBITRATION_TX5_RX1

ETH_DMAARBITRATION_TX6_RX1

ETH_DMAARBITRATION_TX7_RX1

ETH_DMAARBITRATION_TX8_RX1

ETH DMA Interrupts

ETH_DMA_NORMAL_IT

ETH_DMA_ABNORMAL_IT

ETH_DMA_CONTEXT_DESC_ERROR_IT

ETH_DMA_FATAL_BUS_ERROR_IT

ETH_DMA_EARLY_RX_IT

ETH_DMA_EARLY_TX_IT

ETH_DMA_RX_WATCHDOG_TIMEOUT_IT

ETH_DMA_RX_PROCESS_STOPPED_IT

ETH_DMA_RX_BUFFER_UNAVAILABLE_IT

ETH_DMA_RX_IT

ETH_DMA_TX_BUFFER_UNAVAILABLE_IT

ETH_DMA_TX_PROCESS_STOPPED_IT

ETH_DMA_TX_IT

ETH DMA Rx Descriptor Bit Definition

ETH_DMARXNDESCRF_BUF1AP

Header or Buffer 1 Address Pointer

ETH_DMARXNDESCRF_BUF2AP

Buffer 2 Address Pointer

ETH_DMARXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMARXNDESCRF_IOC

Interrupt Enabled on Completion

ETH_DMARXNDESCRF_BUF2V

Buffer 2 Address Valid

ETH_DMARXNDESCRF_BUF1V

Buffer 1 Address Valid

ETH_DMARXNDESCWBF_IVT

Inner VLAN Tag

ETH_DMARXNDESCWBF_OVT

Outer VLAN Tag

ETH_DMARXNDESCWBF_OPC

OAM Sub-Type Code, or MAC Control Packet opcode

ETH_DMARXNDESCWBF_TD

Timestamp Dropped

ETH_DMARXNDESCWBF_TSA

Timestamp Available

ETH_DMARXNDESCWBF_PV

PTP Version

ETH_DMARXNDESCWBF_PFT

PTP Packet Type

ETH_DMARXNDESCWBF_PMT_NO

PTP Message Type: No PTP message received

ETH_DMARXNDESCWBF_PMT_SYNC

PTP Message Type: SYNC (all clock types)

ETH_DMARXNDESCWBF_PMT_FUP

PTP Message Type: Follow_Up (all clock types)

ETH_DMARXNDESCWBF_PMT_DREQ

PTP Message Type: Delay_Req (all clock types)

ETH_DMARXNDESCWBF_PMT_DRESP

PTP Message Type: Delay_Resp (all clock types)

ETH_DMARXNDESCWBF_PMT_PDREQ

PTP Message Type: Pdelay_Req (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESP

PTP Message Type: Pdelay_Resp (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESPFUP

PTP Message Type: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_ANNOUNCE

PTP Message Type: Announce

ETH_DMARXNDESCWBF_PMT_MANAG

PTP Message Type: Management

ETH_DMARXNDESCWBF_PMT_SIGN

PTP Message Type: Signaling

ETH_DMARXNDESCWBF_PMT_RESERVED

PTP Message Type: PTP packet with Reserved message type

ETH_DMARXNDESCWBF_IPCE

IP Payload Error

ETH_DMARXNDESCWBF_IPCB

IP Checksum Bypassed

ETH_DMARXNDESCWBF_IPV6

IPv6 header Present

ETH_DMARXNDESCWBF_IPV4

IPv4 header Present

ETH_DMARXNDESCWBF_IPHE

IP Header Error

ETH_DMARXNDESCWBF_PT

Payload Type mask

ETH_DMARXNDESCWBF_PT_UNKNOWN

Payload Type: Unknown type or IP/AV payload not processed

ETH_DMARXNDESCWBF_PT_UDP

Payload Type: UDP

ETH_DMARXNDESCWBF_PT_TCP

Payload Type: TCP

ETH_DMARXNDESCWBF_PT_ICMP

Payload Type: ICMP

ETH_DMARXNDESCWBF_L3L4FM

L3 and L4 Filter Number Matched: if reset filter 0 is matched , if set filter 1 is matched

ETH_DMARXNDESCWBF_L4FM

Layer 4 Filter Match

ETH_DMARXNDESCWBF_L3FM

Layer 3 Filter Match

ETH_DMARXNDESCWBF_MADRM

MAC Address Match or Hash Value

ETH_DMARXNDESCWBF_HF

Hash Filter Status

ETH_DMARXNDESCWBF_DAF

Destination Address Filter Fail

ETH_DMARXNDESCWBF_SAF

SA Address Filter Fail

ETH_DMARXNDESCWBF_VF

VLAN Filter Status

ETH_DMARXNDESCWBF_ARPNR

ARP Reply Not Generated

ETH_DMARXNDESCWBF_OWN

Own Bit

ETH_DMARXNDESCWBF_CTXT

Receive Context Descriptor

ETH_DMARXNDESCWBF_FD

First Descriptor

ETH_DMARXNDESCWBF_LD

Last Descriptor

ETH_DMARXNDESCWBF_RS2V

Receive Status RDES2 Valid

ETH_DMARXNDESCWBF_RS1V

Receive Status RDES1 Valid

ETH_DMARXNDESCWBF_RS0V

Receive Status RDES0 Valid

ETH_DMARXNDESCWBF_CE

CRC Error

ETH_DMARXNDESCWBF_GP

Giant Packet

ETH_DMARXNDESCWBF_RWT

Receive Watchdog Timeout

ETH_DMARXNDESCWBF_OE

Overflow Error

ETH_DMARXNDESCWBF_RE

Receive Error

ETH_DMARXNDESCWBF_DE

Dribble Bit Error

ETH_DMARXNDESCWBF_LT

Length/Type Field

ETH_DMARXNDESCWBF_LT_LP

The packet is a length packet

ETH_DMARXNDESCWBF_LT_TP

The packet is a type packet

ETH_DMARXNDESCWBF_LT_ARP

The packet is a ARP Request packet type

ETH_DMARXNDESCWBF_LT_VLAN

The packet is a type packet with VLAN Tag

ETH_DMARXNDESCWBF_LT_DVLAN

The packet is a type packet with Double VLAN Tag

ETH_DMARXNDESCWBF_LT_MAC

The packet is a MAC Control packet type

ETH_DMARXNDESCWBF_LT_OAM

The packet is a OAM packet type

ETH_DMARXNDESCWBF_ES

Error Summary

ETH_DMARXNDESCWBF_PL

Packet Length

ETH_DMARXCDESC_RTSL

Receive Packet Timestamp Low

ETH_DMARXCDESC_RTSH

Receive Packet Timestamp High

ETH_DMARXCDESC_OWN

Own Bit

ETH_DMARXCDESC_CTXT

Receive Context Descriptor

ETH DMA Status Flags

ETH_DMA_RX_NO_ERROR_FLAG

ETH_DMA_RX_DESC_READ_ERROR_FLAG

ETH_DMA_RX_DESC_WRITE_ERROR_FLAG

ETH_DMA_RX_BUFFER_READ_ERROR_FLAG

ETH_DMA_RX_BUFFER_WRITE_ERROR_FLAG

ETH_DMA_TX_NO_ERROR_FLAG

ETH_DMA_TX_DESC_READ_ERROR_FLAG

ETH_DMA_TX_DESC_WRITE_ERROR_FLAG

ETH_DMA_TX_BUFFER_READ_ERROR_FLAG

ETH_DMA_TX_BUFFER_WRITE_ERROR_FLAG

ETH_DMA_CONTEXT_DESC_ERROR_FLAG

ETH_DMA_FATAL_BUS_ERROR_FLAG

ETH_DMA_EARLY_TX_IT_FLAG

ETH_DMA_RX_WATCHDOG_TIMEOUT_FLAG

ETH_DMA_RX_PROCESS_STOPPED_FLAG

ETH_DMA_RX_BUFFER_UNAVAILABLE_FLAG

ETH_DMA_TX_PROCESS_STOPPED_FLAG

ETH DMA Tx Descriptor Bit Definition

ETH_DMATXNDESCRF_B1AP

Transmit Packet Timestamp Low

ETH_DMATXNDESCRF_B2AP

Transmit Packet Timestamp High

ETH_DMATXNDESCRF_IOC

Interrupt on Completion

ETH_DMATXNDESCRF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCRF_B2L

Buffer 2 Length

ETH_DMATXNDESCRF_VTIR

VLAN Tag Insertion or Replacement mask

ETH_DMATXNDESCRF_VTIR_DISABLE

Do not add a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REMOVE

Remove the VLAN tag from the packets before transmission.

ETH_DMATXNDESCRF_VTIR_INSERT

Insert a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REPLACE

Replace the VLAN tag.

ETH_DMATXNDESCRF_B1L

Buffer 1 Length

ETH_DMATXNDESCRF_HL

Header Length

ETH_DMATXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCRF_CTXT

Context Type

ETH_DMATXNDESCRF_FD

First Descriptor

ETH_DMATXNDESCRF_LD

Last Descriptor

ETH_DMATXNDESCRF_CPC

CRC Pad Control mask

ETH_DMATXNDESCRF_CPC_CRCPAD_INSERT

CRC Pad Control: CRC and Pad Insertion

ETH_DMATXNDESCRF_CPC_CRC_INSERT

CRC Pad Control: CRC Insertion (Disable Pad Insertion)

ETH_DMATXNDESCRF_CPC_DISABLE

CRC Pad Control: Disable CRC Insertion

ETH_DMATXNDESCRF_CPC_CRC_REPLACE

CRC Pad Control: CRC Replacement

ETH_DMATXNDESCRF_SAIC

SA Insertion Control mask

ETH_DMATXNDESCRF_SAIC_DISABLE

SA Insertion Control: Do not include the source address

ETH_DMATXNDESCRF_SAIC_INSERT

SA Insertion Control: Include or insert the source address

ETH_DMATXNDESCRF_SAIC_REPLACE

SA Insertion Control: Replace the source address

ETH_DMATXNDESCRF_THL

TCP Header Length

ETH_DMATXNDESCRF_TSE

TCP segmentation enable

ETH_DMATXNDESCRF_CIC

Checksum Insertion Control: 4 cases

ETH_DMATXNDESCRF_CIC_DISABLE

Do Nothing: Checksum Engine is disabled

ETH_DMATXNDESCRF_CIC_IPHDR_INSERT

Only IP header checksum calculation and insertion are enabled.

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT

IP header checksum and payload checksum calculation and insertion are enabled, but pseudo header checksum is not calculated in hardware

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT_PHDR_CALC

IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo header checksum is calculated in hardware.

ETH_DMATXNDESCRF_TPL

TCP Payload Length

ETH_DMATXNDESCRF_FL

Transmit End of Ring

ETH_DMATXNDESCWBF_TTSL

Buffer1 Address Pointer or TSO Header Address Pointer

ETH_DMATXNDESCWBF_TTSH

Buffer2 Address Pointer

ETH_DMATXNDESCWBF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCWBF_CTXT

Context Type

ETH_DMATXNDESCWBF_FD

First Descriptor

ETH_DMATXNDESCWBF_LD

Last Descriptor

ETH_DMATXNDESCWBF_TTSS

Tx Timestamp Status

ETH_DMATXNDESCWBF_DP

Disable Padding

ETH_DMATXNDESCWBF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCWBF_ES

Error summary: OR of the following bits: IHE || UF || ED || EC || LCO || PCE || NC || LCA || FF || JT

ETH_DMATXNDESCWBF_JT

Jabber Timeout

ETH_DMATXNDESCWBF_FF

Packet Flushed: DMA/MTL flushed the packet due to SW flush

ETH_DMATXNDESCWBF_PCE

Payload Checksum Error

ETH_DMATXNDESCWBF_LCA

Loss of Carrier: carrier lost during transmission

ETH_DMATXNDESCWBF_NC

No Carrier: no carrier signal from the transceiver

ETH_DMATXNDESCWBF_LCO

Late Collision: transmission aborted due to collision

ETH_DMATXNDESCWBF_EC

Excessive Collision: transmission aborted after 16 collisions

ETH_DMATXNDESCWBF_CC

Collision Count

ETH_DMATXNDESCWBF_ED

Excessive Deferral

ETH_DMATXNDESCWBF_UF

Underflow Error: late data arrival from the memory

ETH_DMATXNDESCWBF_DB

Deferred Bit

ETH_DMATXNDESCWBF_IHE

IP Header Error

ETH_DMATXCDESC_TTSL

Transmit Packet Timestamp Low

ETH_DMATXCDESC_TTSH

Transmit Packet Timestamp High

ETH_DMATXCDESC_IVT

Inner VLAN Tag

ETH_DMATXCDESC_MSS

Maximum Segment Size

ETH_DMATXCDESC_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXCDESC_CTXT

Context Type

ETH_DMATXCDESC_OSTC

One-Step Timestamp Correction Enable

ETH_DMATXCDESC_TCMSSV

One-Step Timestamp Correction Input or MSS Valid

ETH_DMATXCDESC_CDE

Context Descriptor Error

ETH_DMATXCDESC_IVTIR

Inner VLAN Tag Insert or Replace Mask

ETH_DMATXCDESC_IVTIR_DISABLE

Do not add the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REMOVE

Remove the inner VLAN tag from the packets before transmission.

ETH_DMATXCDESC_IVTIR_INSERT

Insert the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REPLACE

Replace the inner VLAN tag.

ETH_DMATXCDESC_IVLTV

Inner VLAN Tag Valid

ETH_DMATXCDESC_VLTV

VLAN Tag Valid

ETH_DMATXCDESC_VT

VLAN Tag

ETH Duplex Mode

ETH_FULLDUPLEX_MODE

ETH_HALFDUPLEX_MODE

ETH Error Code

HAL_ETH_ERROR_NONE

No error

HAL_ETH_ERROR_PARAM

Busy error

HAL_ETH_ERROR_BUSY

Parameter error

HAL_ETH_ERROR_TIMEOUT

Timeout error

HAL_ETH_ERROR_DMA

DMA transfer error

HAL_ETH_ERROR_MAC

MAC transfer error

HAL_ETH_ERROR_INVALID_CALLBACK

Invalid Callback error

ETH Exported Macros

__HAL_ETH_RESET_HANDLE_STATE

Description:

- Reset ETH handle state.

Parameters:

- __HANDLE__: specifies the ETH handle.

Return value:

- None

__HAL_ETH_DMA_ENABLE_IT

Description:

- Enables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be enabled

Return value:

- None

__HAL_ETH_DMA_DISABLE_IT

Description:

- Disables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be disabled.

Return value:

- None

__HAL_ETH_DMA_GET_IT_SOURCE

Description:

- Gets the ETHERNET DMA IT source enabled or disabled.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: ETH DMA IT Source enabled or disabled

__HAL_ETH_DMA_GET_IT

Description:

- Gets the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: state of ETH DMA IT (SET or RESET)

__HAL_ETH_DMA_CLEAR_IT

Description:

- Clears the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

__HAL_ETH_DMA_GET_FLAG

Description:

- Checks whether the specified ETHERNET DMA flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_DMA_CLEAR_FLAG

Description:

- Clears the specified ETHERNET DMA flag.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_MAC_ENABLE_IT

Description:

- Enables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_DISABLE_IT

Description:

- Disables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_GET_IT

Description:

- Checks whether the specified ETHERNET MAC flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __INTERRUPT__: specifies the flag to check.

Return value:

- The: state of ETH MAC IT (SET or RESET). External interrupt line 46 Connected to the ETH wakeup EXTI Line

ETH_WAKEUP_EXTI_LINE

__HAL_ETH_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the ETH WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be enabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified ETH WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- EXTI: ETH WAKEUP Line Status.

__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the ETH WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE

Description:

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_GET_PTP_CONTROL

__HAL_ETH_SET_PTP_CONTROL

ETH frame settings

ETH_MAX_PACKET_SIZE

ETH_HEADER + 2*VLAN_TAG + MAX_ETH_PAYLOAD + ETH_CRC

ETH_HEADER

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

ETH_CRC

Ethernet CRC

ETH_VLAN_TAG

optional 802.1q VLAN Tag

ETH_MIN_PAYLOAD

Minimum Ethernet payload size

ETH_MAX_PAYLOAD

Maximum Ethernet payload size

ETH_JUMBO_FRAME_PAYLOAD

Jumbo frame payload size

ETH Inter Packet Gap

ETH_INTERPACKETGAP_96BIT

ETH_INTERPACKETGAP_88BIT

ETH_INTERPACKETGAP_80BIT

ETH_INTERPACKETGAP_72BIT

ETH_INTERPACKETGAP_64BIT

ETH_INTERPACKETGAP_56BIT

ETH_INTERPACKETGAP_48BIT

ETH_INTERPACKETGAP_40BIT

ETH MAC addresses

ETH_MAC_ADDRESS0

ETH_MAC_ADDRESS1

ETH_MAC_ADDRESS2

ETH_MAC_ADDRESS3

ETH MAC Interrupts

ETH_MAC_RX_STATUS_IT

ETH_MAC_TX_STATUS_IT

ETH_MAC_TIMESTAMP_IT

ETH_MAC_LPI_IT

ETH_MAC_PMT_IT

ETH_MAC_PHY_IT

ETH MAC Rx Tx Status

ETH_RECEIVE_WATCHDOG_TIMEOUT

ETH_EXECESSIVE_COLLISIONS

ETH_LATE_COLLISIONS

ETH_EXECESSIVE_DEFERRAL

ETH_LOSS_OF_CARRIER

ETH_NO_CARRIER

ETH_TRANSMIT_JABBR_TIMEOUT

ETH MAC Wake Up Event

ETH_WAKEUP_PACKET_RECIEVED

ETH_MAGIC_PACKET_RECIEVED

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS_4

ETH_PAUSELOWTHRESHOLD_MINUS_28

ETH_PAUSELOWTHRESHOLD_MINUS_36

ETH_PAUSELOWTHRESHOLD_MINUS_144

ETH_PAUSELOWTHRESHOLD_MINUS_256

ETH_PAUSELOWTHRESHOLD_MINUS_512

ETH Preamble Length

ETH_PREAMBLELENGTH_7

ETH_PREAMBLELENGTH_5

ETH_PREAMBLELENGTH_3

ETH PTP Config Status

HAL_ETH_PTP_NOT_CONFIGURATED

ETH PTP Configuration not done

HAL_ETH_PTP_CONFIGURATED

ETH PTP Configuration done

ETH Receive Mode

ETH_RECEIVESTOREFORWARD

ETH_RECEIVETHRESHOLD8_64

ETH_RECEIVETHRESHOLD8_32

ETH_RECEIVETHRESHOLD8_96

ETH_RECEIVETHRESHOLD8_128

ETH Rx Checksum Status

ETH_CHECKSUM_BYPASSED

ETH_CHECKSUM_IP_HEADER_ERROR

ETH_CHECKSUM_IP_PAYLOAD_ERROR

ETH Rx DMA Burst Length

ETH_RXDMABURSTLENGTH_1BEAT

ETH_RXDMABURSTLENGTH_2BEAT

ETH_RXDMABURSTLENGTH_4BEAT

ETH_RXDMABURSTLENGTH_8BEAT

ETH_RXDMABURSTLENGTH_16BEAT

ETH_RXDMABURSTLENGTH_32BEAT

ETH Rx Error Code

ETH_DRIBBLE_BIT_ERROR

ETH_RECEIVE_ERROR

ETH_RECEIVE_OVERFLOW

ETH_WATCHDOG_TIMEOUT

ETH_GIANT_PACKET

ETH_CRC_ERROR

ETH Rx IP Header Type

ETH_IP_HEADER_IPV4

ETH_IP_HEADER_IPV6

ETH Rx L3 Filter Status

ETH_L3_FILTER0_MATCH

ETH_L3_FILTER1_MATCH

ETH Rx L4 Filter Status

ETH_L4_FILTER0_MATCH

ETH_L4_FILTER1_MATCH

ETH Rx MAC Filter Status

ETH_HASH_FILTER_PASS

ETH_VLAN_FILTER_PASS

ETH_DEST_ADDRESS_FAIL

ETH_SOURCE_ADDRESS_FAIL

ETH Rx Payload Type

ETH_IP_PAYLOAD_UNKNOWN

ETH_IP_PAYLOAD_UDP

ETH_IP_PAYLOAD_TCP

ETH_IP_PAYLOAD_ICMPN

ETH Source Addr Control

ETH_SOURCEADDRESS_DISABLE

ETH_SOURCEADDRESS_INSERT_ADDR0

ETH_SOURCEADDRESS_INSERT_ADDR1

ETH_SOURCEADDRESS_REPLACE_ADDR0

ETH_SOURCEADDRESS_REPLACE_ADDR1

ETH Speed

ETH_SPEED_10M

ETH_SPEED_100M

ETH States

HAL_ETH_STATE_RESET
: Peripheral not yet Initialized or disabled

HAL_ETH_STATE_READY
: Peripheral Communication started

HAL_ETH_STATE_BUSY
: an internal process is ongoing

HAL_ETH_STATE_STARTED
: an internal process is started

HAL_ETH_STATE_ERROR
: Error State

ETH Transmit Mode

ETH_TRANSMITSTOREFORWARD

ETH_TRANSMITTHRESHOLD_32

ETH_TRANSMITTHRESHOLD_64

ETH_TRANSMITTHRESHOLD_96

ETH_TRANSMITTHRESHOLD_128

ETH_TRANSMITTHRESHOLD_192

ETH_TRANSMITTHRESHOLD_256

ETH_TRANSMITTHRESHOLD_384

ETH_TRANSMITTHRESHOLD_512

ETH Tx DMA Burst Length

ETH_TXDMABURSTLENGTH_1BEAT

ETH_TXDMABURSTLENGTH_2BEAT

ETH_TXDMABURSTLENGTH_4BEAT

ETH_TXDMABURSTLENGTH_8BEAT

ETH_TXDMABURSTLENGTH_16BEAT

ETH_TXDMABURSTLENGTH_32BEAT

ETH Tx Packet Attributes

ETH_TX_PACKETS_FEATURES_CSUM

ETH_TX_PACKETS_FEATURES_SAIC

ETH_TX_PACKETS_FEATURES_VLANTAG

ETH_TX_PACKETS_FEATURES_INNERVLANTAG

ETH_TX_PACKETS_FEATURES_TSO

ETH_TX_PACKETS_FEATURES_CRCPAD

ETH Tx Packet Checksum Control

ETH_CHECKSUM_DISABLE

ETH_CHECKSUM_IPHDR_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC

ETH Tx Packet CRC Pad Control

ETH_CRC_PAD_DISABLE

ETH_CRC_PAD_INSERT

ETH_CRC_INSERT

ETH_CRC_REPLACE

ETH Tx Packet Inner VLAN Control

ETH_INNER_VLAN_DISABLE

ETH_INNER_VLAN_REMOVE

ETH_INNER_VLAN_INSERT

ETH_INNER_VLAN_REPLACE

ETH Tx Packet Source Addr Control

ETH_SRC_ADDR_CONTROL_DISABLE

ETH_SRC_ADDR_INSERT

ETH_SRC_ADDR_REPLACE

ETH Tx Packet VLAN Control

ETH_VLAN_DISABLE

ETH_VLAN_REMOVE

ETH_VLAN_INSERT

ETH_VLAN_REPLACE

ETH VLAN Tag Comparison

ETH_VLANTAGCOMPARISON_16BIT

ETH_VLANTAGCOMPARISON_12BIT

ETH Watchdog Timeout

ETH_WATCHDOGTIMEOUT_2KB

ETH_WATCHDOGTIMEOUT_3KB

ETH_WATCHDOGTIMEOUT_4KB

ETH_WATCHDOGTIMEOUT_5KB

ETH_WATCHDOGTIMEOUT_6KB

ETH_WATCHDOGTIMEOUT_7KB

ETH_WATCHDOGTIMEOUT_8KB

ETH_WATCHDOGTIMEOUT_9KB

ETH_WATCHDOGTIMEOUT_10KB

ETH_WATCHDOGTIMEOUT_11KB

ETH_WATCHDOGTIMEOUT_12KB

ETH_WATCHDOGTIMEOUT_13KB

ETH_WATCHDOGTIMEOUT_14KB

ETH_WATCHDOGTIMEOUT_15KB

ETH_WATCHDOGTIMEOUT_16KB

25 HAL ETH Extension Driver

25.1 ETHEx Firmware driver registers structures

25.1.1 ETH_RxVLANConfigTypeDef

ETH_RxVLANConfigTypeDef is defined in the stm32h5xx_hal_eth_ex.h

Data Fields

- ***FunctionalState InnerVLANTagInStatus***
- ***uint32_t StripInnerVLANTag***
- ***FunctionalState InnerVLANTag***
- ***FunctionalState DoubleVLANProcessing***
- ***FunctionalState VLANTagHashTableMatch***
- ***FunctionalState VLANTagInStatus***
- ***uint32_t StripVLANTag***
- ***uint32_t VLANTypeCheck***
- ***FunctionalState VLANTagInverceMatch***

Field Documentation

- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTagInStatus***
  Enables or disables Inner VLAN Tag in Rx Status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripInnerVLANTag***
  Sets the Inner VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_Inner_VLAN_Tag_Stripping***
- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTag***
  Enables or disables Inner VLAN Tag
- ***FunctionalState ETH_RxVLANConfigTypeDef::DoubleVLANProcessing***
  Enable or Disable double VLAN processing
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagHashTableMatch***
  Enable or Disable VLAN Tag Hash Table Match
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInStatus***
  Enable or Disable VLAN Tag in Rx status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripVLANTag***
  Set the VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_VLAN_Tag_Stripping***
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANTypeCheck***
  Enable or Disable VLAN Type Check This parameter can be a value of ***ETHEx_VLAN_Type_Check***
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInverceMatch***
  Enable or disable VLAN Tag Inverse Match

25.1.2 ETH_TxVLANConfigTypeDef

ETH_TxVLANConfigTypeDef is defined in the stm32h5xx_hal_eth_ex.h

Data Fields

- ***FunctionalState SourceTxDesc***
- ***FunctionalState SVLANType***
- ***uint32_t VLANTagControl***

Field Documentation

- ***FunctionalState ETH_TxVLANConfigTypeDef::SourceTxDesc***
  Enable or Disable VLAN tag source from DMA tx descriptors
- ***FunctionalState ETH_TxVLANConfigTypeDef::SVLANType***
  Enable or Disable insertion of SVLAN type
- ***uint32_t ETH_TxVLANConfigTypeDef::VLANTagControl***
  Sets the VLAN tag control in tx packets This parameter can be a value of ***ETHEx_VLAN_Tag_Control***

25.1.3 ETH_L3FilterConfigTypeDef

ETH_L3FilterConfigTypeDef is defined in the stm32h5xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcAddrFilterMatch***
- ***uint32_t DestAddrFilterMatch***
- ***uint32_t SrcAddrHigherBitsMatch***
- ***uint32_t DestAddrHigherBitsMatch***
- ***uint32_t Ip4SrcAddr***
- ***uint32_t Ip4DestAddr***
- ***uint32_t Ip6Addr***

Field Documentation

- ***uint32_t ETH_L3FilterConfigTypeDef::Protocol***
  Sets the L3 filter protocol to IPv4 or IPv6 This parameter can be a value of ***ETHEx_L3_Protocol***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrFilterMatch***
  Sets the L3 filter source address match This parameter can be a value of ***ETHEx_L3_Source_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrFilterMatch***
  Sets the L3 filter destination address match This parameter can be a value of ***ETHEx_L3_Destination_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrHigherBitsMatch***
  Sets the L3 filter source address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrHigherBitsMatch***
  Sets the L3 filter destination address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4SrcAddr***
  Sets the L3 filter IPv4 source address if IPv4 protocol is used This parameter can be a value from 0x0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4DestAddr***
  Sets the L3 filter IPv4 destination address if IPv4 protocol is used This parameter can be a value from 0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip6Addr[4]***
  Sets the L3 filter IPv6 address if IPv6 protocol is used This parameter must be a table of 4 words (4* 32 bits)

25.1.4 ETH_L4FilterConfigTypeDef

ETH_L4FilterConfigTypeDef is defined in the stm32h5xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcPortFilterMatch***
- ***uint32_t DestPortFilterMatch***
- ***uint32_t SourcePort***
- ***uint32_t DestinationPort***

Field Documentation

- ***uint32_t ETH_L4FilterConfigTypeDef::Protocol***
  Sets the L4 filter protocol to TCP or UDP This parameter can be a value of ***ETHEx_L4_Protocol***
- ***uint32_t ETH_L4FilterConfigTypeDef::SrcPortFilterMatch***
  Sets the L4 filter source port match This parameter can be a value of ***ETHEx_L4_Source_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::DestPortFilterMatch***
  Sets the L4 filter destination port match This parameter can be a value of ***ETHEx_L4_Destination_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::SourcePort***
  Sets the L4 filter source port This parameter must be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_L4FilterConfigTypeDef::DestinationPort***
  Sets the L4 filter destination port This parameter must be a value from 0x0 to 0xFFFF

25.2 ETHEx Firmware driver API description

The following section lists the various functions of the ETHEx library.

25.2.1 Extended features functions

This section provides functions allowing to:

- Configure ARP offload module
- Configure L3 and L4 filters
- Configure Extended VLAN features
- Configure Energy Efficient Ethernet module

This section contains the following APIs:

- *HAL_ETHEx_EnableARPOffload()*
- *HAL_ETHEx_DisableARPOffload()*
- *HAL_ETHEx_SetARPAddressMatch()*
- *HAL_ETHEx_SetL4FilterConfig()*
- *HAL_ETHEx_GetL4FilterConfig()*
- *HAL_ETHEx_SetL3FilterConfig()*
- *HAL_ETHEx_GetL3FilterConfig()*
- *HAL_ETHEx_EnableL3L4Filtering()*
- *HAL_ETHEx_DisableL3L4Filtering()*
- *HAL_ETHEx_GetRxVLANConfig()*
- *HAL_ETHEx_SetRxVLANConfig()*
- *HAL_ETHEx_SetVLANHashTable()*
- *HAL_ETHEx_GetTxVLANConfig()*
- *HAL_ETHEx_SetTxVLANConfig()*
- *HAL_ETHEx_SetTxVLANIdentifier()*
- *HAL_ETHEx_EnableVLANProcessing()*
- *HAL_ETHEx_DisableVLANProcessing()*
- *HAL_ETHEx_EnterLPIMode()*
- *HAL_ETHEx_ExitLPIMode()*
- *HAL_ETHEx_GetMACLPIEvent()*

25.2.2 Detailed description of functions

HAL_ETHEx_EnableARPOffload

Function name

void HAL_ETHEx_EnableARPOffload (ETH_HandleTypeDef * heth)

Function description

Enables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_DisableARPOffload

Function name

void HAL_ETHEx_DisableARPOffload (ETH_HandleTypeDef * heth)

Function description

Disables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_SetARPAddressMatch

Function name

void HAL_ETHEx_SetARPAddressMatch (ETH_HandleTypeDef * heth, uint32_t lpAddress)

Function description

Set the ARP Match IP address.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **lpAddress:** IP Address to be matched for incoming ARP requests

Return values

- **None:**

HAL_ETHEx_EnableL3L4Filtering

Function name

void HAL_ETHEx_EnableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

Enables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_DisableL3L4Filtering

Function name

void HAL_ETHEx_DisableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

Disables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_GetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/ disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0 ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that will contain the L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_GetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/ disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0 ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/ disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0 ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that contains L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/ disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0 ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_EnableVLANProcessing

Function name

void HAL_ETHEx_EnableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

Enables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_DisableVLANProcessing

Function name

void HAL_ETHEx_DisableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

Disables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_GetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetRxVLANConfig (ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

Get the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that will contain the VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetRxVLANConfig (ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

Set the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that contains VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetVLANHashTable

Function name

void HAL_ETHEx_SetVLANHashTable (ETH_HandleTypeDef * heth, uint32_t VLANHashTable)

Function description

Set the VLAN Hash Table.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANHashTable:** VLAN hash table 16 bit value

Return values

- **None:**

HAL_ETHEx_GetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag, ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

Get the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that will contain the Tx VLAN filter configuration.

Return values

- **HAL:** Status.

HAL_ETHEx_SetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag, ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

Set the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that contains Tx VLAN filter configuration.

Return values

- **HAL:** Status

HAL_ETHEx_SetTxVLANIdentifier

Function name

void HAL_ETHEx_SetTxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t VLANTag, uint32_t VLANIdentifier)

Function description

Set the VLAN Tag Identifier for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **VLANIdentifier:** VLAN Identifier 16 bit value

Return values

- **None:**

HAL_ETHEx_EnterLPIMode

Function name

void HAL_ETHEx_EnterLPIMode (ETH_HandleTypeDef * heth, FunctionalState TxAutomate, FunctionalState TxClockStop)

Function description

Enters the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **TxAutomate:** Enable/Disable automate enter/exit LPI mode.
- **TxClockStop:** Enable/Disable Tx clock stop in LPI mode.

Return values

- **None:**

HAL_ETHEx_ExitLPIMode

Function name

void HAL_ETHEx_ExitLPIMode (ETH_HandleTypeDef * heth)

Function description

Exits the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_GetMACLPIEvent

Function name

uint32_t HAL_ETHEx_GetMACLPIEvent (const ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC LPI event.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event

25.3 ETHEx Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 ETHEx

ETHEx

ETHEx L3 Destination Match

ETH_L3_DEST_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_DEST_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_DEST_ADDR_MATCH_DISABLE

ETHEx L3 Filter

ETH_L3_FILTER_0

ETH_L3_FILTER_1

ETHEx L3 Protocol

ETH_L3_IPV6_MATCH

ETH_L3_IPV4_MATCH

ETHEx L3 Source Match

ETH_L3_SRC_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_SRC_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_SRC_ADDR_MATCH_DISABLE

ETHEx L4 Destination Match

ETH_L4_DEST_PORT_PERFECT_MATCH_ENABLE

ETH_L4_DEST_PORT_INVERSE_MATCH_ENABLE

ETH_L4_DEST_PORT_MATCH_DISABLE

ETHEx L4 Filter

ETH_L4_FILTER_0

ETH_L4_FILTER_1

ETHEx L4 Protocol

ETH_L4_UDP_MATCH

ETH_L4_TCP_MATCH

ETHEx L4 Source Match

ETH_L4_SRC_PORT_PERFECT_MATCH_ENABLE

ETH_L4_SRC_PORT_INVERSE_MATCH_ENABLE

ETH_L4_SRC_PORT_MATCH_DISABLE

ETHEx LPI Event

ETH_TX_LPI_ENTRY

ETH_TX_LPI_EXIT

ETH_RX_LPI_ENTRY

ETH_RX_LPI_EXIT

ETHEx Rx Inner VLAN Tag Stripping

ETH_INNERVLANTAGRXSTRIPPING_NONE

ETH_INNERVLANTAGRXSTRIPPING_IFPASS

ETH_INNERVLANTAGRXSTRIPPING_IFFAILS

ETH_INNERVLANTAGRXSTRIPPING_ALWAYS

ETHEx Rx VLAN Tag Stripping

ETH_VLANTAGRXSTRIPPING_NONE

ETH_VLANTAGRXSTRIPPING_IFPASS

ETH_VLANTAGRXSTRIPPING_IFFAILS

ETH_VLANTAGRXSTRIPPING_ALWAYS

ETHEx Tx VLAN Tag

ETH_INNER_TX_VLANTAG

ETH_OUTER_TX_VLANTAG

ETHEx_VLAN_Tag_Control

ETH_VLANTAGCONTROL_NONE

ETH_VLANTAGCONTROL_DELETE

ETH_VLANTAGCONTROL_INSERT

ETH_VLANTAGCONTROL_REPLACE

ETHEx VLAN Type Check

ETH_VLANTYPECHECK_DISABLE

ETH_VLANTYPECHECK_SVLAN

ETH_VLANTYPECHECK_CVLAN

26 HAL EXTI Generic Driver

26.1 EXTI Firmware driver registers structures

26.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32h5xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***void(* RisingCallback***
- ***void(* FallingCallback***

Field Documentation

- ***uint32_t EXTI_HandleTypeDef::Line***
  Exti line number
- ***void(* EXTI_HandleTypeDef::RisingCallback)(void)***
  Exti rising callback
- ***void(* EXTI_HandleTypeDef::FallingCallback)(void)***
  Exti falling callback

26.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32h5xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***uint32_t Mode***
- ***uint32_t Trigger***
- ***uint32_t GPIOSel***

Field Documentation

- ***uint32_t EXTI_ConfigTypeDef::Line***
  The Exti line to be configured. This parameter can be a value of ***EXTI_Line***
- ***uint32_t EXTI_ConfigTypeDef::Mode***
  The Exit Mode to be configured for a core. This parameter can be a combination of ***EXTI_Mode***
- ***uint32_t EXTI_ConfigTypeDef::Trigger***
  The Exti Trigger to be configured. This parameter can be a value of ***EXTI_Trigger***
- ***uint32_t EXTI_ConfigTypeDef::GPIOSel***
  The Exti GPIO multiplexer selection to be configured. This parameter is only possible for line 0 to 15. It can be a value of ***EXTI_GPIOSel***

26.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

26.2.1 EXTI Peripheral features

- Each Exti line can be configured within this driver.
- Exti line can be configured in 3 different modes
  - Interrupt
  - Event
  - Both of them
- Configurable Exti lines can be configured with 3 different triggers
  - Rising
  - Falling
  - Both of them

- When set in interrupt mode, configurable Exti lines have two diffenrents interrupt pending registers which allow to distinguish which transition occurs:
  - Rising edge pending interrupt
  - Falling
- Exti lines 0 to 15 are linked to gpio pin number 0 to 15. Gpio port can be selected through multiplexer.

26.2.2 How to use this driver

1. Configure the EXTI line using HAL_EXTI_SetConfigLine().
   - Choose the interrupt line number by setting "Line" member from EXTI_ConfigTypeDef structure.
   - Configure the interrupt and/or event mode using "Mode" member from EXTI_ConfigTypeDef structure.
   - For configurable lines, configure rising and/or falling trigger "Trigger" member from EXTI_ConfigTypeDef structure.
   - For Exti lines linked to gpio, choose gpio port using "GPIOSel" member from GPIO_InitTypeDef structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
   - Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   - Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_GetPending().
7. Generate software interrupt using HAL_EXTI_GenerateSWI().

26.2.3 EXTI attributes functions

This section contains the following APIs:

- ***HAL_EXTI_ConfigLineAttributes()***
- ***HAL_EXTI_GetConfigLineAttributes()***
- ***HAL_EXTI_LockConfigAttributes()***
- ***HAL_EXTI_GetLockConfigAttributes()***

26.2.4 Configuration functions

This section contains the following APIs:

- ***HAL_EXTI_SetConfigLine()***
- ***HAL_EXTI_GetConfigLine()***
- ***HAL_EXTI_ClearConfigLine()***
- ***HAL_EXTI_RegisterCallback()***
- ***HAL_EXTI_GetHandle()***

26.2.5 Detailed description of functions

HAL_EXTI_SetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_SetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Set configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on EXTI configuration to be set.

Return values

- **HAL:** Status.

HAL_EXTI_GetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Get configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on structure to store Exti configuration.

Return values

- **HAL:** Status.

HAL_EXTI_ClearConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_ClearConfigLine (const EXTI_HandleTypeDef * hexti)

Function description

Clear whole configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.

Return values

- **HAL:** Status.

HAL_EXTI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_EXTI_RegisterCallback (EXTI_HandleTypeDef * hexti, EXTI_CallbackIDTypeDef CallbackID, void(*)(void) pPendingCbfn)

Function description

Register callback for a dedicaated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  - EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (const EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be checked. This parameter can be one of the following values:
  - EXTI_TRIGGER_RISING
  - EXTI_TRIGGER_FALLING

Return values

- **1:** if interrupt is pending else 0.

HAL_EXTI_ClearPending

Function name

void HAL_EXTI_ClearPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Clear interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be clear. This parameter can be one of the following values:
  - EXTI_TRIGGER_RISING
  - EXTI_TRIGGER_FALLING

Return values

- **None.:**

HAL_EXTI_GenerateSWI

Function name

void HAL_EXTI_GenerateSWI (const EXTI_HandleTypeDef * hexti)

Function description

Generate a software interrupt for a dedicated line.

Parameters

- **hexti:** Exti handle.

Return values

- **None.:**

HAL_EXTI_ConfigLineAttributes

Function name

void HAL_EXTI_ConfigLineAttributes (uint32_t ExtiLine, uint32_t LineAttributes)

Function description

Configure the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **LineAttributes:** can be one or a combination of the following values:
  - EXTI_LINE_PRIV Privileged-only access
  - EXTI_LINE_NPRIV Privileged/Non-privileged access
  - EXTI_LINE_SEC Secure-only access
  - EXTI_LINE_NSEC Secure/Non-secure access

Return values

- **None:**

Notes

- Available attributes are to secure EXTI line and set EXT line as privileged. Default state is not secure and unprivileged access allowed.
- Secure and non-secure attributes can only be set from the secure state when the system implements the security (TZEN=1).
- Security and privilege attributes can be set independently.

HAL_EXTI_GetConfigLineAttributes

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLineAttributes (uint32_t ExtiLine, uint32_t * pLineAttributes)

Function description

Get the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **pLineAttributes:** pointer to return line attributes.

Return values

- **HAL:** Status.

Notes

- Secure and non-secure attributes are only available from secure state when the system implements the security (TZEN=1)

HAL_EXTI_LockConfigAttributes

Function name

HAL_StatusTypeDef HAL_EXTI_LockConfigAttributes (void)

Function description

Lock the global EXTI security and privilege configuration.

Return values

- **HAL:** Status.

HAL_EXTI_GetLockConfigAttributes

Function name

HAL_StatusTypeDef HAL_EXTI_GetLockConfigAttributes (uint32_t *const pLockState)

Function description

Get the global EXTI security and privilege lock configuration.

Parameters

- **pLockState:** : Pointer to returned security and privilege configuration

Return values

- **HAL:** Status.

26.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 EXTI

EXTI

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOE

EXTI_GPIOF

EXTI_GPIOG

EXTI_GPIOH

EXTI_GPIOI

EXTI Line

EXTI_LINE_0

EXTI_LINE_1

EXTI_LINE_2

EXTI_LINE_3

EXTI_LINE_4

EXTI_LINE_5

EXTI_LINE_6

EXTI_LINE_7

EXTI_LINE_8

EXTI_LINE_9

EXTI_LINE_10

EXTI_LINE_11

EXTI_LINE_12

EXTI_LINE_13

EXTI_LINE_14

EXTI_LINE_15

EXTI_LINE_16

EXTI_LINE_17

EXTI_LINE_18

EXTI_LINE_19

EXTI_LINE_20

EXTI_LINE_21

EXTI_LINE_22

EXTI_LINE_23

EXTI_LINE_24

EXTI_LINE_25

EXTI_LINE_26

EXTI_LINE_27

EXTI_LINE_28

EXTI_LINE_29

EXTI_LINE_30

EXTI_LINE_31

EXTI_LINE_32

EXTI_LINE_33

EXTI_LINE_34

EXTI_LINE_35

EXTI_LINE_36

EXTI_LINE_37

EXTI_LINE_38

EXTI_LINE_39

EXTI_LINE_40

EXTI_LINE_41

EXTI_LINE_42

EXTI_LINE_43

EXTI_LINE_44

EXTI_LINE_45

EXTI_LINE_46

EXTI_LINE_47

EXTI_LINE_48

EXTI_LINE_49

EXTI_LINE_50

EXTI_LINE_51

EXTI_LINE_52

EXTI_LINE_53

EXTI_LINE_54

EXTI_LINE_55

EXTI_LINE_56

EXTI_LINE_57

EXTI line attributes

EXTI_LINE_SEC

< Secure line attribute Non-secure line attribute

EXTI_LINE_NSEC

EXTI_LINE_PRIV

< Privileged line attribute Non-privileged line attribute

EXTI_LINE_NPRIV

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI Security Privilege Configuration

EXTI_ATTRIBUTES_UNLOCKED

EXTI_ATTRIBUTES_LOCKED

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

27 HAL FDCAN Generic Driver

27.1 FDCAN Firmware driver registers structures

27.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t ClockDivider***
- ***uint32_t FrameFormat***
- ***uint32_t Mode***
- ***FunctionalState AutoRetransmission***
- ***FunctionalState TransmitPause***
- ***FunctionalState ProtocolException***
- ***uint32_t NominalPrescaler***
- ***uint32_t NominalSyncJumpWidth***
- ***uint32_t NominalTimeSeg1***
- ***uint32_t NominalTimeSeg2***
- ***uint32_t DataPrescaler***
- ***uint32_t DataSyncJumpWidth***
- ***uint32_t DataTimeSeg1***
- ***uint32_t DataTimeSeg2***
- ***uint32_t StdFiltersNbr***
- ***uint32_t ExtFiltersNbr***
- ***uint32_t TxFifoQueueMode***

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::ClockDivider***
  Specifies the FDCAN kernel clock divider. The clock is common to all FDCAN instances. This parameter is applied only at initialisation of first FDCAN instance. This parameter can be a value of ***FDCAN_clock_divider***.
- ***uint32_t FDCAN_InitTypeDef::FrameFormat***
  Specifies the FDCAN frame format. This parameter can be a value of ***FDCAN_frame_format***
- ***uint32_t FDCAN_InitTypeDef::Mode***
  Specifies the FDCAN mode. This parameter can be a value of ***FDCAN_operating_mode***
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***
  Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***
  Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***
  Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE
- ***uint32_t FDCAN_InitTypeDef::NominalPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512
- ***uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg1***
  Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg2***
  Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128

- ***uint32_t FDCAN_InitTypeDef::DataPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg1***
  Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg2***
  Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::StdFiltersNbr***
  Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 28
- ***uint32_t FDCAN_InitTypeDef::ExtFiltersNbr***
  Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 8
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueMode***
  Tx FIFO/Queue Mode selection. This parameter can be a value of ***FDCAN_txFifoQueue_Mode***

27.1.2 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t IdType***
- ***uint32_t FilterIndex***
- ***uint32_t FilterType***
- ***uint32_t FilterConfig***
- ***uint32_t FilterID1***
- ***uint32_t FilterID2***

Field Documentation

- ***uint32_t FDCAN_FilterTypeDef::IdType***
  Specifies the identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_FilterTypeDef::FilterIndex***
  Specifies the filter which will be initialized. This parameter must be a number between:
  - 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  - 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterType***
  Specifies the filter type. This parameter can be a value of ***FDCAN_filter_type***. The value FDCAN_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID.
- ***uint32_t FDCAN_FilterTypeDef::FilterConfig***
  Specifies the filter configuration. This parameter can be a value of ***FDCAN_filter_config***
- ***uint32_t FDCAN_FilterTypeDef::FilterID1***
  Specifies the filter identification 1. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterID2***
  Specifies the filter identification 2. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID

27.1.3 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***

- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxEventFifoControl***
- ***uint32_t MessageMarker***

Field Documentation

- ***uint32_t FDCAN_TxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxHeaderTypeDef::IdType***
  Specifies the identifier type for the message that will be transmitted. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxFrameType***
  Specifies the frame type of the message that will be transmitted. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::DataLength***
  Specifies the length of the frame that will be transmitted. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxHeaderTypeDef::FDFormat***
  Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl***
  Specifies the event FIFO control. This parameter can be a value of ***Section 27.3.1 FDCAN_EFC***
- ***uint32_t FDCAN_TxHeaderTypeDef::MessageMarker***
  Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

27.1.4 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t RxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t RxTimestamp***
- ***uint32_t FilterIndex***
- ***uint32_t IsFilterMatchingFrame***

Field Documentation

- ***uint32_t FDCAN_RxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IdType***
  Specifies the identifier type of the received message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxFrameType***
  Specifies the the received message frame type. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::DataLength***
  Specifies the received frame length. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_RxHeaderTypeDef::FDFormat***
  Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp***
  Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_RxHeaderTypeDef::FilterIndex***
  Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
  - 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  - 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID When the frame is a Non-Filter matching frame, this parameter is unused.
- ***uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame***
  Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter takes 0 if the frame matched an Rx filter or 1 if it did not match any Rx filter

27.1.5 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxTimestamp***
- ***uint32_t MessageMarker***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t FDCAN_TxEventFifoTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxEventFifoTypeDef::IdType***
  Specifies the identifier type for the transmitted message. This parameter can be a value of ***FDCAN_id_type***

- ***uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType***
  Specifies the frame type of the transmitted message. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::DataLength***
  Specifies the length of the transmitted frame. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch***
  Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxEventFifoTypeDef::FDFormat***
  Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp***
  Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker***
  Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::EventType***
  Specifies the event type. This parameter can be a value of ***FDCAN_event_type***

27.1.6 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t FilterList***
- ***uint32_t FilterIndex***
- ***uint32_t MessageStorage***
- ***uint32_t MessageIndex***

Field Documentation

- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterList***
  Specifies the filter list of the matching filter element. This parameter can be:
  - 0 : Standard Filter List
  - 1 : Extended Filter List
- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex***
  Specifies the index of matching filter element. This parameter can be a number between:
  - 0 and (SRAMCAN_FLS_NBR-1), if FilterList is 0 (Standard)
  - 0 and (SRAMCAN_FLE_NBR-1), if FilterList is 1 (Extended)
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage***
  Specifies the HP Message Storage. This parameter can be a value of ***FDCAN_hp_msg_storage***
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex***
  Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

27.1.7 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t LastErrorCode***
- ***uint32_t DataLastErrorCode***
- ***uint32_t Activity***
- ***uint32_t ErrorPassive***
- ***uint32_t Warning***
- ***uint32_t BusOff***

- ***uint32_t RxESIflag***
- ***uint32_t RxBRSflag***
- ***uint32_t RxFDFflag***
- ***uint32_t ProtocolException***
- ***uint32_t TDCvalue***

Field Documentation

- ***uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode***
  Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode***
  Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Activity***
  Specifies the FDCAN module communication state. This parameter can be a value of ***FDCAN_communication_state***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive***
  Specifies the FDCAN module error status. This parameter can be:
  - 0 : The FDCAN is in Error_Active state
  - 1 : The FDCAN is in Error_Passive state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Warning***
  Specifies the FDCAN module warning status. This parameter can be:
  - 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
  - 1 : at least one of error counters has reached the Error_Warning limit of 96
- ***uint32_t FDCAN_ProtocolStatusTypeDef::BusOff***
  Specifies the FDCAN module Bus_Off status. This parameter can be:
  - 0 : The FDCAN is not in Bus_Off state
  - 1 : The FDCAN is in Bus_Off state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag***
  Specifies ESI flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its ESI flag set
  - 1 : Last received CAN FD message had its ESI flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag***
  Specifies BRS flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its BRS flag set
  - 1 : Last received CAN FD message had its BRS flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag***
  Specifies if CAN FD message (FDF flag set) has been received since last protocol status. This parameter can be:
  - 0 : No CAN FD message received
  - 1 : CAN FD message received
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***
  Specifies the FDCAN module Protocol Exception status. This parameter can be:
  - 0 : No protocol exception event occurred since last read access
  - 1 : Protocol exception event occurred
- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***
  Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

27.1.8 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***

- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***
  Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***
  Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***
  Specifies the Receive Error Passive status. This parameter can be:
  – 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
  – 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128
- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***
  Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 255. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 255; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

27.1.9 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***uint32_t StandardFilterSA***
- ***uint32_t ExtendedFilterSA***
- ***uint32_t RxFIFO0SA***
- ***uint32_t RxFIFO1SA***
- ***uint32_t TxEventFIFOSA***
- ***uint32_t TxFIFOQSA***

Field Documentation

- ***uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA***
  Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA***
  Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA***
  Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA***
  Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA***
  Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA***
  Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address

27.1.10 __FDCAN_HandleTypeDef

__FDCAN_HandleTypeDef is defined in the stm32h5xx_hal_fdcan.h

Data Fields

- ***FDCAN_GlobalTypeDef * Instance***
- ***FDCAN_InitTypeDef Init***
- ***FDCAN_MsgRamAddressTypeDef msgRam***
- ***uint32_t LatestTxFifoQRequest***
- ***__IO HAL_FDCAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* TxEventFifoCallback***
- ***void(* RxFifo0Callback***
- ***void(* RxFifo1Callback***

- ***void(* TxFifoEmptyCallback***
- ***void(* TxBufferCompleteCallback***
- ***void(* TxBufferAbortCallback***
- ***void(* HighPriorityMessageCallback***
- ***void(* TimestampWraparoundCallback***
- ***void(* TimeoutOccurredCallback***
- ***void(* ErrorCallback***
- ***void(* ErrorStatusCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FDCAN_GlobalTypeDef* __FDCAN_HandleTypeDef::Instance***
  Register base address
- ***FDCAN_InitTypeDef __FDCAN_HandleTypeDef::Init***
  FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef __FDCAN_HandleTypeDef::msgRam***
  FDCAN Message RAM blocks
- ***uint32_t __FDCAN_HandleTypeDef::LatestTxFifoQRequest***
  FDCAN Tx buffer index of latest Tx FIFO/Queue request
- ***__IO HAL_FDCAN_StateTypeDef __FDCAN_HandleTypeDef::State***
  FDCAN communication state
- ***HAL_LockTypeDef __FDCAN_HandleTypeDef::Lock***
  FDCAN locking object
- ***__IO uint32_t __FDCAN_HandleTypeDef::ErrorCode***
  FDCAN Error code
- ***void(* __FDCAN_HandleTypeDef::TxEventFifoCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TxEventFifoITs)***
  FDCAN Tx Event Fifo callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo0Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)***
  FDCAN Rx Fifo 0 callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo1Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo1ITs)***
  FDCAN Rx Fifo 1 callback
- ***void(* __FDCAN_HandleTypeDef::TxFifoEmptyCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Tx Fifo Empty callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferCompleteCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer complete callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferAbortCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer abort callback
- ***void(* __FDCAN_HandleTypeDef::HighPriorityMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN High priority message callback
- ***void(* __FDCAN_HandleTypeDef::TimestampWraparoundCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timestamp wraparound callback
- ***void(* __FDCAN_HandleTypeDef::TimeoutOccurredCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timeout occurred callback
- ***void(* __FDCAN_HandleTypeDef::ErrorCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Error callback

- ***void(* __FDCAN_HandleTypeDef::ErrorStatusCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ErrorStatusITs)***
  FDCAN Error status callback
- ***void(* __FDCAN_HandleTypeDef::MspInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp Init callback
- ***void(* __FDCAN_HandleTypeDef::MspDeInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp DeInit callback

27.2 FDCAN Firmware driver API description

The following section lists the various functions of the FDCAN library.

27.2.1 How to use this driver

1. Initialize the FDCAN peripheral using HAL_FDCAN_Init function.
2. If needed , configure the reception filters and optional features using the following configuration functions:
   – HAL_FDCAN_ConfigFilter
   – HAL_FDCAN_ConfigGlobalFilter
   – HAL_FDCAN_ConfigExtendedIdMask
   – HAL_FDCAN_ConfigRxFifoOverwrite
   – HAL_FDCAN_ConfigRamWatchdog
   – HAL_FDCAN_ConfigTimestampCounter
   – HAL_FDCAN_EnableTimestampCounter
   – HAL_FDCAN_DisableTimestampCounter
   – HAL_FDCAN_ConfigTimeoutCounter
   – HAL_FDCAN_EnableTimeoutCounter
   – HAL_FDCAN_DisableTimeoutCounter
   – HAL_FDCAN_ConfigTxDelayCompensation
   – HAL_FDCAN_EnableTxDelayCompensation
   – HAL_FDCAN_DisableTxDelayCompensation
   – HAL_FDCAN_EnableISOMode
   – HAL_FDCAN_DisableISOMode
   – HAL_FDCAN_EnableEdgeFiltering
   – HAL_FDCAN_DisableEdgeFiltering
3. Start the FDCAN module using HAL_FDCAN_Start function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
   – HAL_FDCAN_AddMessageToTxFifoQ
   – HAL_FDCAN_AbortTxRequest
5. After having submitted a Tx request in Tx Fifo or Queue, it is possible to get Tx buffer location used to place the Tx request thanks to HAL_FDCAN_GetLatestTxFifoQRequestBuffer API. It is then possible to abort later on the corresponding Tx Request using HAL_FDCAN_AbortTxRequest API.
6. When a message is received into the FDCAN message RAM, it can be retrieved using the HAL_FDCAN_GetRxMessage function.
7. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
8. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
   – HAL_FDCAN_IsTxBufferMessagePending
   – HAL_FDCAN_GetRxFifoFillLevel
   – HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.
2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

Callback registration

27.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the FDCAN.
- De-initialize the FDCAN.
- Enter FDCAN peripheral in power down mode.
- Exit power down mode.
- Register callbacks.
- Unregister callbacks.

This section contains the following APIs:

- ***HAL_FDCAN_Init()***
- ***HAL_FDCAN_DeInit()***
- ***HAL_FDCAN_MspInit()***
- ***HAL_FDCAN_MspDeInit()***
- ***HAL_FDCAN_EnterPowerDownMode()***
- ***HAL_FDCAN_ExitPowerDownMode()***
- ***HAL_FDCAN_RegisterCallback()***
- ***HAL_FDCAN_UnRegisterCallback()***
- ***HAL_FDCAN_RegisterTxEventFifoCallback()***
- ***HAL_FDCAN_UnRegisterTxEventFifoCallback()***
- ***HAL_FDCAN_RegisterRxFifo0Callback()***
- ***HAL_FDCAN_UnRegisterRxFifo0Callback()***
- ***HAL_FDCAN_RegisterRxFifo1Callback()***
- ***HAL_FDCAN_UnRegisterRxFifo1Callback()***
- ***HAL_FDCAN_RegisterTxBufferCompleteCallback()***
- ***HAL_FDCAN_UnRegisterTxBufferCompleteCallback()***
- ***HAL_FDCAN_RegisterTxBufferAbortCallback()***
- ***HAL_FDCAN_UnRegisterTxBufferAbortCallback()***
- ***HAL_FDCAN_RegisterErrorStatusCallback()***
- ***HAL_FDCAN_UnRegisterErrorStatusCallback()***
- ***HAL_FDCAN_RegisterCallback()***

27.2.3 Configuration functions

This section provides functions allowing to:

- HAL_FDCAN_ConfigFilter : Configure the FDCAN reception filters
- HAL_FDCAN_ConfigGlobalFilter : Configure the FDCAN global filter
- HAL_FDCAN_ConfigExtendedIdMask : Configure the extended ID mask
- HAL_FDCAN_ConfigRxFifoOverwrite : Configure the Rx FIFO operation mode
- HAL_FDCAN_ConfigRamWatchdog : Configure the RAM watchdog
- HAL_FDCAN_ConfigTimestampCounter : Configure the timestamp counter
- HAL_FDCAN_EnableTimestampCounter : Enable the timestamp counter
- HAL_FDCAN_DisableTimestampCounter : Disable the timestamp counter

- HAL_FDCAN_GetTimestampCounter : Get the timestamp counter value
- HAL_FDCAN_ResetTimestampCounter : Reset the timestamp counter to zero
- HAL_FDCAN_ConfigTimeoutCounter : Configure the timeout counter
- HAL_FDCAN_EnableTimeoutCounter : Enable the timeout counter
- HAL_FDCAN_DisableTimeoutCounter : Disable the timeout counter
- HAL_FDCAN_GetTimeoutCounter : Get the timeout counter value
- HAL_FDCAN_ResetTimeoutCounter : Reset the timeout counter to its start value
- HAL_FDCAN_ConfigTxDelayCompensation : Configure the transmitter delay compensation
- HAL_FDCAN_EnableTxDelayCompensation : Enable the transmitter delay compensation
- HAL_FDCAN_DisableTxDelayCompensation : Disable the transmitter delay compensation
- HAL_FDCAN_EnableISOMode : Enable ISO 11898-1 protocol mode
- HAL_FDCAN_DisableISOMode : Disable ISO 11898-1 protocol mode
- HAL_FDCAN_EnableEdgeFiltering : Enable edge filtering during bus integration
- HAL_FDCAN_DisableEdgeFiltering : Disable edge filtering during bus integration

This section contains the following APIs:

- ***HAL_FDCAN_ConfigFilter()***
- ***HAL_FDCAN_ConfigGlobalFilter()***
- ***HAL_FDCAN_ConfigExtendedIdMask()***
- ***HAL_FDCAN_ConfigRxFifoOverwrite()***
- ***HAL_FDCAN_ConfigRamWatchdog()***
- ***HAL_FDCAN_ConfigTimestampCounter()***
- ***HAL_FDCAN_EnableTimestampCounter()***
- ***HAL_FDCAN_DisableTimestampCounter()***
- ***HAL_FDCAN_GetTimestampCounter()***
- ***HAL_FDCAN_ResetTimestampCounter()***
- ***HAL_FDCAN_ConfigTimeoutCounter()***
- ***HAL_FDCAN_EnableTimeoutCounter()***
- ***HAL_FDCAN_DisableTimeoutCounter()***
- ***HAL_FDCAN_GetTimeoutCounter()***
- ***HAL_FDCAN_ResetTimeoutCounter()***
- ***HAL_FDCAN_ConfigTxDelayCompensation()***
- ***HAL_FDCAN_EnableTxDelayCompensation()***
- ***HAL_FDCAN_DisableTxDelayCompensation()***
- ***HAL_FDCAN_EnableISOMode()***
- ***HAL_FDCAN_DisableISOMode()***
- ***HAL_FDCAN_EnableEdgeFiltering()***
- ***HAL_FDCAN_DisableEdgeFiltering()***

27.2.4 Control functions

This section provides functions allowing to:

- HAL_FDCAN_Start : Start the FDCAN module
- HAL_FDCAN_Stop : Stop the FDCAN module and enable access to configuration registers
- HAL_FDCAN_AddMessageToTxFifoQ : Add a message to the Tx FIFO/Queue and activate the corresponding transmission request
- HAL_FDCAN_GetLatestTxFifoQRequestBuffer : Get Tx buffer index of latest Tx FIFO/Queue request
- HAL_FDCAN_AbortTxRequest : Abort transmission request
- HAL_FDCAN_GetRxMessage : Get an FDCAN frame from the Rx FIFO zone into the message RAM
- HAL_FDCAN_GetTxEvent : Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM
- HAL_FDCAN_GetHighPriorityMessageStatus : Get high priority message status
- HAL_FDCAN_GetProtocolStatus : Get protocol status

- HAL_FDCAN_GetErrorCounters : Get error counter values
- HAL_FDCAN_IsTxBufferMessagePending : Check if a transmission request is pending on the selected Tx buffer
- HAL_FDCAN_GetRxFifoFillLevel : Return Rx FIFO fill level
- HAL_FDCAN_GetTxFifoFreeLevel : Return Tx FIFO free level
- HAL_FDCAN_IsRestrictedOperationMode : Check if the FDCAN peripheral entered Restricted Operation Mode
- HAL_FDCAN_ExitRestrictedOperationMode : Exit Restricted Operation Mode

This section contains the following APIs:

- ***HAL_FDCAN_Start()***
- ***HAL_FDCAN_Stop()***
- ***HAL_FDCAN_AddMessageToTxFifoQ()***
- ***HAL_FDCAN_GetLatestTxFifoQRequestBuffer()***
- ***HAL_FDCAN_AbortTxRequest()***
- ***HAL_FDCAN_GetRxMessage()***
- ***HAL_FDCAN_GetTxEvent()***
- ***HAL_FDCAN_GetHighPriorityMessageStatus()***
- ***HAL_FDCAN_GetProtocolStatus()***
- ***HAL_FDCAN_GetErrorCounters()***
- ***HAL_FDCAN_IsTxBufferMessagePending()***
- ***HAL_FDCAN_GetRxFifoFillLevel()***
- ***HAL_FDCAN_GetTxFifoFreeLevel()***
- ***HAL_FDCAN_IsRestrictedOperationMode()***
- ***HAL_FDCAN_ExitRestrictedOperationMode()***

27.2.5 Interrupts management

This section provides functions allowing to:

- HAL_FDCAN_ConfigInterruptLines : Assign interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_ActivateNotification : Enable interrupts
- HAL_FDCAN_DeactivateNotification : Disable interrupts
- HAL_FDCAN_IRQHandler : Handles FDCAN interrupt request

This section contains the following APIs:

- ***HAL_FDCAN_ConfigInterruptLines()***
- ***HAL_FDCAN_ActivateNotification()***
- ***HAL_FDCAN_DeactivateNotification()***
- ***HAL_FDCAN_IRQHandler()***

27.2.6 Callback functions

This subsection provides the following callback functions:

- HAL_FDCAN_TxEventFifoCallback
- HAL_FDCAN_RxFifo0Callback
- HAL_FDCAN_RxFifo1Callback
- HAL_FDCAN_TxFifoEmptyCallback
- HAL_FDCAN_TxBufferCompleteCallback
- HAL_FDCAN_TxBufferAbortCallback
- HAL_FDCAN_HighPriorityMessageCallback
- HAL_FDCAN_TimestampWraparoundCallback
- HAL_FDCAN_TimeoutOccurredCallback
- HAL_FDCAN_ErrorCallback
- HAL_FDCAN_ErrorStatusCallback

This section contains the following APIs:

- *HAL_FDCAN_TxEventFifoCallback()*
- *HAL_FDCAN_RxFifo0Callback()*
- *HAL_FDCAN_RxFifo1Callback()*
- *HAL_FDCAN_TxFifoEmptyCallback()*
- *HAL_FDCAN_TxBufferCompleteCallback()*
- *HAL_FDCAN_TxBufferAbortCallback()*
- *HAL_FDCAN_TimestampWraparoundCallback()*
- *HAL_FDCAN_TimeoutOccurredCallback()*
- *HAL_FDCAN_HighPriorityMessageCallback()*
- *HAL_FDCAN_ErrorCallback()*
- *HAL_FDCAN_ErrorStatusCallback()*

27.2.7 Peripheral State functions

This subsection provides functions allowing to :

- HAL_FDCAN_GetState() : Return the FDCAN state.
- HAL_FDCAN_GetError() : Return the FDCAN error code if any.

This section contains the following APIs:

- *HAL_FDCAN_GetState()*
- *HAL_FDCAN_GetError()*

27.2.8 Detailed description of functions

HAL_FDCAN_Init

Function name

HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN peripheral according to the specified parameters in the FDCAN_InitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN peripheral registers to their default reset values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

DeInitializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enter FDCAN peripheral in sleep mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit power down mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID, pFDCAN_CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID)

Function description

Unregister a FDCAN CallBack.

Parameters

- **hfdcan:** pointer to a FDCAN_HandleTypeDef structure that contains the configuration information for FDCAN module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_FDCAN_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty callback ID
  - HAL_FDCAN_HIGH_PRIO_MESSAGE_CB_ID High priority message callback ID
  - HAL_FDCAN_TIMESTAMP_WRAPAROUND_CB_ID Timestamp wraparound callback ID
  - HAL_FDCAN_TIMEOUT_OCCURRED_CB_ID Timeout occurred callback ID
  - HAL_FDCAN_ERROR_CALLBACK_CB_ID Error callback ID
  - HAL_FDCAN_MSPINIT_CB_ID MspInit callback ID
  - HAL_FDCAN_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxEventFifoCallbackTypeDef pCallback)

Function description

Register Tx Event Fifo FDCAN Callback To be used instead of the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Event Fifo Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Event Fifo FDCAN Callback Tx Event Fifo FDCAN Callback is redirected to the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo0CallbackTypeDef pCallback)

Function description

Register Rx Fifo 0 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 0 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 0 FDCAN Callback Rx Fifo 0 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo1CallbackTypeDef pCallback)

Function description

Register Rx Fifo 1 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 1 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 1 FDCAN Callback Rx Fifo 1 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferCompleteCallbackTypeDef pCallback)

Function description

Register Tx Buffer Complete FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Complete Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Complete FDCAN Callback Tx Buffer Complete FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferAbortCallbackTypeDef pCallback)

Function description

Register Tx Buffer Abort FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Abort Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Abort FDCAN Callback Tx Buffer Abort FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ErrorStatusCallbackTypeDef pCallback)

Function description

Register Error Status FDCAN Callback To be used instead of the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Error Status Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Error Status FDCAN Callback Error Status FDCAN Callback is redirected to the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDef * hfdcan, const FDCAN_FilterTypeDef * sFilterConfig)

Function description

Configure the FDCAN reception filter according to the specified parameters in the FDCAN_FilterTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sFilterConfig:** pointer to an FDCAN_FilterTypeDef structure that contains the filter configuration information

Return values

- **HAL:** status

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

Configure the FDCAN global filter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NonMatchingStd:** Defines how received messages with 11-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **NonMatchingExt:** Defines how received messages with 29-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **RejectRemoteStd:** Filter or reject all the remote 11-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.
- **RejectRemoteExt:** Filter or reject all the remote 29-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.

Return values

- **HAL:** status

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

Configure the extended ID mask.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Mask:** Extended ID Mask. This parameter must be a number between 0 and 0x1FFFFFFF.

Return values

- **HAL:** status

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

Configure the Rx FIFO operation mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1
- **OperationMode:** operation mode. This parameter can be a value of
  - FDCAN_Rx_FIFO_operation_mode.

Return values

- **HAL:** status

HAL_FDCAN_ConfigRamWatchdog

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t CounterStartValue)

Function description

Configure the RAM watchdog.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **CounterStartValue:** Start value of the Message RAM Watchdog Counter, This parameter must be a number between 0x00 and 0xFF, with the reset value of 0x00 the counter is disabled.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampPrescaler)

Function description

Configure the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampPrescaler:** Timestamp Counter Prescaler. This parameter can be a value of
  - FDCAN_Timestamp_Prescaler.

Return values

- **HAL:** status

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampOperation)

Function description

Enable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampOperation:** Timestamp counter operation. This parameter can be a value of
  - FDCAN_Timestamp.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timestamp counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timestamp:** counter value

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timestamp counter to zero.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

Configure the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeoutOperation:** Timeout counter operation. This parameter can be a value of
  - FDCAN_Timeout_Operation.
- **TimeoutPeriod:** Start value of the timeout down-counter. This parameter must be a number between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimeoutCounter

Function name

uint16_t HAL_FDCAN_GetTimeoutCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timeout counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timeout:** counter value

HAL_FDCAN_ResetTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timeout counter to its start value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan, uint32_t TdcOffset, uint32_t TdcFilter)

Function description

Configure the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TdcOffset:** Transmitter Delay Compensation Offset. This parameter must be a number between 0x00 and 0x7F.
- **TdcFilter:** Transmitter Delay Compensation Filter Window Length. This parameter must be a number between 0x00 and 0x7F.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

Start the FDCAN module.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

Stop the FDCAN module and enable access to configuration registers.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxFifoQ

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TxHeaderTypeDef * pTxHeader, const uint8_t * pTxData)

Function description

Add a message to the Tx FIFO/Queue and activate the corresponding transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.

Return values

- **HAL:** status

HAL_FDCAN_GetLatestTxFifoQRequestBuffer

Function name

uint32_t HAL_FDCAN_GetLatestTxFifoQRequestBuffer (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get Tx buffer index of latest Tx FIFO/Queue request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** buffer index of last Tx FIFO/Queue request
  - Any value of
    - FDCAN_Tx_location if Tx request has been submitted.
  - 0 if no Tx FIFO/Queue request have been submitted.

HAL_FDCAN_AbortTxRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

Abort transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetRxMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)

Function description

Get an FDCAN frame from the Rx FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxLocation:** Location of the received message to be read. This parameter can be a value of
  - FDCAN_Rx_location.
- **pRxHeader:** pointer to a FDCAN_RxHeaderTypeDef structure.
- **pRxData:** pointer to a buffer where the payload of the Rx frame will be stored.

Return values

- **HAL:** status

HAL_FDCAN_GetTxEvent

Function name

HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxEventFifoTypeDef * pTxEvent)

Function description

Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxEvent:** pointer to a FDCAN_TxEventFifoTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_HpMsgStatusTypeDef * HpMsgStatus)

Function description

Get high priority message status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **HpMsgStatus:** pointer to an FDCAN_HpMsgStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetProtocolStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ProtocolStatusTypeDef * ProtocolStatus)

Function description

Get protocol status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ProtocolStatus:** pointer to an FDCAN_ProtocolStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetErrorCounters

Function name

HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ErrorCountersTypeDef * ErrorCounters)

Function description

Get error counter values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorCounters:** pointer to an FDCAN_ErrorCountersTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_IsTxBufferMessagePending

Function name

uint32_t HAL_FDCAN_IsTxBufferMessagePending (const FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)

Function description

Check if a transmission request is pending on the selected Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxBufferIndex:** Tx buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **Status:**
  - 0 : No pending transmission request on TxBufferIndex list.
  - 1 : Pending transmission request on TxBufferIndex.

HAL_FDCAN_GetRxFifoFillLevel

Function name

uint32_t HAL_FDCAN_GetRxFifoFillLevel (const FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)

Function description

Return Rx FIFO fill level.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1

Return values

- **Rx:** FIFO fill level.

HAL_FDCAN_GetTxFifoFreeLevel

Function name

uint32_t HAL_FDCAN_GetTxFifoFreeLevel (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return Tx FIFO free level: number of consecutive free Tx FIFO elements starting from Tx FIFO GetIndex.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** FIFO free level.

HAL_FDCAN_IsRestrictedOperationMode

Function name

uint32_t HAL_FDCAN_IsRestrictedOperationMode (const FDCAN_HandleTypeDef * hfdcan)

Function description

Check if the FDCAN peripheral entered Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Status:**
  - 0 : Normal FDCAN operation.
  - 1 : Restricted Operation Mode active.

HAL_FDCAN_ExitRestrictedOperationMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t ITList, uint32_t InterruptLine)

Function description

Assign interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ITList:** indicates which interrupts group will be assigned to the selected interrupt line. This parameter can be any combination of
  - FDCAN_Interrupts_Group.
- **InterruptLine:** Interrupt line. This parameter can be a value of
  - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)

Function description

Enable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveITs:** indicates which interrupts will be enabled. This parameter can be any combination of
  - FDCAN_Interrupts.
- **BufferIndexes:** Tx Buffer Indexes. This parameter can be any combination of
  - FDCAN_Tx_location. This parameter is ignored if ActiveITs does not include one of the following:
    - FDCAN_IT_TX_COMPLETE
    - FDCAN_IT_TX_ABORT_COMPLETE

Return values

- **HAL:** status

HAL_FDCAN_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)

Function description

Disable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveITs:** indicates which interrupts will be disabled. This parameter can be any combination of
  - FDCAN_Interrupts.

Return values

- **HAL:** status

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

Handles FDCAN interrupt request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

Tx Event callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxEventFifoITs:** indicates which Tx Event FIFO interrupts are signaled. This parameter can be any combination of
  - FDCAN_Tx_Event_Fifo_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

Rx FIFO 0 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo0ITs:** indicates which Rx FIFO 0 interrupts are signaled. This parameter can be any combination of
  - FDCAN_Rx_Fifo0_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

Rx FIFO 1 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo1ITs:** indicates which Rx FIFO 1 interrupts are signaled. This parameter can be any combination of
  - FDCAN_Rx_Fifo1_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxFifoEmptyCallback

Function name

void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Tx FIFO Empty callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TxBufferCompleteCallback

Function name

void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Complete callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the transmitted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_TxBufferAbortCallback

Function name

void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Cancellation callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the aborted buffers. This parameter can be any combination of
  – FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

High Priority Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timestamp Wraparound callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timeout Occurred callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Error callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorStatusCallback

Function name

void HAL_FDCAN_ErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ErrorStatusITs)

Function description

Error status callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorStatusITs:** indicates which Error Status interrupts are signaled. This parameter can be any combination of
  - FDCAN_Error_Status_Interrupts.

Return values

- **None:**

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN error code.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **FDCAN:** Error Code

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** state

27.3 FDCAN Firmware driver defines

The following section lists the various define and macros of the module.

27.3.1 FDCAN

FDCAN

FDCAN Bit Rate Switching

FDCAN_BRS_OFF

FDCAN frames transmitted/received without bit rate switching

FDCAN_BRS_ON

FDCAN frames transmitted/received with bit rate switching

FDCAN Clock Divider

FDCAN_CLOCK_DIV1

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state

FDCAN_COM_STATE_SYNC

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts

FDCAN_IT_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Length Code

FDCAN_DLC_BYTES_0

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts

FDCAN_IT_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Error Status Interrupts

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

__HAL_FDCAN_RESET_HANDLE_STATE

Description:

- Reset FDCAN handle state.

Parameters:

- __HANDLE__: FDCAN handle.

Return value:

- None

__HAL_FDCAN_ENABLE_IT

Description:

- Enable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_DISABLE_IT

Description:

- Disable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_IT

Description:

- Check whether the specified FDCAN interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be one of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_CLEAR_IT

Description:

- Clear the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the interrupts to clear. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_FLAG

Description:

- Check whether the specified FDCAN flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN flag. This parameter can be one of
  - FDCAN_flags

Return value:

- FlagStatus

__HAL_FDCAN_CLEAR_FLAG

Description:

- Clear the specified FDCAN flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the flags to clear. This parameter can be any combination of
  - FDCAN_flags

Return value:

- None

__HAL_FDCAN_GET_IT_SOURCE

Description:

- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN interrupt source to check. This parameter can be a value of
  - FDCAN_Interrupts

Return value:

- ITStatus

FDCAN Filter Configuration

FDCAN_FILTER_DISABLE

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN Filter Type

FDCAN_FILTER_RANGE

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags

FDCAN_FLAG_TX_COMPLETE

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE

Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING

Error_Warning status changed

FDCAN_FLAG_BUS_OFF

Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED

Timeout reached

FDCAN format

FDCAN_CLASSIC_CAN

Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format

FDCAN_FRAME_CLASSIC

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switching

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switching

FDCAN Frame Type

FDCAN_DATA_FRAME

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage

FDCAN_HP_STORAGE_NO_FIFO

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type

FDCAN_STANDARD_ID

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN Interrupts Group

FDCAN_IT_GROUP_RX_FIFO0

RX FIFO 0 Interrupts Group: RF0LL: Rx FIFO 0 Message Lost RF0FL: Rx FIFO 0 is Full RF0NL: Rx FIFO 0 Has New Message

FDCAN_IT_GROUP_RX_FIFO1

RX FIFO 1 Interrupts Group: RF1LL: Rx FIFO 1 Message Lost RF1FL: Rx FIFO 1 is Full RF1NL: Rx FIFO 1 Has New Message

FDCAN_IT_GROUP_SMSG

Status Message Interrupts Group: TCFL: Transmission Cancellation Finished TCL: Transmission Completed HPML: High Priority Message

FDCAN_IT_GROUP_TX_FIFO_ERROR

TX FIFO Error Interrupts Group: TEFLL: Tx Event FIFO Element Lost TEFFL: Tx Event FIFO Full TEFNL: Tx Event FIFO New Entry TFEL: Tx FIFO Empty Interrupt Line

FDCAN_IT_GROUP_MISC

Misc. Interrupts Group: TOOL: Timeout Occurred MRAFL: Message RAM Access Failure TSWL: Timestamp Wraparound

FDCAN_IT_GROUP_BIT_LINE_ERROR

Bit and Line Error Interrupts Group: EPL: Error Passive ELOL: Error Logging Overflow

FDCAN_IT_GROUP_PROTOCOL_ERROR

Protocol Error Group: ARAL: Access to Reserved Address Line PEDL: Protocol Error in Data Phase Line PEAL: Protocol Error in Arbitration Phase Line WDIL: Watchdog Interrupt Line BOL: Bus_Off Status EWL: Warning Status

FDCAN Interrupts List

FDCAN_IT_LIST_RX_FIFO0

RX FIFO 0 Interrupts List

FDCAN_IT_LIST_RX_FIFO1

RX FIFO 1 Interrupts List

FDCAN_IT_LIST_SMSG

Status Message Interrupts List

FDCAN_IT_LIST_TX_FIFO_ERROR

TX FIFO Error Interrupts List

FDCAN_IT_LIST_MISC

Misc. Interrupts List

FDCAN_IT_LIST_BIT_LINE_ERROR

Bit and Line Error Interrupts List

FDCAN_IT_LIST_PROTOCOL_ERROR

Protocol Error Interrupts List

FDCAN interrupt line

FDCAN_INTERRUPT_LINE0

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames

FDCAN_ACCEPT_IN_RX_FIFO0

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode

FDCAN_MODE_NORMAL

Normal mode

FDCAN_MODE_RESTRICTED_OPERATION

Restricted Operation mode

FDCAN_MODE_BUS_MONITORING

Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK

Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK

External LoopBack mode

FDCAN protocol error code

FDCAN_PROTOCOL_ERROR_NONE

No error occurred

FDCAN_PROTOCOL_ERROR_STUFF

Stuff error

FDCAN_PROTOCOL_ERROR_FORM

Form error

FDCAN_PROTOCOL_ERROR_ACK

Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1

Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0

Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC

CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE

No change since last read

FDCAN reject remote frames

FDCAN_FILTER_REMOTE

Filter remote frames

FDCAN_REJECT_REMOTE

Reject all remote frames

FDCAN Rx FIFO 0 Interrupts

FDCAN_IT_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts

FDCAN_IT_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode

FDCAN_RX_FIFO_BLOCKING

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts

FDCAN_IT_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN Rx Location

FDCAN_RX_FIFO0

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN timeout operation

FDCAN_TIMEOUT_CONTINUOUS

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp

FDCAN_TIMESTAMP_INTERNAL

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler

FDCAN_TIMESTAMP_PRESC_1

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multiplied by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multiplied by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multiplied by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multiplied by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multiplied by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multiplied by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multiplied by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multiplied by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multiplied by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multiplied by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multiplied by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multiplied by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multiplied by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multiplied by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multiplied by 16

FDCAN Tx FIFO/Queue Mode

FDCAN_TX_FIFO_OPERATION

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts

FDCAN_IT_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location

FDCAN_TX_BUFFER0

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

28 HAL FLASH Generic Driver

28.1 FLASH Firmware driver registers structures

28.1.1 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32h5xx_hal_flash.h

Data Fields

- ***HAL_LockTypeDef Lock***
- ***uint32_t ErrorCode***
- ***uint32_t ProcedureOnGoing***
- ***uint32_t Address***
- ***uint32_t Bank***
- ***uint32_t Sector***
- ***uint32_t NbSectorsToErase***

Field Documentation

- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
  FLASH locking object
- ***uint32_t FLASH_ProcessTypeDef::ErrorCode***
  FLASH error code
- ***uint32_t FLASH_ProcessTypeDef::ProcedureOnGoing***
  Internal variable to indicate which procedure is ongoing or not in IT context
- ***uint32_t FLASH_ProcessTypeDef::Address***
  Internal variable to save address selected for program
- ***uint32_t FLASH_ProcessTypeDef::Bank***
  Internal variable to save current bank selected during erase in IT context
- ***uint32_t FLASH_ProcessTypeDef::Sector***
  Internal variable to define the current sector which is erasing
- ***uint32_t FLASH_ProcessTypeDef::NbSectorsToErase***
  Internal variable to save the remaining sectors to erase in IT context

28.2 FLASH Firmware driver API description

The following section lists the various functions of the FLASH library.

28.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB C-Bus accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- TrustZone aware
- Watermark-based area protection
- Block-based sector protection
- Error code correction (ECC)

28.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32H5xx devices.

1. FLASH Memory IO Programming functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Flash memory programming by 128 bits (user area, OBKeys) and 16 bits (OTP and Flash high-cycle data area)
   - There Two modes of programming :
     - Polling mode using HAL_FLASH_Program() function
     - Interrupt mode using HAL_FLASH_Program_IT() function
2. Interrupts and flags management functions :
   - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
   - Callback functions are called when the flash operations are finished : HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise HAL_FLASH_OperationErrorCallback()
   - Get error flag status by calling HAL_FLASH_GetError()
3. Option bytes management functions :
   - Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
   - Launch the reload of the option bytes using HAL_FLASH_OB_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

Note: The contents of the Flash memory are not guaranteed if a device reset occurs during a Flash memory operation.

28.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:

- ***HAL_FLASH_Program()***
- ***HAL_FLASH_Program_IT()***
- ***HAL_FLASH_IRQHandler()***
- ***HAL_FLASH_EndOfOperationCallback()***
- ***HAL_FLASH_OperationErrorCallback()***

28.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- ***HAL_FLASH_Unlock()***
- ***HAL_FLASH_Lock()***
- ***HAL_FLASH_OB_Unlock()***
- ***HAL_FLASH_OB_Lock()***
- ***HAL_FLASH_OB_Launch()***

28.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:

- ***HAL_FLASH_GetError()***

28.2.6 Detailed description of functions

HAL_FLASH_Program

Function name

HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t FlashAddress, uint32_t DataAddress)

Function description

Program a quad-word at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type
- **FlashAddress:** specifies the address to be programmed. This parameter shall be aligned to the Flash word (128-bit)
- **DataAddress:** specifies the address of data to be programmed This parameter shall be 32-bit aligned

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_Program_IT

Function name

HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t FlashAddress, uint32_t DataAddress)

Function description

Program a quad-word at a specified address with interrupt enabled.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type
- **FlashAddress:** specifies the address to be programmed. This parameter shall be aligned to the Flash word (128-bit)
- **DataAddress:** specifies the address of data to be programmed This parameter shall be 32-bit aligned

Return values

- **HAL:** Status

HAL_FLASH_IRQHandler

Function name

void HAL_FLASH_IRQHandler (void)

Function description

This function handles FLASH interrupt request.

Return values

- **None:**

HAL_FLASH_EndOfOperationCallback

Function name

void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)

Function description

FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased) Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_OperationErrorCallback

Function name

void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)

Function description

FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector number which returned an error Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_Unlock (void)

Function description

Unlock the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_Lock (void)

Function description

Locks the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)

Function description

Unlock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)

Function description

Lock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Launch

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)

Function description

Launch the option bytes loading.

Return values

- **HAL:** Status

HAL_FLASH_GetError

Function name

uint32_t HAL_FLASH_GetError (void)

Function description

Get the specific FLASH error flag.

Return values

- **HAL_FLASH_ERRORCode:** The returned value can be:
  - HAL_FLASH_ERROR_NONE : No error set
  - HAL_FLASH_ERROR_WRP : Write Protection Error
  - HAL_FLASH_ERROR_PGS : Program Sequence Error
  - HAL_FLASH_ERROR_STRB : Strobe Error
  - HAL_FLASH_ERROR_INC : Inconsistency Error
  - HAL_FLASH_ERROR_OBK : OBK Error
  - HAL_FLASH_ERROR_OBKW : OBK Write Error
  - HAL_FLASH_ERROR_OB_CHANGE : Option Byte Change Error
  - HAL_FLASH_ERROR_ECCC : ECC Single Correction Error
  - HAL_FLASH_ERROR_ECCD : ECC Double Detection Error

FLASH_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)

Function description

Wait for a FLASH operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout

Return values

- **HAL_StatusTypeDef:** HAL Status

28.3 FLASH Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 FLASH

FLASH

FLASH Banks

FLASH_BANK_1

Bank 1

FLASH_BANK_2

Bank 2

FLASH_BANK_BOTH

Bank1 and Bank2

FLASH Block-Base Attributes

FLASH_BB_SEC

Flash Block-Based Security Attributes

FLASH_BB_PRIV

Flash Block-Based Privilege Attributes

FLASH Error Code

HAL_FLASH_ERROR_NONE

No error

HAL_FLASH_ERROR_WRP

Write Protection Error

HAL_FLASH_ERROR_PGS

Program Sequence Error

HAL_FLASH_ERROR_STRB

Strobe Error

HAL_FLASH_ERROR_INC

Inconsistency Error

HAL_FLASH_ERROR_OBK

OBK Error

HAL_FLASH_ERROR_OBKW

OBK Write Error

HAL_FLASH_ERROR_OB_CHANGE

Option Byte Change Error

HAL_FLASH_ERROR_ECCC

ECC Single Correction Error

HAL_FLASH_ERROR_ECCD

ECC Double Detection Error

FLASH Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency This parameter can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait states
  - FLASH_LATENCY_11: FLASH Eleven wait states
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

Return value:

- none

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency This return value can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait states
  - FLASH_LATENCY_11: FLASH Eleven wait states
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP : End of FLASH Operation Interrupt
  - FLASH_IT_WRPERR : Write Protection Error Interrupt
  - FLASH_IT_PGSERR : Program Sequence Error Interrupt
  - FLASH_IT_STRBERR : Strobe Error Interrupt
  - FLASH_IT_INCERR : Inconsistency Error Interrupt
  - FLASH_IT_OBKERR : OBK Error Interrupt
  - FLASH_IT_OBKWERR : OBK Write Error Interrupt
  - FLASH_IT_OPTCHANGEERR : Option Byte Change Error Interrupt
  - FLASH_IT_ECCC : Single ECC Error Correction Interrupt

Return value:

- none

__HAL_FLASH_ENABLE_IT_NS

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP : End of FLASH Operation Interrupt
  - FLASH_IT_WRPERR : Write Protection Error Interrupt
  - FLASH_IT_PGSERR : Program Sequence Error Interrupt
  - FLASH_IT_STRBERR : Strobe Error Interrupt
  - FLASH_IT_INCERR : Inconsistency Error Interrupt
  - FLASH_IT_OBKERR : OBK Error Interrupt
  - FLASH_IT_OBKWERR : OBK Write Error Interrupt
  - FLASH_IT_OPTCHANGEERR : Option Byte Change Error Interrupt
  - FLASH_IT_ECCC : Single ECC Error Correction Interrupt

Return value:

- none

__HAL_FLASH_DISABLE_IT_NS

__HAL_FLASH_GET_FLAG

Description:

- Checks whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values :
  - FLASH_FLAG_BSY : FLASH Busy flag
  - FLASH_FLAG_WBNE : Write Buffer Not Empty flag
  - FLASH_FLAG_EOP : End Of Operation flag
  - FLASH_FLAG_WRPERR : Write Protection Error flag
  - FLASH_FLAG_PGSERR : Program Sequence Error flag
  - FLASH_FLAG_STRBERR : Strobe Error flag
  - FLASH_FLAG_INCERR : Inconsistency Error flag
  - FLASH_FLAG_OBKERR : OBK Error flag
  - FLASH_FLAG_OBKWERR : OBK Write Error flag
  - FLASH_FLAG_OPTCHANGEERR : Option Byte Change Error flag
  - FLASH_FLAG_ECCC : Single ECC Error Correction flag
  - FLASH_FLAG_ECCD : Double Detection ECC Error flag

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

__HAL_FLASH_GET_FLAG_NS

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the specified FLASH flag.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be one of the following values :
  - FLASH_FLAG_BSY : FLASH Busy flag
  - FLASH_FLAG_WBNE : Write Buffer Not Empty flag
  - FLASH_FLAG_EOP : End Of Operation flag
  - FLASH_FLAG_WRPERR : Write Protection Error flag
  - FLASH_FLAG_PGSERR : Program Sequence Error flag
  - FLASH_FLAG_STRBERR : Strobe Error flag
  - FLASH_FLAG_INCERR : Inconsistency Error flag
  - FLASH_FLAG_OBKERR : OBK Error flag
  - FLASH_FLAG_OBKWERR : OBK Write Error flag
  - FLASH_FLAG_OPTCHANGEERR : Option Byte Change Error flag
  - FLASH_FLAG_ECCC : Single ECC Error Correction flag
  - FLASH_FLAG_ECCD : Double Detection ECC Error flag
  - FLASH_FLAG_ALL_ERRORS: All errors flags

Return value:

- none

__HAL_FLASH_CLEAR_FLAG_NS

FLASH Flag definition

FLASH_FLAG_BSY

FLASH Busy flag

FLASH_FLAG_WBNE

FLASH Write Buffer Not Empty flag

FLASH_FLAG_DBNE

FLASH data Buffer Not Empty flag

FLASH_FLAG_EOP

FLASH End Of operation flag

FLASH_FLAG_WRPERR

FLASH Write Protection Error flag

FLASH_FLAG_PGSERR

FLASH Program Sequence Error flag

FLASH_FLAG_STRBERR

FLASH Strobe Error flag

FLASH_FLAG_INCERR

FLASH Inconsistency Error flag

FLASH_FLAG_OBKERR

FLASH OBK Error flag

FLASH_FLAG_OBKWERR

FLASH OBK Write Error flag

FLASH_FLAG_OPTCHANGEERR

FLASH Option Byte change Error flag

FLASH_FLAG_ECCC

FLASH ECC Correction flag

FLASH_FLAG_ECCD

FLASH ECC Detection flag

FLASH_FLAG_SR_ERRORS

FLASH_FLAG_ECCR_ERRORS

FLASH_FLAG_ALL_ERRORS

All FLASH error flags

FLASH Interrupts definition

FLASH_IT_EOP

End of FLASH Operation interrupt enable

FLASH_IT_WRPERR

Write Protection Error interrupt enable

FLASH_IT_PGSERR

Program Sequence Error interrupt enable

FLASH_IT_STRBERR

Strobe Error interrupt enable

FLASH_IT_INCERR

Inconsistency Error interrupt enable

FLASH_IT_OBKERR

OBK Error interrupt enable

FLASH_IT_OBKWERR

OBK Write Error interrupt enable

FLASH_IT_OPTCHANGEERR

Option Byte change Error interrupt enable

FLASH_IT_ECCC

Single ECC Error Correction interrupt enable

FLASH_IT_ALL

All Flash interrupt sources

FLASH Keys

FLASH_KEY1

FLASH_KEY2

FLASH_OPT_KEY1

FLASH_OPT_KEY2

FLASH_OBK_KEY1

FLASH_OBK_KEY2

FLASH Latency

FLASH_LATENCY_0

FLASH Zero wait cycle

FLASH_LATENCY_1

FLASH One wait cycle

FLASH_LATENCY_2

FLASH Two wait cycles

FLASH_LATENCY_3

FLASH Three wait cycles

FLASH_LATENCY_4

FLASH Four wait cycles

FLASH_LATENCY_5

FLASH Five wait cycles

FLASH_LATENCY_6

FLASH Six wait cycles

FLASH_LATENCY_7

FLASH Seven wait cycles

FLASH_LATENCY_8

FLASH Eight wait cycle

FLASH_LATENCY_9

FLASH Nine wait cycle

FLASH_LATENCY_10

FLASH Ten wait cycles

FLASH_LATENCY_11

FLASH Eleven wait cycles

FLASH_LATENCY_12

FLASH Twelve wait cycles

FLASH_LATENCY_13

FLASH Thirteen wait cycles

FLASH_LATENCY_14

FLASH Fourteen wait cycles

FLASH_LATENCY_15

FLASH Fifteen wait cycles

FLASH OBK Swap Offset

FLASH_OBK_SWAP_OFFSET_NO_DATA

No data will be copied from current to alternate OBK

FLASH_OBK_SWAP_OFFSET_HDPL0

HDPL0 data will be copied from current to alternate OBK

FLASH_OBK_SWAP_OFFSET_HDPL1

HDPL0/1 data will be copied from current to alternate OBK

FLASH_OBK_SWAP_OFFSET_HDPL2

HDPL0/1/2 data will be copied from current to alternate OBK

FLASH_OBK_SWAP_OFFSET_HDPL3_S

HDPL0/1/2/3_S data will be copied from current to alternate OBK

FLASH_OBK_SWAP_OFFSET_ALL

All OBK data (511) will be copied from current to alternate OBK

FLASH Option Bytes Boot configuration

OB_BOOT_NS

Non-secure boot address

OB_BOOT_SEC

Secure boot address

FLASH Option Bytes Boot Lock

OB_BOOT_LOCK_DISABLE

Boot lock disable

OB_BOOT_LOCK_ENABLE

Boot lock enable

FLASH Product State

OB_PROD_STATE_OPEN

OB_PROD_STATE_PROVISIONING

OB_PROD_STATE_IROT_PROVISIONED

OB_PROD_STATE_TZ_CLOSED

OB_PROD_STATE_CLOSED

OB_PROD_STATE_LOCKED

OB_PROD_STATE_REGRESSION

OB_PROD_STATE_NS_REGRESSION

FLASH Option Bytes User BKPRAM ECC check

OB_BKPRAM_ECC_ENABLE

BKPRAM ECC check enable

OB_BKPRAM_ECC_DISABLE

BKPRAM ECC check disable

FLASH OB Boot UBE

OB_UBE_OEM_IROT

OEM-iRoT (user flash) selected

OB_UBE_ST_IROT

ST-iRoT (system flash) selected

FLASH BOR High Enable Status

OB_BORH_DISABLE

BOR high status bit disabled

OB_BORH_ENABLE

BOR high status bit enabled

FLASH BOR Reset Level

OB_BOR_LEVEL_1

Reset level 1 threshold

OB_BOR_LEVEL_2

Reset level 2 threshold

OB_BOR_LEVEL_3

Reset level 3 threshold

FLASH Option Bytes VDDIO2 IO HSLV

OB_IO_VDDIO2_HSLV_DISABLE

High-speed IO at low VDDIO2 voltage feature disabled

OB_IO_VDDIO2_HSLV_ENABLE

High-speed IO at low VDDIO2 voltage feature enabled

FLASH Option Bytes VDD IO HSLV

OB_IO_VDD_HSLV_DISABLE

High-speed IO at low VDD voltage feature disabled

OB_IO_VDD_HSLV_ENABLE

High-speed IO at low VDD voltage feature enabled

FLASH IWDG Counter Freeze in STANDBY

OB_IWDG_STDBY_FREEZE

IWDG counter frozen in STANDBY mode

OB_IWDG_STDBY_ACTIVE

IWDG counter active in STANDBY mode

FLASH IWDG Counter Freeze in STOP

OB_IWDG_STOP_FREEZE

IWDG counter frozen in STOP mode

OB_IWDG_STOP_ACTIVE

IWDG counter active in STOP mode

FLASH Option Bytes User IWDG Type

OB_IWDG_HW

Hardware independent watchdog

OB_IWDG_SW

Software independent watchdog

FLASH Option Bytes nRST_STDBY

OB_STANDBY_RST

Reset generated when entering in standby mode

OB_STANDBY_NORST

No reset generated when entering in standby mode

FLASH Option Bytes nRST_STOP

OB_STOP_RST

Reset generated when entering in stop mode

OB_STOP_NORST

No reset generated when entering in stop mode

FLASH Option Bytes SRAM1_3 Erase On Reset

OB_SRAM1_3_RST_ERASE

SRAM1 and SRAM3 erased when a system reset occurs

OB_SRAM1_3_RST_NOT_ERASE

SRAM1 and SRAM3 are not erased when a system reset occurs

FLASH Option Bytes User SRAM2 ECC check

OB_SRAM2_ECC_ENABLE

SRAM2 ECC check enable

OB_SRAM2_ECC_DISABLE

SRAM2 ECC check disable

FLASH Option Bytes SRAM2 Erase On Reset

OB_SRAM2_RST_ERASE

SRAM2 erased when a system reset occurs

OB_SRAM2_RST_NOT_ERASE

SRAM2 is not erased when a system reset occurs

FLASH Option Bytes User SRAM3 ECC check

OB_SRAM3_ECC_ENABLE

SRAM3 ECC check enable

OB_SRAM3_ECC_DISABLE

SRAM3 ECC check disable

FLASH OB SWAP BANK

OB_SWAP_BANK_DISABLE

Bank swap disabled

OB_SWAP_BANK_ENABLE

Bank swap enabled

FLASH OB USER Type

OB_USER_BOR_LEV

BOR reset Level

OB_USER_BORH_EN

BOR high enable status

OB_USER_IWDG_SW

Independent watchdog selection

OB_USER_WWDG_SW

Window watchdog selection

OB_USER_NRST_STOP

Reset generated when entering the stop mode

OB_USER_NRST_STDBY

Reset generated when entering the standby mode

OB_USER_IO_VDD_HSLV

High speed IO at low voltage configuration bit

OB_USER_IO_VDDIO2_HSLV

High speed IO2 at low voltage configuration bit

OB_USER_IWDG_STOP

Independent watchdog counter freeze in stop mode

OB_USER_IWDG_STDBY

Independent watchdog counter freeze in standby mode

OB_USER_BOOT_UBE

Unique Boot entry

OB_USER_SWAP_BANK

Swap banks

OB_USER_SRAM1_3_RST

SRAM1 and SRAM3 erase upon system reset

OB_USER_SRAM2_RST

SRAM2 Erase when system reset

OB_USER_BKPRAM_ECC

Backup RAM ECC detection and correction enable

OB_USER_SRAM3_ECC

SRAM3 ECC detection and correction enable

OB_USER_SRAM2_ECC

SRAM2 ECC detection and correction enable

OB_USER_SRAM1_ECC

SRAM1 ECC detection and correction enable

OB_USER_TZEN

Global TrustZone security enable

OB_USER_ALL

FLASH Option Bytes Global TrustZone

OB_TZEN_DISABLE

Global TrustZone security disabled

OB_TZEN_ENABLE

Global TrustZone security enabled

FLASH Option Bytes User WWDG Type

OB_WWDG_HW

Hardware window watchdog

OB_WWDG_SW

Software window watchdog

FLASH Option Bytes Write Protection Sectors

OB_WRP_SECTOR_0TO3

Write protection of Sector0 to Sector3

OB_WRP_SECTOR_4TO7

Write protection of Sector4 to Sector7

OB_WRP_SECTOR_8TO11

Write protection of Sector8 to Sector11

OB_WRP_SECTOR_12TO15

Write protection of Sector12 to Sector15

OB_WRP_SECTOR_16TO19

Write protection of Sector16 to Sector19

OB_WRP_SECTOR_20TO23

Write protection of Sector20 to Sector23

OB_WRP_SECTOR_24TO27

Write protection of Sector24 to Sector27

OB_WRP_SECTOR_28TO31

Write protection of Sector28 to Sector31

OB_WRP_SECTOR_32TO35

Write protection of Sector32 to Sector35

OB_WRP_SECTOR_36TO39

Write protection of Sector36 to Sector39

OB_WRP_SECTOR_40TO43

Write protection of Sector40 to Sector43

OB_WRP_SECTOR_44TO47

Write protection of Sector44 to Sector47

OB_WRP_SECTOR_48TO51

Write protection of Sector48 to Sector51

OB_WRP_SECTOR_52TO55

Write protection of Sector52 to Sector55

OB_WRP_SECTOR_56TO59

Write protection of Sector56 to Sector59

OB_WRP_SECTOR_60TO63

Write protection of Sector60 to Sector63

OB_WRP_SECTOR_64TO67

Write protection of Sector64 to Sector67

OB_WRP_SECTOR_68TO71

Write protection of Sector68 to Sector71

OB_WRP_SECTOR_72TO75

Write protection of Sector72 to Sector75

OB_WRP_SECTOR_76TO79

Write protection of Sector76 to Sector79

OB_WRP_SECTOR_80TO83

Write protection of Sector80 to Sector83

OB_WRP_SECTOR_84TO87

Write protection of Sector84 to Sector87

OB_WRP_SECTOR_88TO91

Write protection of Sector88 to Sector91

OB_WRP_SECTOR_92TO95

Write protection of Sector92 to Sector95

OB_WRP_SECTOR_96TO99

Write protection of Sector96 to Sector99

OB_WRP_SECTOR_100TO103

Write protection of Sector100 to Sector103

OB_WRP_SECTOR_104TO107

Write protection of Sector104 to Sector107

OB_WRP_SECTOR_108TO111

Write protection of Sector108 to Sector111

OB_WRP_SECTOR_112TO115

Write protection of Sector112 to Sector115

OB_WRP_SECTOR_116TO119

Write protection of Sector116 to Sector119

OB_WRP_SECTOR_120TO123

Write protection of Sector120 to Sector123

OB_WRP_SECTOR_124TO127

Write protection of Sector124 to Sector127

OB_WRP_SECTOR_ALL

Write protection of all Sectors

FLASH Operation Area

FLASH_OPERATION_AREA_BANK_1

Operation in Flash Bank 1

FLASH_OPERATION_AREA_BANK_2

Operation in Flash Bank 2

FLASH_OPERATION_AREA_SYSF

Operation in System Flash memory

FLASH_OPERATION_AREA_DATA

Operation in Flash high-cycle data area

FLASH_OPERATION_AREA_OTP

Operation in Flash OTP area

FLASH Operation Type

FLASH_OPERATION_TYPE_NONE

No Flash operation

FLASH_OPERATION_TYPE_QUADWORD

Single write operation

FLASH_OPERATION_TYPE_OBKALTERASE

OBK alternate sector erase operation

FLASH_OPERATION_TYPE_SECTORERASE

Sector erase operation

FLASH_OPERATION_TYPE_BANKERASE

Bank erase operation

FLASH_OPERATION_TYPE_MASSERASE

Mass erase operation

FLASH_OPERATION_TYPE_OPTIONCHANGE

Option change operation

FLASH_OPERATION_TYPE_OBKSWAP

OBK swap operation

FLASH Option Type

OPTIONBYTE_WRP

WRP option byte configuration

OPTIONBYTE_PROD_STATE

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_BOOTADDR

BOOT address option byte configuration

OPTIONBYTE_BOOT_LOCK

Boot lock option byte configuration

OPTIONBYTE_OTP_LOCK

OTP Lock option byte configuration

OPTIONBYTE_HDP

Hide Protection area option byte configuration

OPTIONBYTE_EDATA

Flash high-cycle data area option byte configuration

OPTIONBYTE_WMSEC

Watermark-based secure area option byte configuration

OPTIONBYTE_ALL

All option byte configuration

FLASH OTP blocks

FLASH_OTP_BLOCK_0

OTP Block0

FLASH_OTP_BLOCK_1

OTP Block1

FLASH_OTP_BLOCK_2

OTP Block2

FLASH_OTP_BLOCK_3

OTP Block3

FLASH_OTP_BLOCK_4

OTP Block4

FLASH_OTP_BLOCK_5

OTP Block5

FLASH_OTP_BLOCK_6

OTP Block6

FLASH_OTP_BLOCK_7

OTP Block7

FLASH_OTP_BLOCK_8

OTP Block8

FLASH_OTP_BLOCK_9

OTP Block9

FLASH_OTP_BLOCK_10

OTP Block10

FLASH_OTP_BLOCK_11

OTP Block11

FLASH_OTP_BLOCK_12

OTP Block12

FLASH_OTP_BLOCK_13

OTP Block13

FLASH_OTP_BLOCK_14

OTP Block14

FLASH_OTP_BLOCK_15

OTP Block15

FLASH_OTP_BLOCK_16

OTP Block16

FLASH_OTP_BLOCK_17

OTP Block17

FLASH_OTP_BLOCK_18

OTP Block18

FLASH_OTP_BLOCK_19

OTP Block19

FLASH_OTP_BLOCK_20

OTP Block20

FLASH_OTP_BLOCK_21

OTP Block21

FLASH_OTP_BLOCK_22

OTP Block22

FLASH_OTP_BLOCK_23

OTP Block23

FLASH_OTP_BLOCK_24

OTP Block24

FLASH_OTP_BLOCK_25

OTP Block25

FLASH_OTP_BLOCK_26

OTP Block26

FLASH_OTP_BLOCK_27

OTP Block27

FLASH_OTP_BLOCK_28

OTP Block28

FLASH_OTP_BLOCK_29

OTP Block29

FLASH_OTP_BLOCK_30

OTP Block30

FLASH_OTP_BLOCK_31

OTP Block31

FLASH_OTP_BLOCK_ALL

OTP All Blocks

FLASH privilege mode

FLASH_NSPRIV_GRANTED

access to non-secure Flash registers is granted to privileged or unprivileged access

FLASH_NSPRIV_DENIED

access to non-secure Flash registers is denied to non-privilege access

FLASH_SPRIV_GRANTED

access to secure Flash registers is granted to privileged or unprivileged access

FLASH_SPRIV_DENIED

access to secure Flash registers is denied to non-privilege access

FLASH Programming Delay

FLASH_PROGRAMMING_DELAY_0

programming delay set for Flash running at 70 MHz or below

FLASH_PROGRAMMING_DELAY_1

programming delay set for Flash running between 70 MHz and 185 MHz

FLASH_PROGRAMMING_DELAY_2

programming delay set for Flash running between 185 MHz and 225 MHz

FLASH_PROGRAMMING_DELAY_3

programming delay set for Flash at startup

FLASH Sectors

FLASH_SECTOR_0

Sector Number 0

FLASH_SECTOR_1

Sector Number 1

FLASH_SECTOR_2

Sector Number 2

FLASH_SECTOR_3

Sector Number 3

FLASH_SECTOR_4

Sector Number 4

FLASH_SECTOR_5

Sector Number 5

FLASH_SECTOR_6

Sector Number 6

FLASH_SECTOR_7

Sector Number 7

FLASH_SECTOR_8

Sector Number 8

FLASH_SECTOR_9

Sector Number 9

FLASH_SECTOR_10

Sector Number 10

FLASH_SECTOR_11

Sector Number 11

FLASH_SECTOR_12

Sector Number 12

FLASH_SECTOR_13

Sector Number 13

FLASH_SECTOR_14

Sector Number 14

FLASH_SECTOR_15

Sector Number 15

FLASH_SECTOR_16

Sector Number 16

FLASH_SECTOR_17

Sector Number 17

FLASH_SECTOR_18

Sector Number 18

FLASH_SECTOR_19

Sector Number 19

FLASH_SECTOR_20

Sector Number 20

FLASH_SECTOR_21

Sector Number 21

FLASH_SECTOR_22

Sector Number 22

FLASH_SECTOR_23

Sector Number 23

FLASH_SECTOR_24

Sector Number 24

FLASH_SECTOR_25

Sector Number 25

FLASH_SECTOR_26

Sector Number 26

FLASH_SECTOR_27

Sector Number 27

FLASH_SECTOR_28

Sector Number 28

FLASH_SECTOR_29

Sector Number 29

FLASH_SECTOR_30

Sector Number 30

FLASH_SECTOR_31

Sector Number 31

FLASH_SECTOR_32

Sector Number 32

FLASH_SECTOR_33

Sector Number 33

FLASH_SECTOR_34

Sector Number 34

FLASH_SECTOR_35

Sector Number 35

FLASH_SECTOR_36

Sector Number 36

FLASH_SECTOR_37

Sector Number 37

FLASH_SECTOR_38

Sector Number 38

FLASH_SECTOR_39

Sector Number 39

FLASH_SECTOR_40

Sector Number 40

FLASH_SECTOR_41

Sector Number 41

FLASH_SECTOR_42

Sector Number 42

FLASH_SECTOR_43

Sector Number 43

FLASH_SECTOR_44

Sector Number 44

FLASH_SECTOR_45

Sector Number 45

FLASH_SECTOR_46

Sector Number 46

FLASH_SECTOR_47

Sector Number 47

FLASH_SECTOR_48

Sector Number 48

FLASH_SECTOR_49

Sector Number 49

FLASH_SECTOR_50

Sector Number 50

FLASH_SECTOR_51

Sector Number 51

FLASH_SECTOR_52

Sector Number 52

FLASH_SECTOR_53

Sector Number 53

FLASH_SECTOR_54

Sector Number 54

FLASH_SECTOR_55

Sector Number 55

FLASH_SECTOR_56

Sector Number 56

FLASH_SECTOR_57

Sector Number 57

FLASH_SECTOR_58

Sector Number 58

FLASH_SECTOR_59

Sector Number 59

FLASH_SECTOR_60

Sector Number 60

FLASH_SECTOR_61

Sector Number 61

FLASH_SECTOR_62

Sector Number 62

FLASH_SECTOR_63

Sector Number 63

FLASH_SECTOR_64

Sector Number 64

FLASH_SECTOR_65

Sector Number 65

FLASH_SECTOR_66

Sector Number 66

FLASH_SECTOR_67

Sector Number 67

FLASH_SECTOR_68

Sector Number 68

FLASH_SECTOR_69

Sector Number 69

FLASH_SECTOR_70

Sector Number 70

FLASH_SECTOR_71

Sector Number 71

FLASH_SECTOR_72

Sector Number 72

FLASH_SECTOR_73

Sector Number 73

FLASH_SECTOR_74

Sector Number 74

FLASH_SECTOR_75

Sector Number 75

FLASH_SECTOR_76

Sector Number 76

FLASH_SECTOR_77

Sector Number 77

FLASH_SECTOR_78

Sector Number 78

FLASH_SECTOR_79

Sector Number 79

FLASH_SECTOR_80

Sector Number 80

FLASH_SECTOR_81

Sector Number 81

FLASH_SECTOR_82

Sector Number 82

FLASH_SECTOR_83

Sector Number 83

FLASH_SECTOR_84

Sector Number 84

FLASH_SECTOR_85

Sector Number 85

FLASH_SECTOR_86

Sector Number 86

FLASH_SECTOR_87

Sector Number 87

FLASH_SECTOR_88

Sector Number 88

FLASH_SECTOR_89

Sector Number 89

FLASH_SECTOR_90

Sector Number 90

FLASH_SECTOR_91

Sector Number 91

FLASH_SECTOR_92

Sector Number 92

FLASH_SECTOR_93

Sector Number 93

FLASH_SECTOR_94

Sector Number 94

FLASH_SECTOR_95

Sector Number 95

FLASH_SECTOR_96

Sector Number 96

FLASH_SECTOR_97

Sector Number 97

FLASH_SECTOR_98

Sector Number 98

FLASH_SECTOR_99

Sector Number 99

FLASH_SECTOR_100

Sector Number 100

FLASH_SECTOR_101

Sector Number 101

FLASH_SECTOR_102

Sector Number 102

FLASH_SECTOR_103

Sector Number 103

FLASH_SECTOR_104

Sector Number 104

FLASH_SECTOR_105

Sector Number 105

FLASH_SECTOR_106

Sector Number 106

FLASH_SECTOR_107

Sector Number 107

FLASH_SECTOR_108

Sector Number 108

FLASH_SECTOR_109

Sector Number 109

FLASH_SECTOR_110

Sector Number 110

FLASH_SECTOR_111

Sector Number 111

FLASH_SECTOR_112

Sector Number 112

FLASH_SECTOR_113

Sector Number 113

FLASH_SECTOR_114

Sector Number 114

FLASH_SECTOR_115

Sector Number 115

FLASH_SECTOR_116

Sector Number 116

FLASH_SECTOR_117

Sector Number 117

FLASH_SECTOR_118

Sector Number 118

FLASH_SECTOR_119

Sector Number 119

FLASH_SECTOR_120

Sector Number 120

FLASH_SECTOR_121

Sector Number 121

FLASH_SECTOR_122

Sector Number 122

FLASH_SECTOR_123

Sector Number 123

FLASH_SECTOR_124

Sector Number 124

FLASH_SECTOR_125

Sector Number 125

FLASH_SECTOR_126

Sector Number 126

FLASH_SECTOR_127

Sector Number 127

FLASH Type Erase

FLASH_TYPEERASE_SECTORS

Secure flash sectors erase activation

FLASH_TYPEERASE_SECTORS_NS

Non-secure flash sectors erase activation

FLASH_TYPEERASE_MASSERASE

Secure flash mass erase activation

FLASH_TYPEERASE_MASSERASE_NS

Non-secure flash mass erase activation

FLASH_TYPEERASE_OBK_ALT

Flash OBK erase activation

FLASH Program Type

FLASH_TYPEPROGRAM_QUADWORD

Program a quad-word (128-bit) at a specified secure address

FLASH_TYPEPROGRAM_QUADWORD_NS

Program a quad-word (128-bit) at a specified non-secure address

FLASH_TYPEPROGRAM_QUADWORD_OBK

Program a quad-word (128-bit) of OBK to current sector

FLASH_TYPEPROGRAM_QUADWORD_OBK_ALT

Program a quad-word (128-bit) of OBK to alternate sector

FLASH_TYPEPROGRAM_HALFWORD_EDATA

Program a flash high-cycle data half-word (16-bit)at a specified secure address

FLASH_TYPEPROGRAM_HALFWORD_EDATA_NS

Program a flash high-cycle data half-word (16-bit)at a specified non-secure address

FLASH_TYPEPROGRAM_HALFWORD_OTP

Program an OTP half-word (16-bit)at a specified address

FLASH WRP State

OB_WRPSTATE_DISABLE

Disable the write protection of the desired flash sectors

OB_WRPSTATE_ENABLE

Enable the write protection of the desired flash sectors

29 HAL FLASH Extension Driver

29.1 FLASHEx Firmware driver registers structures

29.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32h5xx_hal_flash_ex.h

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t Sector***
- ***uint32_t NbSectors***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  Mass erase or sector Erase. This parameter can be a value of ***FLASH_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***
  Select banks to erase when Mass erase is enabled. This parameter can be a value of ***FLASH_Banks*** (FLASH_BANK_BOTH should be used only for mass erase)
- ***uint32_t FLASH_EraseInitTypeDef::Sector***
  Initial FLASH sector to erase when Mass erase is disabled This parameter can be a value of ***FLASH_Sectors***
- ***uint32_t FLASH_EraseInitTypeDef::NbSectors***
  Number of sectors to be erased. This parameter can be a value between 1 and (max number of sectors in the bank - value of initial sector)

29.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32h5xx_hal_flash_ex.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t ProductState***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t USERConfig2***
- ***uint32_t Banks***
- ***uint32_t WRPState***
- ***uint32_t WRPSector***
- ***uint32_t BootConfig***
- ***uint32_t BootAddr***
- ***uint32_t BootLock***
- ***uint32_t OTPBlockLock***
- ***uint32_t HDPStartSector***
- ***uint32_t HDPEndSector***
- ***uint32_t EDATASize***
- ***uint32_t WMSecStartSector***
- ***uint32_t WMSecEndSector***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  Option byte to be configured. This parameter can be a value of ***FLASH_Option_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::ProductState***
  Set the product state. This parameter can be a value of ***FLASH_OB_Product_State***

- ***uint32_t FLASH_OBProgramInitTypeDef::USERType***
  Select the User Option Byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig***
  Value of the User Option Byte (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_BOR_LEVEL***, ***FLASH_OB_USER_BORH_EN***, ***FLASH_OB_USER_IWDG_SW***, ***FLASH_OB_USER_WWDG_SW***, ***FLASH_OB_USER_nRST_STOP***, ***FLASH_OB_USER_nRST_STANDBY***, ***FLASH_OB_USER_IO_VDD_HSLV***, ***FLASH_OB_USER_IO_VDDIO2_HSLV***, ***FLASH_OB_USER_IWDG_STOP***, ***FLASH_OB_USER_IWDG_STANDBY***, ***FLASH_OB_USER_BOOT_UBE***, ***FLASH_OB_USER_SWAP_BANK***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig2***
  Value of the User Option Byte (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_SRAM1_3_RST***, ***FLASH_OB_USER_SRAM2_RST***, ***FLASH_OB_USER_BKPRAM_ECC***, ***FLASH_OB_USER_SRAM3_ECC***, ***FLASH_OB_USER_SRAM2_ECC***, ***Section 28.3.1 FLASH_OB_USER_SRAM1_RST***, ***Section 28.3.1 FLASH_OB_USER_SRAM1_ECC***, ***FLASH_OB_USER_TZEN***
- ***uint32_t FLASH_OBProgramInitTypeDef::Banks***
  Select banks for WRP , HDP and secure area configuration. This parameter must be a value of ***FLASH_Banks***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
  Write protection activation or deactivation. This parameter can be a value of ***FLASH_WRP_State***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector***
  Specifies the sector(s) to be write protected. The value of this parameter depend on device used within the same series
- ***uint32_t FLASH_OBProgramInitTypeDef::BootConfig***
  Specifies if the Boot Address to be configured: secure or non-secure. This parameter must be a value of ***FLASH_OB_BOOT_CONFIG*** enumeration
- ***uint32_t FLASH_OBProgramInitTypeDef::BootAddr***
  Boot address (used for OPTIONBYTE_BOOTADDR). This parameter must be a value between 0x0 and 0xFFFFFF00
- ***uint32_t FLASH_OBProgramInitTypeDef::BootLock***
  Configuration of the boot lock (used for OPTIONBYTE_BOOT_LOCK). This parameter must be a value of ***FLASH_OB_BOOT_LOCK***
- ***uint32_t FLASH_OBProgramInitTypeDef::OTPBlockLock***
  Specifies the OTP block(s) to be locked. This parameter must be a value of ***FLASH_OTP_Blocks***
- ***uint32_t FLASH_OBProgramInitTypeDef::HDPStartSector***
  Start sector of HDP area (used for OPTIONBYTE_HDP). This parameter must be a value between 0 and (max number of sectors in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::HDPEndSector***
  End sector of HDP area (used for OPTIONBYTE_HDP). This parameter must be a value between 0 and (max number of sectors in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::EDATASize***
  Specifies the number of Flash high-cycle sectors. This parameter must be a value between 0 and 8 (sectors)
- ***uint32_t FLASH_OBProgramInitTypeDef::WMSecStartSector***
  Start sector of secure area (used for OPTIONBYTE_WMSEC). This parameter must be a value between 0 and (max number of sectors in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::WMSecEndSector***
  End sector of secure area (used for OPTIONBYTE_WMSEC). This parameter must be a value between 0 and (max number of sectors in the bank - 1)

29.1.3 FLASH_BBAttributesTypeDef

FLASH_BBAttributesTypeDef is defined in the stm32h5xx_hal_flash_ex.h

Data Fields

- ***uint32_t Bank***
- ***uint32_t BBAttributesType***

- ***uint32_t BBAttributes_array***

Field Documentation

- ***uint32_t FLASH_BBAttributesTypeDef::Bank***
  Selection of the associated bank of Block-based Area. This parameter must be a value of ***FLASH_Banks***
- ***uint32_t FLASH_BBAttributesTypeDef::BBAttributesType***
  Block-Based Attributes type. This parameter must be a value of ***FLASH_BB_Attributes***
- ***uint32_t FLASH_BBAttributesTypeDef::BBAttributes_array[FLASH_BLOCKBASED_NB_REG]***
  Each bit specifies the block-based attribute configuration of a sector: 0 means sector non-protected, 1 means sector protected. Protection (secure or privilege) depends on BBAttributesType value

29.1.4 FLASH_OperationTypeDef

FLASH_OperationTypeDef is defined in the stm32h5xx_hal_flash_ex.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t FlashArea***
- ***uint32_t Address***

Field Documentation

- ***uint32_t FLASH_OperationTypeDef::OperationType***
  Flash operation Type. This parameter must be a value of ***FLASH_Operation_Type***
- ***uint32_t FLASH_OperationTypeDef::FlashArea***
  Flash operation memory area. This parameter must be a value of ***FLASH_Operation_Area***
- ***uint32_t FLASH_OperationTypeDef::Address***
  Flash operation Address offset. This parameter is given by bank, and must be a value between 0x0 and 0xFFFF0

29.1.5 FLASH_HDPExtensionTypeDef

FLASH_HDPExtensionTypeDef is defined in the stm32h5xx_hal_flash_ex.h

Data Fields

- ***uint32_t Banks***
- ***uint32_t NbSectors***

Field Documentation

- ***uint32_t FLASH_HDPExtensionTypeDef::Banks***
  Selection of the associated bank of HDP Area. This parameter must be a value of ***FLASH_Banks***
- ***uint32_t FLASH_HDPExtensionTypeDef::NbSectors***
  Number of sectors to be HDP extended. This parameter can be a value between 1 and max number of sectors in the bank

29.2 FLASHEx Firmware driver API description

The following section lists the various functions of the FLASHEx library.

29.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32H5xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- Product State protection
- Write protection
- Secure access only protection
- Bank / register swapping (when Dual-Bank)
- Watermark-based secure protection
- Block-based secure protection
- Block-based privilege protection

- Hide Protection areas

29.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32H5xx devices. It includes

1. FLASH Memory Erase functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Erase function: Sector erase, bank erase and dual-bank mass erase
   - There are two modes of erase :
     - Polling Mode using HAL_FLASHEx_Erase()
     - Interrupt Mode using HAL_FLASHEx_Erase_IT()
2. Option Bytes Programming functions: Use HAL_FLASHEx_OBProgram() to:
   - Configure the write protection per bank
   - Set the Product State
   - Program the user Option Bytes
   - Configure the watermark security for each area
   - Configure the Hide protection areas
   - Configure the Boot addresses
3. Get Option Bytes Configuration function: Use HAL_FLASHEx_OBGetConfig() to:
   - Get the value of a write protection area
   - Get the Product State
   - Get the value of the user Option Bytes
   - Get the configuration of watermark security areas
   - Get the configuration of Hide protection areas
   - Get the value of a boot address
4. Block-based secure / privilege area configuration function: Use HAL_FLASHEx_ConfigBBAttributes()
   - Bit-field allowing to secure or un-secure each sector
   - Bit-field allowing to privilege or un-privilege each sector
5. Get the block-based secure / privilege area configuration function: Use HAL_FLASHEx_GetConfigBBAttributes()
   - Return the configuration of the block-based security and privilege for all the sectors
6. Privilege mode configuration function: Use HAL_FLASHEx_ConfigPrivMode()
   - FLASH register can be protected against non-privilege accesses
7. Get the privilege mode configuration function: Use HAL_FLASHEx_GetPrivMode()
   - Return if the FLASH registers are protected against non-privilege accesses

29.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extended FLASH programming operations Operations.

This section contains the following APIs:

- ***HAL_FLASHEx_Erase()***
- ***HAL_FLASHEx_Erase_IT()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***
- ***HAL_FLASHEx_OBK_Unlock()***
- ***HAL_FLASHEx_OBK_Lock()***
- ***HAL_FLASHEx_OBK_Swap()***
- ***HAL_FLASHEx_GetOperation()***

29.2.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * SectorError)

Function description

Perform a mass erase or erase the specified FLASH memory sectors.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **SectorError:** pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased).

Return values

- **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)

Function description

Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values

- **HAL:** Status

HAL_FLASHEx_OBProgram

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Program option bytes.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

Notes

- To configure any option bytes, the option lock bit OPTLOCK must be cleared with the call of HAL_FLASH_OB_Unlock() function.
- New option bytes configuration will be taken into account in two cases: after an option bytes launch through the call of HAL_FLASH_OB_Launch()after a power-on reset (BOR reset or exit from Standby/ Shutdown modes)

HAL_FLASHEx_OBGetConfig

Function name

void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Get the Option byte configuration.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

Return values

- **None:**

Notes

- The parameter Banks of the pOBInit structure must be set exclusively to FLASH_BANK_1 or FLASH_BANK_2, as this parameter is use to get the given Bank WRP, PCROP and secured area configuration.

HAL_FLASHEx_OBK_Unlock

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBK_Unlock (void)

Function description

Unlock the FLASH OBK register access.

Return values

- **HAL:** Status

HAL_FLASHEx_OBK_Lock

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBK_Lock (void)

Function description

Locks the FLASH OBK register access.

Return values

- **HAL:** Status

HAL_FLASHEx_OBK_Swap

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBK_Swap (uint32_t SwapOffset)

Function description

Swap the FLASH Option Bytes Keys (OBK)

Parameters

- **SwapOffset:** Specifies the number of keys to be swapped. This parameter can be a value between 0 (no OBK data swapped) and 511 (all OBK data swapped). Typical value are available in FLASH OBK Swap Offset

Return values

- **HAL:** Status

HAL_FLASHEx_GetOperation

Function name

void HAL_FLASHEx_GetOperation (FLASH_OperationTypeDef * pFlashOperation)

Function description

Return the on-going Flash Operation.

Parameters

- **pFlashOperation:** [out] pointer to a FLASH_OperationTypeDef structure that contains the Flash operation information.

Return values

- **None:**

HAL_FLASHEx_ConfigBBAttributes

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigBBAttributes (FLASH_BBAttributesTypeDef * pBBAttributes)

Function description

Configure the block-based secure area.

Parameters

- **pBBAttributes:** pointer to an FLASH_BBAttributesTypeDef structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

Notes

- The field pBBAttributes->Bank should indicate which area is requested for the block-based attributes.
- The field pBBAttributes->BBAttributesType should indicate which block-base attribute type is requested: Secure or Privilege.

HAL_FLASHEx_GetConfigBBAttributes

Function name

void HAL_FLASHEx_GetConfigBBAttributes (FLASH_BBAttributesTypeDef * pBBAttributes)

Function description

Return the block-based attributes.

Parameters

- **pBBAttributes:** [in/out] pointer to an FLASH_BBAttributesTypeDef structure that contains the configuration information.

Return values

- **None:**

Notes

- The field pBBAttributes->Bank should indicate which area is requested for the block-based attributes.
- The field pBBAttributes->BBAttributesType should indicate which block-base attribute type is requested: Secure or Privilege.

HAL_FLASHEx_ConfigPrivMode

Function name

void HAL_FLASHEx_ConfigPrivMode (uint32_t PrivMode)

Function description

Configuration of the privilege attribute.

Parameters

- **PrivMode:** indicate privilege mode configuration This parameter can be one of the following values:
  - FLASH_SPRIV_GRANTED: access to secure Flash registers is granted to privileged or unprivileged access
  - FLASH_SPRIV_DENIED: access to secure Flash registers is denied to unprivileged access
  - FLASH_NSPRIV_GRANTED: access to non-secure Flash registers is granted to privileged or unprivileged access
  - FLASH_NSPRIV_DENIED: access to non-secure Flash registers is denied to unprivilege access

Return values

- **None:**

HAL_FLASHEx_GetPrivMode

Function name

uint32_t HAL_FLASHEx_GetPrivMode (void)

Function description

Return the value of the privilege attribute.

Return values

- **It:** indicates the privilege mode configuration. This return value can be one of the following values:
  - FLASH_SPRIV_GRANTED: access to secure Flash registers is granted to privileged or unprivileged access
  - FLASH_SPRIV_DENIED: access to secure Flash registers is denied to unprivileged access
  - FLASH_NSPRIV_GRANTED: access to non-secure Flash registers is granted to privileged or unprivileged access
  - FLASH_NSPRIV_DENIED: access to Flash registers is denied to unprivilege accessP

HAL_FLASHEx_ConfigSecInversion

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigSecInversion (uint32_t SecInvState)

Function description

Configuration of the security inversion.

Parameters

- **SecInvState:** indicate the flash security state configuration This parameter can be one of the following values:
  - FLASH_SEC_INV_DISABLE: Security state of Flash is not inverted
  - FLASH_SEC_INV_ENABLE: Security state of Flash is inverted

Return values

- **HAL:** Status

HAL_FLASHEx_GetSecInversion

Function name

uint32_t HAL_FLASHEx_GetSecInversion (void)

Function description

Return the value of the security inversion.

Return values

- **It:** indicates the flash security state configuration This return value can be one of the following values:
  - FLASH_SEC_INV_DISABLE: Security state of Flash is not inverted
  - FLASH_SEC_INV_ENABLE: Security state of Flash is inverted

HAL_FLASHEx_ConfigHDPExtension

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigHDPExtension (const FLASH_HDPExtensionTypeDef * pHDPExtension)

Function description

Configure the HDP extension area.

Parameters

- **pHDPExtension:** pointer to an FLASH_HDPExtentionTypeDef structure that contains the configuration information.

Return values

- **HAL:** Status

Notes

- The field pHDPExtension->Banks should indicate which area is requested for the HDP extension.
- The field pHDPExtension->NbSectors should indicate the number of sector to be added to the HDP area.

FLASH_Erase_Sector

Function name

void FLASH_Erase_Sector (uint32_t Sector, uint32_t Banks)

Function description

Erase the specified FLASH memory sector.

Parameters

- **Sector:** FLASH sector to erase This parameter can be a value of FLASH Sectors
- **Banks:** Bank(s) where the sector will be erased This parameter can be one of the following values:
  - FLASH_BANK_1: Sector in bank 1 to be erased
  - FLASH_BANK_2: Sector in bank 2 to be erased

Return values

- **None:**

29.3 FLASHEx Firmware driver defines

The following section lists the various define and macros of the module.

29.3.1 FLASHEx

FLASHEx

FLASHEx Exported Macros

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_SMART_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH smart prefetch buffer.

Return value:

- None

__HAL_FLASH_SMART_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH smart prefetch buffer.

Return value:

- None

__HAL_FLASH_SET_PROGRAM_DELAY

Description:

- Set the FLASH Programming Delay.

Parameters:

- __DELAY__: FLASH Programming Delay This parameter can be a value of

Return value:

- none

__HAL_FLASH_GET_PROGRAM_DELAY

Description:

- Get the FLASH Programming Delay.

Return value:

- FLASH: Programming Delay This return value can be a value of

FLASHEx Private macros to check input parameters

IS_FLASH_TYPEERASE

IS_WRPSTATE

IS_OPTIONBYTE

IS_OB_PRODUCT_STATE

IS_OB_USER_BOR_LEVEL

IS_OB_USER_BORH_EN

IS_OB_USER_IWDG

IS_OB_USER_WWDG

IS_OB_USER_STOP

IS_OB_USER_STANDBY

IS_OB_USER_IO_VDD_HSLV

IS_OB_USER_IO_VDDIO2_HSLV

IS_OB_USER_IWDG_STOP

IS_OB_USER_IWDG_STDBY

IS_OB_USER_BOOT_UBE

IS_OB_USER_SWAP_BANK

IS_OB_USER_SRAM1_3_RST

IS_OB_USER_SRAM2_RST

IS_OB_USER_BKPRAM_ECC

IS_OB_USER_SRAM3_ECC

IS_OB_USER_SRAM2_ECC

IS_OB_USER_TZEN

IS_OB_USER_TYPE

IS_OB_BOOT_CONFIG

IS_OB_BOOT_LOCK

IS_FLASH_BB_EXCLUSIVE

IS_FLASH_CFGPRIVMODE

IS_FLASH_CFGSECINV

IS_FLASH_EDATA_SIZE

30 HAL FMAC Generic Driver

30.1 FMAC Firmware driver registers structures

30.1.1 __FMAC_HandleTypeDef

__FMAC_HandleTypeDef is defined in the stm32h5xx_hal_fmac.h

Data Fields

- ***FMAC_TypeDef * Instance***
- ***uint32_t FilterParam***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***int16_t * pInput***
- ***uint16_t InputCurrentSize***
- ***uint16_t * pInputSize***
- ***int16_t * pOutput***
- ***uint16_t OutputCurrentSize***
- ***uint16_t * pOutputSize***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***DMA_HandleTypeDef * hdmaPreload***
- ***void(* ErrorCallback***
- ***void(* HalfGetDataCallback***
- ***void(* GetDataCallback***
- ***void(* HalfOutputDataReadyCallback***
- ***void(* OutputDataReadyCallback***
- ***void(* FilterConfigCallback***
- ***void(* FilterPreloadCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_FMAC_StateTypeDef State***
- ***__IO HAL_FMAC_StateTypeDef RdState***
- ***__IO HAL_FMAC_StateTypeDef WrState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***FMAC_TypeDef* __FMAC_HandleTypeDef::Instance***
  Register base address
- ***uint32_t __FMAC_HandleTypeDef::FilterParam***
  Filter configuration (operation and parameters). Set to 0 if no valid configuration was applied.
- ***uint8_t __FMAC_HandleTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t __FMAC_HandleTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***int16_t* __FMAC_HandleTypeDef::pInput***
  Pointer to FMAC input data buffer
- ***uint16_t __FMAC_HandleTypeDef::InputCurrentSize***
  Number of the input elements already written into FMAC

- ***uint16_t* __FMAC_HandleTypeDef::pInputSize***
  Number of input elements to write (memory allocated to pInput). In case of early interruption of the filter operation, its value will be updated.
- ***int16_t* __FMAC_HandleTypeDef::pOutput***
  Pointer to FMAC output data buffer
- ***uint16_t __FMAC_HandleTypeDef::OutputCurrentSize***
  Number of the output elements already read from FMAC
- ***uint16_t* __FMAC_HandleTypeDef::pOutputSize***
  Number of output elements to read (memory allocated to pOutput). In case of early interruption of the filter operation, its value will be updated.
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaIn***
  FMAC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaOut***
  FMAC peripheral output data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaPreload***
  FMAC peripheral preloaded data (X1, X2 and Y) DMA handle parameters
- ***void(* __FMAC_HandleTypeDef::ErrorCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC error callback
- ***void(* __FMAC_HandleTypeDef::HalfGetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get half data callback
- ***void(* __FMAC_HandleTypeDef::GetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get data callback
- ***void(* __FMAC_HandleTypeDef::HalfOutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC half output data ready callback
- ***void(* __FMAC_HandleTypeDef::OutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC output data ready callback
- ***void(* __FMAC_HandleTypeDef::FilterConfigCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter configuration callback
- ***void(* __FMAC_HandleTypeDef::FilterPreloadCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter preload callback
- ***void(* __FMAC_HandleTypeDef::MspInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp Init callback
- ***void(* __FMAC_HandleTypeDef::MspDeInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp DeInit callback
- ***HAL_LockTypeDef __FMAC_HandleTypeDef::Lock***
  FMAC locking object
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::State***
  FMAC state related to global handle management This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::RdState***
  FMAC state related to read operations (access to Y buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::WrState***
  FMAC state related to write operations (access to X1 buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO uint32_t __FMAC_HandleTypeDef::ErrorCode***
  FMAC peripheral error code This parameter can be a value of *FMAC_Error_Code*

30.1.2 FMAC_FilterConfigTypeDef

FMAC_FilterConfigTypeDef is defined in the stm32h5xx_hal_fmac.h

Data Fields

- ***uint8_t InputBaseAddress***
- ***uint8_t InputBufferSize***
- ***uint32_t InputThreshold***

- ***uint8_t CoeffBaseAddress***
- ***uint8_t CoeffBufferSize***
- ***uint8_t OutputBaseAddress***
- ***uint8_t OutputBufferSize***
- ***uint32_t OutputThreshold***
- ***int16_t * pCoeffA***
- ***uint8_t CoeffASize***
- ***int16_t * pCoeffB***
- ***uint8_t CoeffBSize***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***uint32_t Clip***
- ***uint32_t Filter***
- ***uint8_t P***
- ***uint8_t Q***
- ***uint8_t R***

Field Documentation

- ***uint8_t FMAC_FilterConfigTypeDef::InputBaseAddress***
  Base address of the input buffer (X1) within the internal memory (0x00 to 0xFF). Ignored if InputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::InputBufferSize***
  Number of 16-bit words allocated to the input buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::InputThreshold***
  Input threshold: the buffer full flag will be set if the number of free spaces in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBaseAddress***
  Base address of the coefficient buffer (X2) within the internal memory (0x00 to 0xFF). Ignored if CoeffBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBufferSize***
  Number of 16-bit words allocated to the coefficient buffer. 0 if a previous configuration should be kept.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBaseAddress***
  Base address of the output buffer (Y) within the internal memory (0x00 to 0xFF). Ignored if OuputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBufferSize***
  Number of 16-bit words allocated to the output buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::OutputThreshold***
  Output threshold: the buffer empty flag will be set if the number of unread values in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffA***
  [IIR only] Initialization of the coefficient vector A. If not needed, it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffASize***
  Size of the coefficient vector A.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffB***
  Initialization of the coefficient vector B. If not needed (re-use of a previously loaded buffer), it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBSize***
  Size of the coefficient vector B.
- ***uint8_t FMAC_FilterConfigTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.

- ***uint8_t FMAC_FilterConfigTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint32_t FMAC_FilterConfigTypeDef::Clip***
  Enable or disable the clipping feature. If the q1.15 range is exceeded, wrapping is done when the clipping feature is disabled and saturation is done when the clipping feature is enabled. This parameter can be a value of ***FMAC_Clip_State***.
- ***uint32_t FMAC_FilterConfigTypeDef::Filter***
  Filter type. This parameter can be a value of ***FMAC_Functions*** (filter related values).
- ***uint8_t FMAC_FilterConfigTypeDef::P***
  Parameter P (vector length, number of filter taps, etc.).
- ***uint8_t FMAC_FilterConfigTypeDef::Q***
  Parameter Q (vector length, etc.). Ignored if not needed.
- ***uint8_t FMAC_FilterConfigTypeDef::R***
  Parameter R (gain, etc.). Ignored if not needed.

30.2 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

30.2.1 How to use this driver

The FMAC HAL driver can be used as follows:

1. Initialize the FMAC low level resources by implementing the HAL_FMAC_MspInit():
   - Enable the FMAC interface clock using __HAL_RCC_FMAC_CLK_ENABLE().
   - In case of using interrupts (e.g. access configured as FMAC_BUFFER_ACCESS_IT):
     - Configure the FMAC interrupt priority using HAL_NVIC_SetPriority().
     - Enable the FMAC IRQ handler using HAL_NVIC_EnableIRQ().
     - In FMAC IRQ handler, call HAL_FMAC_IRQHandler().
   - In case of using DMA to control data transfer (e.g. access configured as FMAC_BUFFER_ACCESS_DMA):
     - Enable the DMA interface clock using __HAL_RCC_DMA1_CLK_ENABLE() or __HAL_RCC_DMA2_CLK_ENABLE() depending on the used DMA instance.
     - Enable the DMAMUX1 interface clock using __HAL_RCC_DMAMUX1_CLK_ENABLE().
     - If the initialization of the internal buffers (coefficients, input, output) is done via DMA, configure and enable one DMA channel for managing data transfer from memory to memory (preload channel).
     - If the input buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from memory to peripheral (input channel).
     - If the output buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from peripheral to memory (output channel).
     - Associate the initialized DMA handle(s) to the FMAC DMA handle(s) using __HAL_LINKDMA().
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the enabled DMA channel(s) using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Initialize the FMAC HAL using HAL_FMAC_Init(). This function resorts to HAL_FMAC_MspInit() for low-level initialization.

3. Configure the FMAC processing (filter) using HAL_FMAC_FilterConfig() or HAL_FMAC_FilterConfig_DMA(). This function:
   - Defines the memory area within the FMAC internal memory (input, coefficients, output) and the associated threshold (input, output).
   - Configures the filter and its parameters:
     - Finite Impulse Response (FIR) filter (also known as convolution).
     - Infinite Impulse Response (IIR) filter (direct form 1).
   - Choose the way to access to the input and output buffers: none, polling, DMA, IT. "none" means the input and/or output data will be handled by another IP (ADC, DAC, etc.).
   - Enable the error interruptions in the input access and/or the output access is done through IT/DMA. If an error occurs, the interruption will be triggered in loop. In order to recover, the user will have to reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init. Optionally, he can also disable the interrupt using __HAL_FMAC_DISABLE_IT; the error status will be kept, but no more interrupt will be triggered.
   - Write the provided coefficients into the internal memory using polling mode (HAL_FMAC_FilterConfig()) or DMA (HAL_FMAC_FilterConfig_DMA()). In the DMA case, HAL_FMAC_FilterConfigCallback() is called when the handling is over.
4. Optionally, the user can enable the error interruption related to saturation by calling __HAL_FMAC_ENABLE_IT. This helps in debugging the filter. If a saturation occurs, the interruption will be triggered in loop. In order to recover, the user will have to:
   - Disable the interruption by calling __HAL_FMAC_DISABLE_IT if the user wishes to continue all the same.
   - Reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init.
5. Optionally, preload input (FIR, IIR) and output (IIR) data using HAL_FMAC_FilterPreload() or HAL_FMAC_FilterPreload_DMA(). In the DMA case, HAL_FMAC_FilterPreloadCallback() is called when the handling is over. This step is optional as the filter can be started without preloaded data.
6. Start the FMAC processing (filter) using HAL_FMAC_FilterStart(). This function also configures the output buffer that will be filled from the circular internal output buffer. The function returns immediately without updating the provided buffer. The IP processing will be active until HAL_FMAC_FilterStop() is called.
7. If the input internal buffer is accessed via DMA, HAL_FMAC_HalfGetDataCallback() will be called to indicate that half of the input buffer has been handled.
8. If the input internal buffer is accessed via DMA or interrupt, HAL_FMAC_GetDataCallback() will be called to require new input data. It will be provided through HAL_FMAC_AppendFilterData() if the DMA isn't in circular mode.
9. If the output internal buffer is accessed via DMA, HAL_FMAC_HalfOutputDataReadyCallback() will be called to indicate that half of the output buffer has been handled.
10. If the output internal buffer is accessed via DMA or interrupt, HAL_FMAC_OutputDataReadyCallback() will be called to require a new output buffer. It will be provided through HAL_FMAC_ConfigFilterOutputBuffer() if the DMA isn't in circular mode.
11. In all modes except none, provide new input data to be processed via HAL_FMAC_AppendFilterData(). This function should only be called once the previous input data has been handled (the preloaded input data isn't concerned).
12. In all modes except none, provide a new output buffer to be filled via HAL_FMAC_ConfigFilterOutputBuffer(). This function should only be called once the previous user's output buffer has been filled.
13. In polling mode, handle the input and output data using HAL_FMAC_PollFilterData(). This function:
   - Write the user's input data (provided via HAL_FMAC_AppendFilterData()) into the FMAC input memory area.
   - Read the FMAC output memory area and write it into the user's output buffer. It will return either when:
   - the user's output buffer is filled.
   - the user's input buffer has been handled. The unused data (unread input data or free output data) will not be saved. The user will have to use the updated input and output sizes to keep track of them.
14. Stop the FMAC processing (filter) using HAL_FMAC_FilterStop().
15. Call HAL_FMAC_DeInit() to de-initialize the FMAC peripheral. This function resorts to HAL_FMAC_MspDeInit() for low-level de-initialization.

30.2.2 Callback registration

The compilation define USE_HAL_FMAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_FMAC_RegisterCallback() to register a user callback. Function HAL_FMAC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_FMAC_UnRegisterCallback() to reset a callback to the default weak function. HAL_FMAC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit.

By default, after the HAL_FMAC_Init() and when the state is HAL_FMAC_STATE_RESET all callbacks are set to the corresponding weak functions: examples GetDataCallback(), OutputDataReadyCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_FMAC_Init() and HAL_FMAC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_FMAC_Init() and HAL_FMAC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_FMAC_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_FMAC_STATE_READY or HAL_FMAC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_FMAC_RegisterCallback() before calling HAL_FMAC_DeInit() or HAL_FMAC_Init() function.

When the compilation define USE_HAL_FMAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

30.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the FMAC peripheral and the associated handle
- DeInitialize the FMAC peripheral
- Initialize the FMAC MSP (MCU Specific Package)
- De-Initialize the FMAC MSP
- Register a User FMAC Callback
- Unregister a FMAC CallBack

This section contains the following APIs:

- ***HAL_FMAC_Init()***
- ***HAL_FMAC_DeInit()***

- *HAL_FMAC_MspInit()*
- *HAL_FMAC_MspDeInit()*
- *HAL_FMAC_RegisterCallback()*
- *HAL_FMAC_UnRegisterCallback()*

30.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the FMAC peripheral: memory area, filter type and parameters, way to access to the input and output memory area (none, polling, IT, DMA).
- Start the FMAC processing (filter).
- Handle the input data that will be provided into FMAC.
- Handle the output data provided by FMAC.
- Stop the FMAC processing (filter).

This section contains the following APIs:

- *HAL_FMAC_FilterConfig()*
- *HAL_FMAC_FilterConfig_DMA()*
- *HAL_FMAC_FilterPreload()*
- *HAL_FMAC_FilterPreload_DMA()*
- *HAL_FMAC_FilterStart()*
- *HAL_FMAC_AppendFilterData()*
- *HAL_FMAC_ConfigFilterOutputBuffer()*
- *HAL_FMAC_PollFilterData()*
- *HAL_FMAC_FilterStop()*

30.2.5 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt: the user's input data is half written (DMA only) or completely written.
- DMA or Interrupt: the user's output buffer is half filled (DMA only) or completely filled.
- DMA or Interrupt: error handling.

This section contains the following APIs:

- *HAL_FMAC_ErrorCallback()*
- *HAL_FMAC_HalfGetDataCallback()*
- *HAL_FMAC_GetDataCallback()*
- *HAL_FMAC_HalfOutputDataReadyCallback()*
- *HAL_FMAC_OutputDataReadyCallback()*
- *HAL_FMAC_FilterConfigCallback()*
- *HAL_FMAC_FilterPreloadCallback()*

30.2.6 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_FMAC_IRQHandler()*

30.2.7 Peripheral State and Error functions

This subsection provides functions allowing to

- Check the FMAC state
- Get error code

This section contains the following APIs:

- *HAL_FMAC_GetState()*

- *HAL_FMAC_GetError()*

30.2.8 Detailed description of functions

HAL_FMAC_Init

Function name

HAL_StatusTypeDef HAL_FMAC_Init (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC peripheral and the associated handle.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_DeInit

Function name

HAL_StatusTypeDef HAL_FMAC_DeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC peripheral.

Parameters

- **hfmac:** pointer to a FMAC structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_MspInit

Function name

void HAL_FMAC_MspInit (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_MspDeInit

Function name

void HAL_FMAC_MspDeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_RegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID, pFMAC_CallbackTypeDef pCallback)

Function description

Register a User FMAC Callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID
- **pCallback:** pointer to the Callback function.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The User FMAC Callback is to be used instead of the weak predefined callback.
- The HAL_FMAC_RegisterCallback() may be called before HAL_FMAC_Init() in HAL_FMAC_STATE_RESET to register callbacks for HAL_FMAC_MSPINIT_CB_ID and HAL_FMAC_MSPDEINIT_CB_ID.

HAL_FMAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_UnRegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a FMAC CallBack.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The FMAC callback is redirected to the weak predefined callback.
- The HAL_FMAC_UnRegisterCallback() may be called before HAL_FMAC_Init() in HAL_FMAC_STATE_RESET to register callbacks for HAL_FMAC_MSPINIT_CB_ID and HAL_FMAC_MSPDEINIT_CB_ID.

HAL_FMAC_FilterConfig

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using polling mode.

HAL_FMAC_FilterConfig_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig_DMA (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using DMA.

HAL_FMAC_FilterPreload

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using polling mode.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterPreload_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload_DMA (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using DMA.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterStart

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStart (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Start the FMAC processing according to the existing FMAC configuration.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** pointer to buffer where output data of FMAC processing will be stored in the next steps. If it is set to NULL, the output will not be read and it will be up to an external IP to empty the output buffer.
- **pOutputSize:** pointer to the size of the output buffer. The number of read data will be written here.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_AppendFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_AppendFilterData (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint16_t * pInputSize)

Function description

Provide a new input buffer that will be loaded into the FMAC input memory area.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** New input vector (additional input data).
- **pInputSize:** Size of the input vector (if all the data can't be written, it will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ConfigFilterOutputBuffer

Function name

HAL_StatusTypeDef HAL_FMAC_ConfigFilterOutputBuffer (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Provide a new output buffer to be filled with the data computed by FMAC unit.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** New output vector.
- **pOutputSize:** Size of the output vector (if the vector can't be entirely filled, pOutputSize will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_PollFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_PollFilterData (FMAC_HandleTypeDef * hfmac, uint32_t Timeout)

Function description

Handle the input and/or output data in polling mode.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **Timeout:** timeout value.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- This function writes the previously provided user's input data and fills the previously provided user's output buffer, according to the existing FMAC configuration (polling mode only). The function returns when the input data has been handled or when the output data is filled. The possible unused data isn't kept. It will be up to the user to handle it. The previously provided pInputSize and pOutputSize will be used to indicate to the size of the read/written data to the user.

HAL_FMAC_FilterStop

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStop (FMAC_HandleTypeDef * hfmac)

Function description

Stop the FMAC processing.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ErrorCallback

Function name

void HAL_FMAC_ErrorCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC error callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfGetDataCallback

Function name

void HAL_FMAC_HalfGetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get half data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetDataCallback

Function name

void HAL_FMAC_GetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfOutputDataReadyCallback

Function name

void HAL_FMAC_HalfOutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC half output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_OutputDataReadyCallback

Function name

void HAL_FMAC_OutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterConfigCallback

Function name

void HAL_FMAC_FilterConfigCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter configuration callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterPreloadCallback

Function name

void HAL_FMAC_FilterPreloadCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter preload callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_IRQHandler

Function name

void HAL_FMAC_IRQHandler (FMAC_HandleTypeDef * hfmac)

Function description

Handle FMAC interrupt request.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetState

Function name

HAL_FMAC_StateTypeDef HAL_FMAC_GetState (const FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC state.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_FMAC_StateTypeDef:** FMAC state

HAL_FMAC_GetError

Function name

uint32_t HAL_FMAC_GetError (const FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC peripheral error.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **uint32_t:** Error bit-map based on FMAC Error code

Notes

- The returned error is a bit-map combination of possible errors.

30.3 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

30.3.1 FMAC

FMAC

FMAC Buffer Access

FMAC_BUFFER_ACCESS_NONE

Buffer handled by an external IP (ADC for instance)

FMAC_BUFFER_ACCESS_DMA

Buffer accessed through DMA

FMAC_BUFFER_ACCESS_POLLING

Buffer accessed through polling

FMAC_BUFFER_ACCESS_IT

Buffer accessed through interruptions

FMAC Clip State

FMAC_CLIP_DISABLED

Clipping disabled

FMAC_CLIP_ENABLED

Clipping enabled

FMAC Data Buffer Threshold

FMAC_THRESHOLD_1

Input: Buffer full flag set if the number of free spaces in the buffer is less than 1. Output: Buffer empty flag set if the number of unread values in the buffer is less than 1.

FMAC_THRESHOLD_2

Input: Buffer full flag set if the number of free spaces in the buffer is less than 2. Output: Buffer empty flag set if the number of unread values in the buffer is less than 2.

FMAC_THRESHOLD_4

Input: Buffer full flag set if the number of free spaces in the buffer is less than 4. Output: Buffer empty flag set if the number of unread values in the buffer is less than 4.

FMAC_THRESHOLD_8

Input: Buffer full flag set if the number of free spaces in the buffer is less than 8. Output: Buffer empty flag set if the number of unread values in the buffer is less than 8.

FMAC_THRESHOLD_NO_VALUE

The configured threshold value shouldn't be changed

FMAC Error code

HAL_FMAC_ERROR_NONE

No error

HAL_FMAC_ERROR_SAT

Saturation error

HAL_FMAC_ERROR_UNFL

Underflow error

HAL_FMAC_ERROR_OVFL

Overflow error

HAL_FMAC_ERROR_DMA

DMA error

HAL_FMAC_ERROR_RESET

Reset error

HAL_FMAC_ERROR_PARAM

Parameter error

HAL_FMAC_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_FMAC_ERROR_TIMEOUT

Timeout error

FMAC Exported Macros

__HAL_FMAC_RESET_HANDLE_STATE

Description:

- Reset FMAC handle state.

Parameters:

- __HANDLE__: FMAC handle.

Return value:

- None

__HAL_FMAC_ENABLE_IT

Description:

- Enable the specified FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_DISABLE_IT

Description:

- Disable the FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_GET_IT

Description:

- Check whether the specified FMAC interrupt occurred or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation Error Flag

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_FMAC_CLEAR_IT

Description:

- Clear specified FMAC interrupt status.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to clear.

Return value:

- None

__HAL_FMAC_GET_FLAG

Description:

- Check whether the specified FMAC status flag is set or not.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation error Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_FMAC_CLEAR_FLAG

Description:

- Clear specified FMAC status flag.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to clear.

Return value:

- None

__HAL_FMAC_GET_IT_SOURCE

Description:

- Check whether the specified FMAC interrupt is enabled or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be one of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- FlagStatus

FMAC status flags

FMAC_FLAG_YEMPTY

Y Buffer Empty Flag

FMAC_FLAG_X1FULL

X1 Buffer Full Flag

FMAC_FLAG_OVFL

Overflow Error Flag

FMAC_FLAG_UNFL

Underflow Error Flag

FMAC_FLAG_SAT

Saturation Error Flag (this helps in debugging a filter)

FMAC Functions

FMAC_FUNC_LOAD_X1

Load X1 buffer

FMAC_FUNC_LOAD_X2

Load X2 buffer

FMAC_FUNC_LOAD_Y

Load Y buffer

FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

FMAC Interrupts Enable bit

FMAC_IT_RIEN

Read Interrupt Enable

FMAC_IT_WIEN

Write Interrupt Enable

FMAC_IT_OVFLIEN

Overflow Error Interrupt Enable

FMAC_IT_UNFLIEN

Underflow Error Interrupt Enable

FMAC_IT_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

31 HAL GPIO Generic Driver

31.1 GPIO Firmware driver registers structures

31.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32h5xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be a value of ***GPIO_pins***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

31.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

31.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

- The external interrupt/event controller consists of up to 39 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

31.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To set the level of several pins and reset level of several other pins in same cycle, use HAL_GPIO_WriteMultipleStatePin().
7. To lock pin configuration until next reset use HAL_GPIO_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

31.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_GPIO_Init()***
- ***HAL_GPIO_DeInit()***

31.2.4 IO attributes functions

This section contains the following APIs:

- ***HAL_GPIO_ConfigPinAttributes()***
- ***HAL_GPIO_GetConfigPinAttributes()***

31.2.5 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, const GPIO_InitTypeDef * pGPIO_Init)

Function description

Initialize the GPIOx peripheral according to the specified parameters in the pGPIO_Init.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **pGPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

Notes

- If GPIOx peripheral pin is used in EXTI_MODE and the pin is secure in case the system implements the security (TZEN=1), it is up to the secure application to insure that the corresponding EXTI line is set secure.

HAL_GPIO_DeInit

Function name

void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)

Function description

De-initialize the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Read the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  - GPIO_PIN_RESET: to clear the port pin
  - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_WriteMultipleStatePin

Function name

void HAL_GPIO_WriteMultipleStatePin (GPIO_TypeDef * GPIOx, uint16_t PinReset, uint16_t PinSet)

Function description

Set and clear several pins of a dedicated port in same cycle.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **PinReset:** specifies the port bits to be reset This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.
- **PinSet:** specifies the port bits to be set This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses.
- Both PinReset and PinSet combinations shall not get any common bit, else assert would be triggered.
- At least one of the two parameters used to set or reset shall be different from zero.

HAL_GPIO_TogglePin

Function name

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Toggle the specified GPIO pin.

Parameters

- **GPIOx:** where x can be (A..I) to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the pin to be toggled.

Return values

- **None:**

HAL_GPIO_EnableHighSPeedLowVoltage

Function name

void HAL_GPIO_EnableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Enable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

HAL_GPIO_DisableHighSPeedLowVoltage

Function name

void HAL_GPIO_DisableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Disable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

HAL_GPIO_LockPin

Function name

HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Lock GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Rising_Callback

Function name

void HAL_GPIO_EXTI_Rising_Callback (uint16_t GPIO_Pin)

Function description

EXTI line rising detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Falling_Callback

Function name

void HAL_GPIO_EXTI_Falling_Callback (uint16_t GPIO_Pin)

Function description

EXTI line falling detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_ConfigPinAttributes

Function name

void HAL_GPIO_ConfigPinAttributes (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t PinAttributes)

Function description

Configure the GPIO pins attributes.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the pin(s) to configure the secure attribute
- **PinAttributes:** specifies the pin(s) to be set in secure mode, other being set non secured.

Return values

- **None:**

Notes

- Available attributes are to secure GPIO pin(s), so this function is only available in secure

HAL_GPIO_GetConfigPinAttributes

Function name

HAL_StatusTypeDef HAL_GPIO_GetConfigPinAttributes (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t * pPinAttributes)

Function description

Get the GPIO pins attributes.

Parameters

- **GPIOx:** where x can be (A..I) for stm32h56xxx and stm32h57xxx family lines and (A..D or H) for stm32h503xx family line to select the GPIO peripheral for STM32H5 family
- **GPIO_Pin:** specifies the single pin to get the secure attribute from
- **pPinAttributes:** pointer to return the pin attributes.

Return values

- **HAL:** Status.

Notes

- Available attributes are to secure GPIO pin(s), so this function is only available in secure

31.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 GPIO

GPIO

GPIO attributes

GPIO_PIN_SEC

Secure pin attribute

GPIO_PIN_NSEC

Non-secure pin attribute

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_RISING_IT

Description:

- Check whether the specified EXTI line is rising edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_RISING_IT

Description:

- Clear the EXTI's line rising pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FALLING_IT

Description:

- Check whether the specified EXTI line is falling edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FALLING_IT

Description:

- Clear the EXTI's line falling pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line(s).

Parameters:

- __EXTI_LINE__: specifies the EXTI line to set. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clear the EXTI line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT

< Input Floating Mode Output Push Pull Mode

GPIO_MODE_OUTPUT_PP

Output Open Drain Mode

GPIO_MODE_OUTPUT_OD

Alternate Function Push Pull Mode

GPIO_MODE_AF_PP

Alternate Function Open Drain Mode

GPIO_MODE_AF_OD

Analog Mode

GPIO_MODE_ANALOG

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_RISING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

GPIO pins

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_ALL

GPIO_PIN_MASK

GPIO pull

GPIO_NOPULL

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO speed

GPIO_SPEED_FREQ_LOW

Low speed

GPIO_SPEED_FREQ_MEDIUM

Medium speed

GPIO_SPEED_FREQ_HIGH

High speed

GPIO_SPEED_FREQ_VERY_HIGH

Very-high speed

32 HAL GPIO Extension Driver

32.1 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

32.1.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_RTC_50HZ

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_TRACE

GPIO_AF0_CSLEEP

GPIO_AF0_CSTOP

GPIO_AF0_CRS

GPIO_AF1_TIM1

GPIO_AF1_TIM2

GPIO_AF1_TIM16

GPIO_AF1_TIM17

GPIO_AF1_LPTIM1

GPIO_AF2_LPTIM3

GPIO_AF2_SAI1

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5

GPIO_AF2_TIM12

GPIO_AF2_TIM15

GPIO_AF3_I3C1

GPIO_AF3_LPTIM2

GPIO_AF3_LPTIM3

GPIO_AF3_LPUART1

GPIO_AF3_OCTOSPI1

GPIO_AF3_TIM1

GPIO_AF3_TIM8

GPIO_AF4_CEC

GPIO_AF4_DCMI

GPIO_AF4_PSSI

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF4_I2C3

GPIO_AF4_I2C4

GPIO_AF4_LPTIM1

GPIO_AF4_LPTIM2

GPIO_AF4_SPI1

GPIO_AF4_TIM15

GPIO_AF4_USART1

GPIO_AF5_CEC

GPIO_AF5_I3C1

GPIO_AF5_SPI3

GPIO_AF5_LPTIM1

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI4

GPIO_AF5_SPI5

GPIO_AF5_SPI6

GPIO_AF6_I2C4

GPIO_AF6_OCTOSPI1

GPIO_AF6_SAI1

GPIO_AF6_SPI3

GPIO_AF6_SPI4

GPIO_AF6_UART4

GPIO_AF6_UART12

GPIO_AF6_USART10

GPIO_AF6_UCPD1

GPIO_AF7_SDMMC1

GPIO_AF7_SPI2

GPIO_AF7_SPI3

GPIO_AF7_SPI6

GPIO_AF7_UART7

GPIO_AF7_UART8

GPIO_AF7_UART12

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF7_USART6

GPIO_AF7_USART10

GPIO_AF7_USART11

GPIO_AF8_LPUART1

GPIO_AF8_SAI2

GPIO_AF8_SDMMC1

GPIO_AF8_SPI6

GPIO_AF8_UART4

GPIO_AF8_UART5

GPIO_AF8_UART8

GPIO_AF9_FDCAN1

GPIO_AF9_FDCAN2

GPIO_AF9_FMC

GPIO_AF9_OCTOSPI1

GPIO_AF9_SDMMC2

GPIO_AF9_TIM13

GPIO_AF9_TIM14

GPIO_AF10_CRS

GPIO_AF10_FMC

GPIO_AF10_OCTOSPI1

GPIO_AF10_SAI2

GPIO_AF10_SDMMC2

GPIO_AF10_TIM8

GPIO_AF10_USB

GPIO_AF11_ETH

GPIO_AF11_FMC

GPIO_AF11_OCTOSPI1

GPIO_AF11_SDMMC2

GPIO_AF11_UART7

GPIO_AF11_UART9

GPIO_AF11_UCPD1

GPIO_AF12_FMC

GPIO_AF12_SDMMC1

GPIO_AF13_DCMI

GPIO_AF13_PSSI

GPIO_AF13_FMC

GPIO_AF13_LPTIM5

GPIO_AF14_LPTIM3

GPIO_AF14_LPTIM4

GPIO_AF14_LPTIM5

GPIO_AF14_LPTIM6

GPIO_AF14_TIM2

GPIO_AF14_UART5

GPIO_AF15_EVENTOUT

IS_GPIO_AF

GPIOEx_Get Port Index

GPIO_GET_INDEX

33 HAL GTZC Generic Driver

33.1 GTZC Firmware driver registers structures

33.1.1 MPCBB_Attribute_ConfigTypeDef

MPCBB_Attribute_ConfigTypeDef is defined in the stm32h5xx_hal_gtzc.h

Data Fields

- ***uint32_t MPCBB_SecConfig_array***
- ***uint32_t MPCBB_PrivConfig_array***
- ***uint32_t MPCBB_LockConfig_array***

Field Documentation

- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_SecConfig_array[GTZC_MPCBB_NB_VCTR_REG_MAX]***
  Each element specifies secure access mode for a super-block. Each bit corresponds to a block inside the super-block. 0 means non-secure, 1 means secure
- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_PrivConfig_array[GTZC_MPCBB_NB_VCTR_REG_MAX]***
  Each element specifies privilege access mode for a super-block. Each bit corresponds to a block inside the super-block. 0 means non-privilege, 1 means privilege
- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_LockConfig_array[GTZC_MPCBB_NB_LCK_VCTR_REG_MAX]***
  Each bit specifies the lock configuration of a super-block (32 blocks). 0 means unlocked, 1 means locked

33.1.2 MPCBB_ConfigTypeDef

MPCBB_ConfigTypeDef is defined in the stm32h5xx_hal_gtzc.h

Data Fields

- ***uint32_t SecureRWIllegalMode***
- ***uint32_t InvertSecureState***
- ***MPCBB_Attribute_ConfigTypeDef AttributeConfig***

Field Documentation

- ***uint32_t MPCBB_ConfigTypeDef::SecureRWIllegalMode***
  Secure read/write illegal access field. It can be a value of ***GTZC_MPCBB_SecureRWIllegalMode***
- ***uint32_t MPCBB_ConfigTypeDef::InvertSecureState***
  Default security state field (can be inverted or not). It can be a value of ***GTZC_MPCBB_InvertSecureState***
- ***MPCBB_Attribute_ConfigTypeDef MPCBB_ConfigTypeDef::AttributeConfig***
  MPCBB attribute configuration sub-structure

33.1.3 MPCWM_ConfigTypeDef

MPCWM_ConfigTypeDef is defined in the stm32h5xx_hal_gtzc.h

Data Fields

- ***uint32_t AreaId***
- ***uint32_t Offset***
- ***uint32_t Length***
- ***uint32_t Attribute***
- ***uint32_t Lock***
- ***uint32_t AreaStatus***

Field Documentation

- ***uint32_t MPCWM_ConfigTypeDef::AreaId***
  Area identifier field. It can be a value of ***GTZC_MPCWM_AreaId***

- ***uint32_t MPCWM_ConfigTypeDef::Offset***
  Offset of the watermark area, starting from the selected memory base address. It must aligned on 128KB for FMC and OCTOSPI memories, and on 32-byte for BKPSRAM
- ***uint32_t MPCWM_ConfigTypeDef::Length***
  Length of the watermark area, starting from the selected Offset. It must aligned on 128KB for FMC and OCTOSPI memories, and on 32-byte for BKPSRAM
- ***uint32_t MPCWM_ConfigTypeDef::Attribute***
  Attributes of the watermark area. It can be a value of ***GTZC_MPCWM_Attribute***
- ***uint32_t MPCWM_ConfigTypeDef::Lock***
  Lock of the watermark area. It can be a value of ***GTZC_MPCWM_Lock***
- ***uint32_t MPCWM_ConfigTypeDef::AreaStatus***
  Status of the watermark area. It can be set to ENABLE or DISABLE

33.2 GTZC Firmware driver API description

The following section lists the various functions of the GTZC library.

33.2.1 GTZC main features

- Global TrustZone Controller (GTZC) composed of three sub-blocks:
  - TZSC: TrustZone security controller This sub-block defines the secure/privileged state of master and slave peripherals. It also controls the secure/privileged state of subregions for the watermark memory peripheral controller (MPCWM).
  - MPCBB: Block-Based memory protection controller This sub-block defines the secure/privileged state of all blocks (512-byte pages) of the associated SRAM.
  - TZIC: TrustZone illegal access controller This sub-block gathers all illegal access events in the system and generates a secure interrupt towards NVIC.
- These sub-blocks are used to configure TrustZone system security in a product having bus agents with programmable-security and privileged attributes (securable) such as:
  - on-chip RAM with programmable secure and/or privilege blocks (pages)
  - AHB and APB peripherals with programmable security and/or privilege access
  - AHB master granted as secure and/or privilege
  - off-chip memories with secure and/or privilege areas
- TZIC accessible only with secure privileged transactions.
- Secure and non-secure access supported for privileged and unprivileged part of TZSC and MPCBB
- Set of registers to define product security settings:
  - Secure and privilege blocks for internal memories
  - Secure and privilege regions for external memories
  - Secure and privileged access mode for securable peripherals
    ==========

33.2.2 How to use this driver

The GTZC HAL driver can be used as follows:

1. Configure or get back securable peripherals attributes using HAL_GTZC_TZSC_ConfigPeriphAttributes() / HAL_GTZC_TZSC_GetConfigPeriphAttributes()
2. Configure or get back MPCWM memories attributes using HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes() / HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes()
3. Lock TZSC sub-block or get lock status using HAL_GTZC_TZSC_Lock() / HAL_GTZC_TZSC_GetLock()
4. Configure or get back MPCBB memories complete configuration using HAL_GTZC_MPCBB_ConfigMem() / HAL_GTZC_MPCBB_GetConfigMem()
5. Configure or get back MPCBB memories attributes using HAL_GTZC_MPCBB_ConfigMemAttributes() / HAL_GTZC_MPCBB_GetConfigMemAttributes()

6. Lock MPCBB configuration or get lock status using HAL_GTZC_MPCBB_Lock() / HAL_GTZC_MPCBB_GetLock()
7. Lock MPCBB super-blocks or get lock status using HAL_GTZC_MPCBB_LockConfig() / HAL_GTZC_MPCBB_GetLockConfig()
8. Illegal access detection can be configured through TZIC sub-block using following functions: HAL_GTZC_TZIC_DisableIT() / HAL_GTZC_TZIC_EnableIT()
9. Illegal access flags can be retrieved through HAL_GTZC_TZIC_GetFlag() and HAL_GTZC_TZIC_ClearFlag() functions
10. Illegal access interrupt service routines are served by HAL_GTZC_IRQHandler() and user can add his own code using HAL_GTZC_TZIC_Callback()

33.2.3 Detailed description of functions

HAL_GTZC_TZSC_ConfigPeriphAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_ConfigPeriphAttributes (uint32_t PeriphId, uint32_t PeriphAttributes)

Function description

Configure TZSC on a single peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **PeriphAttributes:** Peripheral attributes, see GTZC TZSC peripheral attribute values.

Return values

- **HAL:** status.

Notes

- Secure and non-secure attributes can only be set from the secure state when the system implements the security (TZEN=1).
- Privilege and non-privilege attributes can only be set from the privilege state when TZEN=0 or TZEN=1
- Security and privilege attributes can be set independently.
- Default state is non-secure and unprivileged access allowed.

HAL_GTZC_TZSC_GetConfigPeriphAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_GetConfigPeriphAttributes (uint32_t PeriphId, uint32_t * PeriphAttributes)

Function description

Get TZSC configuration on a single peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **PeriphAttributes:** Peripheral attribute pointer. This parameter can be a value of GTZC TZSC peripheral attribute values. If PeriphId target a single peripheral, pointer on a single element. If all peripherals selected (GTZC_PERIPH_ALL), pointer to an array of GTZC_TZSC_PERIPH_NUMBER elements is to be provided.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes (uint32_t MemBaseAddress, const MPCWM_ConfigTypeDef * pMPCWM_Desc)

Function description

Configure a TZSC-MPCWM area.

Parameters

- **MemBaseAddress:** WM identifier.
- **pMPCWM_Desc:** TZSC-MPCWM descriptor pointer. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes (uint32_t MemBaseAddress, MPCWM_ConfigTypeDef * pMPCWM_Desc)

Function description

Get a TZSC-MPCWM area configuration.

Parameters

- **MemBaseAddress:** WM identifier.
- **pMPCWM_Desc:** pointer to a TZSC-MPCWM descriptor. When the WaterMark memory supports two sub-regions A and B. pMPCWM_Desc argument must point to an array of two MPCWM_ConfigTypeDef structures. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_Lock

Function name

void HAL_GTZC_TZSC_Lock (GTZC_TZSC_TypeDef * TZSC_Instance)

Function description

Lock TZSC configuration.

Parameters

- **TZSC_Instance:** TZSC sub-block instance.

Notes

- This function locks the configuration of TZSC_SECCFGRx and TZSC_PRIVCFGRx registers until next reset

HAL_GTZC_TZSC_GetLock

Function name

uint32_t HAL_GTZC_TZSC_GetLock (const GTZC_TZSC_TypeDef * TZSC_Instance)

Function description

Get TZSC configuration lock state.

Parameters

- **TZSC_Instance:** TZSC sub-block instance.

Return values

- **Lock:** State (GTZC_TZSC_LOCK_OFF or GTZC_TZSC_LOCK_ON)

HAL_GTZC_MPCBB_ConfigMem

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_ConfigMem (uint32_t MemBaseAddress, const MPCBB_ConfigTypeDef * pMPCBB_desc)

Function description

Set a complete MPCBB configuration on the SRAM passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pMPCBB_desc:** pointer to MPCBB descriptor. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetConfigMem

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetConfigMem (uint32_t MemBaseAddress, MPCBB_ConfigTypeDef * pMPCBB_desc)

Function description

Get a complete MPCBB configuration on the SRAM passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pMPCBB_desc:** pointer to a MPCBB descriptor. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_ConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_ConfigMemAttributes (uint32_t MemAddress, uint32_t NbBlocks, const uint32_t * pMemAttributes)

Function description

Set a MPCBB attribute configuration on the SRAM passed as parameter for a number of blocks.

Parameters

- **MemAddress:** MPCBB identifier, and start block to configure (must be 512 Bytes aligned).
- **NbBlocks:** Number of blocks to configure (Block size is 512 Bytes).
- **pMemAttributes:** pointer to an array (containing "NbBlocks" elements), with each element must be GTZC_MPCBB_BLOCK_NSEC or GTZC_MPCBB_BLOCK_SEC, and GTZC_MPCBB_BLOCK_NPRIV or GTZC_MPCBB_BLOCK_PRIV.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetConfigMemAttributes (uint32_t MemAddress, uint32_t NbBlocks, uint32_t * pMemAttributes)

Function description

Get a MPCBB attribute configuration on the SRAM passed as parameter for a number of blocks.

Parameters

- **MemAddress:** MPCBB identifier, and start block to get configuration (must be 512 Bytes aligned).
- **NbBlocks:** Number of blocks to get configuration.
- **pMemAttributes:** pointer to an array (containing "NbBlocks" elements), with each element will be GTZC_MPCBB_BLOCK_NSEC or GTZC_MPCBB_BLOCK_SEC, and GTZC_MPCBB_BLOCK_NPRIV or GTZC_MPCBB_BLOCK_PRIV.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_LockConfig

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_LockConfig (uint32_t MemAddress, uint32_t NbSuperBlocks, const uint32_t * pLockAttributes)

Function description

Lock MPCBB super-blocks on the SRAM passed as parameter.

Parameters

- **MemAddress:** MPCBB start-address of super-block to configure (must be 16KBytes aligned).
- **NbSuperBlocks:** Number of super-blocks to configure.
- **pLockAttributes:** pointer to an array (containing "NbSuperBlocks" elements), with for each element: value 0 super-block is unlocked, value 1 super-block is locked (corresponds to GTZC_MPCBB_SUPERBLOCK_UNLOCKED and GTZC_MPCBB_SUPERBLOCK_LOCKED values).

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetLockConfig

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetLockConfig (uint32_t MemAddress, uint32_t NbSuperBlocks, uint32_t * pLockAttributes)

Function description

Get MPCBB super-blocks lock configuration on the SRAM passed as parameter.

Parameters

- **MemAddress:** MPCBB start-address of super-block to get configuration (must be 16KBytes aligned).
- **NbSuperBlocks:** Number of super-blocks to get configuration.
- **pLockAttributes:** pointer to an array (containing "NbSuperBlocks" elements), with for each element: value 0 super-block is unlocked, value 1 super-block is locked (corresponds to GTZC_MPCBB_SUPERBLOCK_UNLOCKED and GTZC_MPCBB_SUPERBLOCK_LOCKED values).

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_Lock

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_Lock (uint32_t MemBaseAddress)

Function description

Lock a MPCBB configuration on the SRAM base address passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.

Return values

- **HAL:** status.

Notes

- This functions locks the control register of the MPCBB until next reset.

HAL_GTZC_MPCBB_GetLock

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetLock (uint32_t MemBaseAddress, uint32_t * pLockState)

Function description

Get MPCBB configuration lock state on the SRAM base address passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pLockState:** pointer to Lock State (GTZC_MPCBB_LOCK_OFF or GTZC_MPCBB_LOCK_ON).

Return values

- **HAL:** status.

HAL_GTZC_TZIC_DisableIT

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_DisableIT (uint32_t PeriphId)

Function description

Disable the interrupt associated to a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_TZIC_EnableIT

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_EnableIT (uint32_t PeriphId)

Function description

Enable the interrupt associated to a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_TZIC_GetFlag

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_GetFlag (uint32_t PeriphId, uint32_t * pFlag)

Function description

Get TZIC flag on a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **pFlag:** Pointer to the flags. If PeriphId target a single peripheral, pointer on a single element. If all peripherals selected (GTZC_PERIPH_ALL), pointer to an array of GTZC_TZIC_PERIPH_NUMBER elements. Element content is either GTZC_TZIC_NO_ILA_EVENT or GTZC_TZSC_ILA_EVENT_PENDING.

Return values

- **HAL:** status

HAL_GTZC_TZIC_ClearFlag

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_ClearFlag (uint32_t PeriphId)

Function description

Clear TZIC flag on a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_IRQHandler

Function name

void HAL_GTZC_IRQHandler (void)

Function description

This function handles GTZC TZIC interrupt request.

Return values

- **None.:**

HAL_GTZC_TZIC_Callback

Function name

void HAL_GTZC_TZIC_Callback (uint32_t PeriphId)

Function description

GTZC TZIC sub-block interrupt callback.

Parameters

- **PeriphId:** Peripheral identifier triggering the illegal access. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values

Return values

- **None.:**

33.3 GTZC Firmware driver defines

The following section lists the various define and macros of the module.

33.3.1 GTZC

GTZC

GTZC Exported Macros

HAL_GTZC_GET_ARRAY_INDEX

GTZC Exported Types

GTZC_MPCBB_NB_VCTR_REG_MAX

< Values needed for

GTZC_MPCBB_NB_LCK_VCTR_REG_MAX

GTZC MPCBB values

GTZC_MPCBB_BLOCK_SIZE

GTZC_MPCBB_SUPERBLOCK_SIZE

GTZC_MPCBB_SUPERBLOCK_UNLOCKED

GTZC_MPCBB_SUPERBLOCK_LOCKED

GTZC_MPCBB_BLOCK_NSEC

GTZC_MPCBB_BLOCK_SEC

GTZC_MPCBB_BLOCK_NPRIV

GTZC_MPCBB_BLOCK_PRIV

GTZC_MPCBB_LOCK_OFF

GTZC_MPCBB_LOCK_ON

GTZC MPCBB INVSECSTATE values

GTZC_MPCBB_INVSECSTATE_NOT_INVERTED

GTZC_MPCBB_INVSECSTATE_INVERTED

GTZC MPCBB SRWILADIS values

GTZC_MPCBB_SRWILADIS_ENABLE

GTZC_MPCBB_SRWILADIS_DISABLE

GTZC MPCWM area identifier values

GTZC_TZSC_MPCWM_ID1

GTZC_TZSC_MPCWM_ID2

GTZC MPCWM Attribute values

GTZC_TZSC_MPCWM_REGION_NSEC

GTZC_TZSC_MPCWM_REGION_SEC

GTZC_TZSC_MPCWM_REGION_NPRIV

GTZC_TZSC_MPCWM_REGION_PRIV

GTZC MPCWM values

GTZC_TZSC_MPCWM_GRANULARITY_1

GTZC_TZSC_MPCWM_GRANULARITY_2

GTZC MPCWM Lock values

GTZC_TZSC_MPCWM_LOCK_OFF

GTZC_TZSC_MPCWM_LOCK_ON

GTZC TZIC flag values

GTZC_TZIC_NO_ILA_EVENT

GTZC_TZIC_ILA_EVENT_PENDING

GTZC TZSC lock values

GTZC_TZSC_LOCK_OFF

GTZC_TZSC_LOCK_ON

GTZC TZSC peripheral attribute values

GTZC_TZSC_PERIPH_SEC
: Secure attribute

GTZC_TZSC_PERIPH_NSEC
: Non-secure attribute

GTZC_TZSC_PERIPH_PRIV
: Privilege attribute

GTZC_TZSC_PERIPH_NPRIV
: Non-privilege attribute

GTZC TZSC and TZIC Peripheral identifier values

GTZC_PERIPH_TIM2

GTZC_PERIPH_TIM3

GTZC_PERIPH_TIM4

GTZC_PERIPH_TIM5

GTZC_PERIPH_TIM6

GTZC_PERIPH_TIM7

GTZC_PERIPH_TIM12

GTZC_PERIPH_TIM13

GTZC_PERIPH_TIM14

GTZC_PERIPH_WWDG

GTZC_PERIPH_IWDG

GTZC_PERIPH_SPI2

GTZC_PERIPH_SPI3

GTZC_PERIPH_USART2

GTZC_PERIPH_USART3

GTZC_PERIPH_UART4

GTZC_PERIPH_UART5

GTZC_PERIPH_I2C1

GTZC_PERIPH_I2C2

GTZC_PERIPH_I3C1

GTZC_PERIPH_CRS

GTZC_PERIPH_USART6

GTZC_PERIPH_USART10

GTZC_PERIPH_USART11

GTZC_PERIPH_HDMICEC

GTZC_PERIPH_DAC1

GTZC_PERIPH_UART7

GTZC_PERIPH_UART8

GTZC_PERIPH_UART9

GTZC_PERIPH_UART12

GTZC_PERIPH_DTS

GTZC_PERIPH_LPTIM2

GTZC_PERIPH_FDCAN1

GTZC_PERIPH_FDCAN2

GTZC_PERIPH_UCPD1

GTZC_PERIPH_TIM1

GTZC_PERIPH_SPI1

GTZC_PERIPH_TIM8

GTZC_PERIPH_USART1

GTZC_PERIPH_TIM15

GTZC_PERIPH_TIM16

GTZC_PERIPH_TIM17

GTZC_PERIPH_SPI4

GTZC_PERIPH_SPI6

GTZC_PERIPH_SAI1

GTZC_PERIPH_SAI2

GTZC_PERIPH_USB

GTZC_PERIPH_SPI5

GTZC_PERIPH_LPUART1

GTZC_PERIPH_I2C3

GTZC_PERIPH_I2C4

GTZC_PERIPH_LPTIM1

GTZC_PERIPH_LPTIM3

GTZC_PERIPH_LPTIM4

GTZC_PERIPH_LPTIM5

GTZC_PERIPH_LPTIM6

GTZC_PERIPH_VREFBUF

GTZC_PERIPH_CRC

GTZC_PERIPH_CORDIC

GTZC_PERIPH_FMAC

GTZC_PERIPH_ETHERNET

GTZC_PERIPH_ICACHE_REG

GTZC_PERIPH_DCACHE1_REG

GTZC_PERIPH_ADC

GTZC_PERIPH_DCMI_PSSI

GTZC_PERIPH_AES

GTZC_PERIPH_HASH

GTZC_PERIPH_RNG

GTZC_PERIPH_PKA

GTZC_PERIPH_SAES

GTZC_PERIPH_SDMMC1

GTZC_PERIPH_SDMMC2

GTZC_PERIPH_FMC_REG

GTZC_PERIPH_OCTOSPI1

GTZC_PERIPH_RAMCFG

GTZC_PERIPH_GPDMA1

GTZC_PERIPH_GPDMA2

GTZC_PERIPH_FLASH

GTZC_PERIPH_FLASH_REG

GTZC_PERIPH_OTFDEC2

GTZC_PERIPH_OTFDEC1

GTZC_PERIPH_SBS

GTZC_PERIPH_RTC

GTZC_PERIPH_TAMP

GTZC_PERIPH_PWR

GTZC_PERIPH_RCC

GTZC_PERIPH_EXTI

GTZC_PERIPH_TZSC

GTZC_PERIPH_TZIC

GTZC_PERIPH_OCTOSPI1_MEM

GTZC_PERIPH_FMC_MEM

GTZC_PERIPH_BKPSRAM

GTZC_PERIPH_SRAM1

GTZC_PERIPH_MPCBB1_REG

GTZC_PERIPH_SRAM2

GTZC_PERIPH_MPCBB2_REG

GTZC_PERIPH_SRAM3

GTZC_PERIPH_MPCBB3_REG

GTZC_PERIPH_ALL

GTZC_TZSC_PERIPH_NUMBER

GTZC_TZIC_PERIPH_NUMBER

34 HAL HASH Generic Driver

34.1 HASH Firmware driver registers structures

34.1.1 HASH_ConfigTypeDef

HASH_ConfigTypeDef is defined in the stm32h5xx_hal_hash.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***
- ***uint32_t Algorithm***

Field Documentation

- ***uint32_t HASH_ConfigTypeDef::DataType***
  no swap (32-bit data), half word swap (16-bit data), byte swap (8-bit data) or bit swap (1-bit data). This parameter can be a value of ***HASH_Data_Type***.
- ***uint32_t HASH_ConfigTypeDef::KeySize***
  The key size is used only in HMAC operation.
- ***uint8_t* HASH_ConfigTypeDef::pKey***
  The key is used only in HMAC operation.
- ***uint32_t HASH_ConfigTypeDef::Algorithm***
  HASH algorithm MD5, SHA1 or SHA2. This parameter can be a value of ***HASH_Algorithm_Selection***

34.1.2 __HASH_HandleTypeDef

__HASH_HandleTypeDef is defined in the stm32h5xx_hal_hash.h

Data Fields

- ***HASH_TypeDef * Instance***
- ***HASH_ConfigTypeDef Init***
- ***uint8_t const * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***__IO uint32_t HashInCount***
- ***uint32_t Size***
- ***uint8_t * pHashKeyBuffPtr***
- ***HAL_HASH_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***__IO HAL_HASH_StateTypeDef State***
- ***__IO uint32_t Accumulation***
- ***void(* InCpltCallback***
- ***void(* DgstCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HASH_TypeDef* __HASH_HandleTypeDef::Instance***
  HASH Register base address
- ***HASH_ConfigTypeDef __HASH_HandleTypeDef::Init***
  HASH required parameters
- ***uint8_t const* __HASH_HandleTypeDef::pHashInBuffPtr***
  Pointer to input buffer

- ***uint8_t* __HASH_HandleTypeDef::pHashOutBuffPtr***
  Pointer to output buffer (digest)
- ***__IO uint32_t __HASH_HandleTypeDef::HashInCount***
  Counter of inputted data
- ***uint32_t __HASH_HandleTypeDef::Size***
  Size of buffer to be processed in bytes
- ***uint8_t* __HASH_HandleTypeDef::pHashKeyBuffPtr***
  Pointer to key buffer (HMAC only)
- ***HAL_HASH_PhaseTypeDef __HASH_HandleTypeDef::Phase***
  HASH peripheral phase
- ***DMA_HandleTypeDef* __HASH_HandleTypeDef::hdmain***
  HASH In DMA Handle parameters
- ***HAL_LockTypeDef __HASH_HandleTypeDef::Lock***
  Locking object
- ***__IO uint32_t __HASH_HandleTypeDef::ErrorCode***
  HASH Error code
- ***__IO HAL_HASH_StateTypeDef __HASH_HandleTypeDef::State***
  HASH peripheral state
- ***__IO uint32_t __HASH_HandleTypeDef::Accumulation***
  HASH multi buffers accumulation flag
- ***void(* __HASH_HandleTypeDef::InCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH input completion callback
- ***void(* __HASH_HandleTypeDef::DgstCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH digest computation complete callback
- ***void(* __HASH_HandleTypeDef::ErrorCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH error callback
- ***void(* __HASH_HandleTypeDef::MspInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp Init callback
- ***void(* __HASH_HandleTypeDef::MspDeInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp DeInit callback

34.2 HASH Firmware driver API description

The following section lists the various functions of the HASH library.

34.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL_HASH_MspInit():
   a. Enable the HASH interface clock using __HAL_RCC_HASH_CLK_ENABLE()
   b. When resorting to interrupt-based APIs (e.g. HAL_HASH_Start_IT())
      - Configure the HASH interrupt priority using HAL_NVIC_SetPriority()
      - Enable the HASH IRQ handler using HAL_NVIC_EnableIRQ()
      - In HASH IRQ handler, call HAL_HASH_IRQHandler() API
   c. When resorting to DMA-based APIs (e.g. HAL_HASH_Start_DMA())
      - Enable the DMA interface clock
      - Configure and enable one DMA to manage data transfer from memory to peripheral (input DMA). Managing data transfer from peripheral to memory can be performed only using CPU.
      - Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA: use HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function:
   a. resorts to HAL_HASH_MspInit() for low-level initialization,
   b. configures the data type: no swap, half word swap, bit swap or byte swap,
   c. configures the Algorithm : MD5, SHA1 or SHA2

3. Three processing schemes are available:
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASH_Start() for HASH or HAL_HMAC_Start() for HMAC
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASH_Start_IT() for HASH or HAL_HMAC_Start_IT() for HMAC
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASH_Start_DMA() for HASH or HAL_HMAC_Start_DMA() for HMAC.
4. When the processing function is called after HAL_HASH_Init(), the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the Peripheral, the digest computation can start.
5. Multi-buffer processing HASH and HMAC are possible in polling, interrupt and DMA modes.
   a. In polling mode, API HAL_HASH_Accumulate()/HAL_HASH_HMAC_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_AccumulateLast()/ HAL_HASH_HMAC_AccumulateLast() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASH_Accumulate_IT()/HAL_HASH_HMAC_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_AccumulateLast_IT()/HAL_HASH_HMAC_AccumulateLast_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASH_Start_DMA() API for HASH and HAL_HASH_HMAC_Start_DMA() API for HMAC . Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASH_Start_DMA()for HASH and HAL_HASH_HMAC_Start_DMA() API for HMAC and retrieve as well the computed digest.
6. To use this driver (version 2.0.0) with application developed with old driver (version 1.0.0) user have to:
   a. Add Algorithm as parameter like DataType or KeySize.
   b. Use new API HAL_HASH_Start() for HASH and HAL_HASH_HMAC_Start() for HMAC processing instead of old API like HAL_HASH_SHA1_Start and HAL_HMAC_SHA1_Start.

34.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH_InitTypeDef and create the associated handle
- DeInitialize the HASH peripheral
- Initialize the HASH MCU Specific Package (MSP)
- DeInitialize the HASH MSP
- Configure HASH (HAL_HASH_SetConfig) with the specified parameters in the HASH_ConfigTypeDef Parameters which are configured in This section are :
- Data Type : no swap, half word swap, bit swap or byte swap
- Algorithm : MD5,SHA1 or SHA2
- Get HASH configuration (HAL_HASH_GetConfig) from the specified parameters in the HASH_HandleTypeDef

This section contains the following APIs:

- ***HAL_HASH_Init()***
- ***HAL_HASH_DeInit()***
- ***HAL_HASH_SetConfig()***
- ***HAL_HASH_GetConfig()***
- ***HAL_HASH_MspInit()***
- ***HAL_HASH_MspDeInit()***
- ***HAL_HASH_RegisterCallback()***
- ***HAL_HASH_UnRegisterCallback()***
- ***HAL_HASH_ProcessSuspend()***

- *HAL_HASH_Resume()*
- *HAL_HASH_Suspend()*

34.2.3 HASH processing functions

This section provides API allowing to calculate the hash value using one of the HASH algorithms supported by the peripheral.

For a single buffer to be hashed, user can resort to one of three processing functions available .

- Polling mode : HAL_HASH_Start()
- Interrupt mode : HAL_HASH_Start_IT()
- DMA mode : HAL_HASH_Start_DMA()

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to :

- Polling mode : HAL_HASH_Accumulate() and wrap-up the digest computation by a call to HAL_HASH_AccumulateLast()
- Interrupt mode : HAL_HASH_Accumulate_IT() and wrap-up the digest computation by a call to HAL_HASH_AccumulateLast_IT()
- DMA mode : HAL_HASH_Start_DMA(), MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro, before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASH_Start_DMA()

This section contains the following APIs:

- *HAL_HASH_Start()*
- *HAL_HASH_Start_IT()*
- *HAL_HASH_Start_DMA()*
- *HAL_HASH_Accumulate()*
- *HAL_HASH_AccumulateLast()*
- *HAL_HASH_Accumulate_IT()*
- *HAL_HASH_AccumulateLast_IT()*

34.2.4 HMAC processing functions

This section provides API allowing to calculate the HMAC (keyed-hash message authentication code) value using:

- one of the algorithms supported by the peripheral
- Key selection
  - Long key : HMAC key is longer than the block size
  - Short key : HMAC key is shorter or equal to the block size

To calculate the HMAC for a single buffer, user can resort to one of three processing functions available .

- Polling mode : HAL_HASH_HMAC_Start()
- Interrupt mode : HAL_HASH_HMAC_Start_IT()
- DMA mode : HAL_HASH_HMAC_Start_DMA()

In case of multi-buffer HMAC processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to :

- Polling mode : HAL_HASH_HMAC_Accumulate() and wrap-up the digest computation by a call to HAL_HASH_HMAC_AccumulateLast()
- Interrupt mode : HAL_HASH_HMAC_Accumulate_IT() and wrap-up the digest computation by a call to HAL_HASH_HMAC_AccumulateLast_IT()
- DMA mode : HAL_HASH_HMAC_Start_DMA(),MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro, before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HMAC processing in feeding the last input buffer through the same API HAL_HASH_HMAC_Start_DMA()

This section contains the following APIs:

- *HAL_HASH_HMAC_Start()*

- *HAL_HASH_HMAC_Accumulate()*
- *HAL_HASH_HMAC_AccumulateLast()*
- *HAL_HASH_HMAC_Start_IT()*
- *HAL_HASH_HMAC_Accumulate_IT()*
- *HAL_HASH_HMAC_AccumulateLast_IT()*
- *HAL_HASH_HMAC_Start_DMA()*

34.2.5 HASH IRQ handler management

This section provides HASH IRQ handler and callback functions.

- HAL_HASH_IRQHandler HASH interrupt request
- HAL_HASH_InCpltCallback input data transfer complete callback
- HAL_HASH_DgstCpltCallback digest computation complete callback
- HAL_HASH_ErrorCallback HASH error callback
- HAL_HASH_GetState return the HASH state
- HAL_HASH_GetError return the HASH error code

This section contains the following APIs:

- *HAL_HASH_IRQHandler()*
- *HAL_HASH_InCpltCallback()*
- *HAL_HASH_DgstCpltCallback()*
- *HAL_HASH_ErrorCallback()*
- *HAL_HASH_GetState()*
- *HAL_HASH_GetError()*

34.2.6 Detailed description of functions

HAL_HASH_Init

Function name

HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)

Function description

Initializes the HASH according to the specified parameters in the HASH_HandleTypeDef and create the associated handle.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **HAL:** status

Notes

- Only Algorithm and DATATYPE bits of HASH Peripheral are set by HAL_HASH_Init(), other configuration bits are set by HASH or HMAC processing APIs.

HAL_HASH_DeInit

Function name

HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH peripheral.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **HAL:** status

HAL_HASH_MspInit

Function name

void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)

Function description

Initialize the HASH MSP.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **None:**

HAL_HASH_MspDeInit

Function name

void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH MSP.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **None:**

HAL_HASH_GetConfig

Function name

HAL_StatusTypeDef HAL_HASH_GetConfig (HASH_HandleTypeDef * hhash, HASH_ConfigTypeDef * pConf)

Function description

Get HASH Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a HASH_HandleTypeDef structure
- **hhash:** pointer to a HASH_ConfigTypeDef structure that contains the configuration information for HASH module

Return values

- **HAL:** status

HAL_HASH_SetConfig

Function name

HAL_StatusTypeDef HAL_HASH_SetConfig (HASH_HandleTypeDef * hhash, HASH_ConfigTypeDef * pConf)

Function description

Configure the HASH according to the specified parameters in the HASH_ConfigTypeDef.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure
- **pConf:** pointer to a HASH_ConfigTypeDef structure that contains the configuration information for HASH module

Return values

- **HAL:** status

HAL_HASH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_RegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID, pHASH_CallbackTypeDef pCallback)

Function description

Register a User HASH Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_HASH_INPUTCPLT_CB_ID input completion callback ID
  - HAL_HASH_DGSTCPLT_CB_ID digest computation completion callback ID
  - HAL_HASH_ERROR_CB_ID error callback ID
  - HAL_HASH_MSPINIT_CB_ID MspInit callback ID
  - HAL_HASH_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_HASH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_UnRegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID)

Function description

Unregister a HASH Callback HASH Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_HASH_INPUTCPLT_CB_ID HASH input completion Callback ID
  - HAL_HASH_DGSTCPLT_CB_ID HASH digest computation completion Callback ID
  - HAL_HASH_ERROR_CB_ID HASH error Callback ID
  - HAL_HASH_MSPINIT_CB_ID HASH MspInit callback ID
  - HAL_HASH_MSPDEINIT_CB_ID HASH MspDeInit callback ID

Return values

- **status:**

HAL_HASH_ProcessSuspend

Function name

HAL_StatusTypeDef HAL_HASH_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

HAL_HASH_Resume

Function name

void HAL_HASH_Resume (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

HAL_HASH_Suspend

Function name

void HAL_HASH_Suspend (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

HAL_HASH_Start

Function name

HAL_StatusTypeDef HAL_HASH_Start (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HASH peripheral processes in polling mode pInBuffer then reads the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** specify timeout value

Return values

- **HAL:** status

HAL_HASH_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_Start_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HASH peripheral processes in interrupt mode pInBuffer then reads the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

HAL_HASH_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_Start_DMA (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HASH peripheral processes in DMA mode pInBuffer then reads the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes (must be a multiple of 4 in case of Multi-buffer and not last buffer).
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

Notes

- Multi-buffer HASH processing is possible, consecutive calls to HAL_HASH_Start_DMA (MDMAT bit must be set) can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_Start_DMA (MDMAT bit must be reset).

HAL_HASH_Accumulate

Function name

HAL_StatusTypeDef HAL_HASH_Accumulate (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint32_t Timeout)

Function description

HASH peripheral processes in polling mode several input buffers.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes and a multiple of 4.
- **Timeout:** specify timeout value

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_Accumulate() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_AccumulateLast()

HAL_HASH_AccumulateLast

Function name

HAL_StatusTypeDef HAL_HASH_AccumulateLast (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_Accumulate() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** specify timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer

HAL_HASH_AccumulateLast_IT

Function name

HAL_StatusTypeDef HAL_HASH_AccumulateLast_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_Accumulate_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer

HAL_HASH_Accumulate_IT

Function name

HAL_StatusTypeDef HAL_HASH_Accumulate_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size)

Function description

HASH peripheral processes in interrupt mode several input buffers.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes and a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_Accumulate_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_AccumulateLast_IT()

HAL_HASH_HMAC_Start

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

HMAC in polling mode, HASH peripheral processes Key then pInBuffer then reads the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** specify timeout value

Return values

- **HAL:** status

HAL_HASH_HMAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start_DMA (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HMAC in DMA mode,HASH peripheral processes Key then pInBuffer in DMA mode then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

Notes

- Multi-buffer HMAC processing is possible, consecutive calls to HAL_HASH_HMAC_Start_DMA (MDMAT bit must be set) can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_HMAC_Start_DMA (MDMAT bit must be reset).

HAL_HASH_HMAC_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Start_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

HMAC in interrupt mode, HASH peripheral process Key then pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

HAL_HASH_HMAC_Accumulate

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Accumulate (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint32_t Timeout)

Function description

HMAC accumulate mode, HASH peripheral processes Key then several input buffers.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes and a multiple of 4
- **Timeout:** specify timeout value

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_HMAC_Accumulate() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_HMAC_AccumulateLast()

HAL_HASH_HMAC_AccumulateLast

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_AccumulateLast (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HMAC signature after several calls to HAL_HASH_HMAC_Accumulate() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** specify timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer

HAL_HASH_HMAC_Accumulate_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_Accumulate_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size)

Function description

HMAC accumulate in interrupt mode, HASH peripheral processes Key then several input buffers.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes and a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_HMAC_Accumulate_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_HMAC_AccumulateLast_IT()

HAL_HASH_HMAC_AccumulateLast_IT

Function name

HAL_StatusTypeDef HAL_HASH_HMAC_AccumulateLast_IT (HASH_HandleTypeDef * hhash, const uint8_t *const pInBuffer, uint32_t Size, uint8_t *const pOutBuffer)

Function description

End computation of a single HMAC signature in interrupt mode, after several calls to HAL_HASH_HMAC_Accumulate() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer

HAL_HASH_IRQHandler

Function name

void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)

Function description

Handle HASH interrupt request.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_IRQHandler() handles interrupts in HMAC processing as well.

HAL_HASH_InCpltCallback

Function name

void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Input data transfer complete call back.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **None:**

Notes

- HAL_HASH_InCpltCallback() is called when the complete input message has been fed to the Peripheral. This API is invoked only when input data are entered under interruption or through DMA.
- In case of HASH or HMAC multi-buffer DMA feeding case (MDMAT bit set), HAL_HASH_InCpltCallback() is called at the end of each buffer feeding to the Peripheral.

HAL_HASH_DgstCpltCallback

Function name

void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Digest computation complete call back.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **None:**

Notes

- HAL_HASH_DgstCpltCallback() is used under interruption, is not relevant with DMA.

HAL_HASH_ErrorCallback

Function name

void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)

Function description

HASH error callback.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module.

Return values

- **None:**

Notes

- Code user can resort to hhash->Status (HAL_ERROR, HAL_TIMEOUT,...) to retrieve the error type.

HAL_HASH_GetState

Function name

HAL_HASH_StateTypeDef HAL_HASH_GetState (const HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle state.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** HASH state

Notes

- The API yields the current state of the handle (BUSY, READY,...).

HAL_HASH_GetError

Function name

uint32_t HAL_HASH_GetError (const HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle error code.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure.

Return values

- **HASH:** Error Code

34.3 HASH Firmware driver defines

The following section lists the various define and macros of the module.

34.3.1 HASH

HASH

HASH algorithm selection

HASH_ALGOSELECTION_SHA1

HASH function is SHA1

HASH_ALGOSELECTION_SHA224

HASH function is SHA224

HASH_ALGOSELECTION_SHA256

HASH function is SHA256

HASH_ALGOSELECTION_SHA384

HASH function is SHA384

HASH_ALGOSELECTION_SHA512_224

HASH function is SHA512_224

HASH_ALGOSELECTION_SHA512_256

HASH function is SHA512_256

HASH_ALGOSELECTION_SHA512

HASH function is SHA512

HASH Data Type

HASH_NO_SWAP

32-bit data. No swapping

HASH_HALFWORD_SWAP

16-bit data. Each half word is swapped

HASH_BYTE_SWAP

8-bit data. All bytes are swapped

HASH_BIT_SWAP

1-bit data. In the word all bits are swapped

HASH Error Definition

HAL_HASH_ERROR_NONE

No error

HAL_HASH_ERROR_BUSY

Busy flag error

HAL_HASH_ERROR_DMA

DMA-based process error

HAL_HASH_ERROR_TIMEOUT

Timeout error

HAL_HASH_ERROR_INVALID_CALLBACK

Invalid Callback error

HASH Exported Macros

__HAL_HASH_GET_FLAG

Description:

- Check whether or not the specified HASH flag is set.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete.
  - HASH_FLAG_DMAS DMA interface is enabled (DMAE=1) or a transfer is ongoing.
  - HASH_FLAG_BUSY The hash core is Busy : processing a block of data.
  - HASH_FLAG_DINNE DIN not empty : the input buffer contains at least one word of data.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_HASH_CLEAR_FLAG

Description:

- Clear the specified HASH flag.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete

Return value:

- None

__HAL_HASH_GET_IT_SOURCE

Description:

- Check whether the specified HASH interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: HASH interrupt source to check This parameter can be one of the following values :
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_HASH_ENABLE_IT

Description:

- Enable the specified HASH interrupt.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_DISABLE_IT

Description:

- Disable the specified HASH interrupt.

Parameters:

- __HANDLE__: specifies the HASH handle.
- __INTERRUPT__: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATE

Description:

- Reset HASH handle state.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_SET_MDMAT

Description:

- Enable the multi-buffer DMA transfer mode.

Return value:

- None

Notes:

- This bit is set when hashing large files when multiple DMA transfers are needed.

__HAL_HASH_RESET_MDMAT

Description:

- Disable the multi-buffer DMA transfer mode.

Return value:

- None

HAL_HASH_VERSION

Description:

- HAL HASH driver version.

Return value:

- None: HAL HASH driver version 2.0.0

HASH flags definitions

HASH_FLAG_DINIS

16 locations are free in the DIN : new block can be entered in the Peripheral

HASH_FLAG_DCIS

Digest calculation complete

HASH_FLAG_DMAS

DMA interface is enabled (DMAE=1) or a transfer is ongoing

HASH_FLAG_BUSY

The hash core is Busy, processing a block of data

HASH_FLAG_DINNE

DIN not empty : input buffer contains at least one word of data

key length only for HMAC mode

HASH_SHORTKEY

HMAC Key size is <= block size (64 or 128 bytes)

HASH_LONGKEY

HMAC Key size is > block size (64 or 128 bytes)

HASH interrupts definitions

HASH_IT_DINI

A new block can be entered into the input buffer (DIN)

HASH_IT_DCI

Digest calculation complete

HASH Mode

HASH_ALGOMODE_HASH

HASH mode

HASH_ALGOMODE_HMAC

HMAC mode

HASH Number of Context Swap Registers

HASH_NUMBER_OF_CSR_REGISTERS

Number of Context Swap Registers

35 HAL HCD Generic Driver

35.1 HCD Firmware driver registers structures

35.1.1 __HCD_HandleTypeDef

__HCD_HandleTypeDef is defined in the stm32h5xx_hal_hcd.h

Data Fields

- ***HCD_TypeDef * Instance***
- ***HCD_InitTypeDef Init***
- ***HCD_HCTypeDef hc***
- ***uint32_t ep0_PmaAllocState***
- ***uint16_t phy_chin_state***
- ***uint16_t phy_chout_state***
- ***uint32_t PMALookupTable***
- ***HCD_HostStateTypeDef HostState***
- ***HAL_LockTypeDef Lock***
- ***__IO HCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* PortEnabledCallback***
- ***void(* PortDisabledCallback***
- ***void(* HC_NotifyURBChangeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HCD_TypeDef* __HCD_HandleTypeDef::Instance***
  Register base address
- ***HCD_InitTypeDef __HCD_HandleTypeDef::Init***
  HCD required parameters
- ***HCD_HCTypeDef __HCD_HandleTypeDef::hc[16]***
  Host channels parameters
- ***uint32_t __HCD_HandleTypeDef::ep0_PmaAllocState***
  EP0 PMA allocation State (allocated, virtual Ch, EP0 direction)
- ***uint16_t __HCD_HandleTypeDef::phy_chin_state[8]***
  Physical Channel in State (Used/Free)
- ***uint16_t __HCD_HandleTypeDef::phy_chout_state[8]***
  Physical Channel out State (Used/Free)
- ***uint32_t __HCD_HandleTypeDef::PMALookupTable[PMA_BLOCKS]***
- ***HCD_HostStateTypeDef __HCD_HandleTypeDef::HostState***
  USB current state DICONNECT/CONNECT/RUN/SUSPEND/RESUME
- ***HAL_LockTypeDef __HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef __HCD_HandleTypeDef::State***
  HCD communication state
- ***__IO uint32_t __HCD_HandleTypeDef::ErrorCode***
  HCD Error code

- ***void* __HCD_HandleTypeDef::pData***
  Pointer Stack Handler
- ***void(* __HCD_HandleTypeDef::SOFCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD SOF callback
- ***void(* __HCD_HandleTypeDef::ConnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Connect callback
- ***void(* __HCD_HandleTypeDef::DisconnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Disconnect callback
- ***void(* __HCD_HandleTypeDef::PortEnabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Enable callback
- ***void(* __HCD_HandleTypeDef::PortDisabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Disable callback
- ***void(* __HCD_HandleTypeDef::HC_NotifyURBChangeCallback)(struct __HCD_HandleTypeDef *hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)***
  USB OTG HCD Host Channel Notify URB Change callback
- ***void(* __HCD_HandleTypeDef::MspInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp Init callback
- ***void(* __HCD_HandleTypeDef::MspDeInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp DeInit callback

35.2 HCD Firmware driver API description

The following section lists the various functions of the HCD library.

35.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macros
      ◦ __HAL_RCC_USB_CLK_ENABLE();
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

35.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_HCD_Init()***
- ***HAL_HCD_HC_Init()***
- ***HAL_HCD_HC_Close()***
- ***HAL_HCD_HC_Halt()***
- ***HAL_HCD_DeInit()***
- ***HAL_HCD_MspInit()***
- ***HAL_HCD_MspDeInit()***
- ***HAL_HCD_SuspendCallback()***
- ***HAL_HCD_ResumeCallback()***
- ***HAL_HCD_RegisterCallback()***
- ***HAL_HCD_UnRegisterCallback()***

- *HAL_HCD_RegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()*

35.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer
This section contains the following APIs:

- *HAL_HCD_HC_SubmitRequest()*
- *HAL_HCD_IRQHandler()*
- *HAL_HCD_SOF_Callback()*
- *HAL_HCD_Connect_Callback()*
- *HAL_HCD_Disconnect_Callback()*
- *HAL_HCD_PortEnabled_Callback()*
- *HAL_HCD_PortDisabled_Callback()*
- *HAL_HCD_HC_NotifyURBChange_Callback()*
- *HAL_HCD_RegisterCallback()*
- *HAL_HCD_UnRegisterCallback()*
- *HAL_HCD_RegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_HC_SetHubInfo()*
- *HAL_HCD_HC_ClearHubInfo()*
- *HAL_HCD_SuspendCallback()*
- *HAL_HCD_ResumeCallback()*

35.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.
This section contains the following APIs:

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_Suspend()*
- *HAL_HCD_Resume()*
- *HAL_HCD_ResetPort()*
- *HAL_HCD_ResumePort()*

35.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.
This section contains the following APIs:

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*
- *HAL_HCD_HC_SetHubInfo()*
- *HAL_HCD_HC_ClearHubInfo()*
- *HAL_HCD_PMAlloc()*
- *HAL_HCD_PMADeAlloc()*
- *HAL_HCD_PMAReset()*

35.2.6 Detailed description of functions

HAL_HCD_Init

Function name

HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)

Function description

Initialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_DeInit

Function name

HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_Init

Function name

HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)

Function description

Initialize a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **epnum:** Endpoint number. This parameter can be a value from 1 to 15
- **dev_address:** Current device address This parameter can be a value from 0 to 255
- **speed:** Current device speed. This parameter can be one of these values: HCD_DEVICE_SPEED_HIGH High speed mode, HCD_DEVICE_SPEED_FULL Full speed mode, HCD_DEVICE_SPEED_LOW Low speed mode
- **ep_type:** Endpoint Type. This parameter can be one of these values: USBH_EP_CONTROL Control type, USBH_EP_ISO Isochronous type, USBH_EP_BULK Bulk type, USBH_EP_INTERRUPT Interrupt type
- **mps:** Max Packet Size. This parameter can be a value from 0 to32K

Return values

- **HAL:** status

HAL_HCD_HC_Halt

Function name

HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Halt a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_HC_Close

Function name

HAL_StatusTypeDef HAL_HCD_HC_Close (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

HAL_HCD_HC_Close Pipe.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_MspInit

Function name

void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)

Function description

Initialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_MspDeInit

Function name

void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID, pHCD_CallbackTypeDef pCallback)

Function description

Register a User USB HCD Callback To be used instead of the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enable callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disable callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB HCD Callback USB HCD callback is redirected to the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID DRD HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enabled callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disabled callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_HCD_RegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd, pHCD_HC_NotifyURBChangeCallbackTypeDef pCallback)

Function description

Register USB HCD Host Channel Notify URB Change Callback To be used instead of the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle
- **pCallback:** pointer to the USB HCD Host Channel Notify URB Change Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd)

Function description

Unregister the USB HCD Host Channel Notify URB Change Callback USB HCD Host Channel Notify URB Change Callback is redirected to the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_SubmitRequest

Function name

HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)

Function description

Submit a new URB for processing.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **direction:** Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep_type:** Endpoint Type. This parameter can be one of these values: USBH_EP_CONTROL : Control type/ USBH_EP_ISO : Isochronous type/ USBH_EP_BULK : Bulk type/ USBH_EP_INTERRUPT : Interrupt type/
- **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1
- **pbuff:** pointer to URB data
- **length:** Length of URB data
- **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

Return values

- **HAL:** status

HAL_HCD_HC_SetHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_SetHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t addr, uint8_t PortNbr)

Function description

Set host channel Hub Information.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 8
- **addr:** Hub address
- **PortNbr:** Hub port number

Return values

- **HAL:** status

HAL_HCD_HC_ClearHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_ClearHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Clear host channel hub information.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 8

Return values

- **HAL:** status

HAL_HCD_IRQHandler

Function name

void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)

Function description

Handle HCD interrupt request.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_SOF_Callback

Function name

void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)

Function description

SOF callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Connect_Callback

Function name

void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Connection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Disconnect_Callback

Function name

void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Disconnection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortEnabled_Callback

Function name

void HAL_HCD_PortEnabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Enabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortDisabled_Callback

Function name

void HAL_HCD_PortDisabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Disabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_SuspendCallback

Function name

void HAL_HCD_SuspendCallback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_ResumeCallback

Function name

void HAL_HCD_ResumeCallback (HCD_HandleTypeDef * hhcd)

Function description

HAL_HCD_HC_NotifyURBChange_Callback

Function name

void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)

Function description

Notify URB state change callback.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15
- **urb_state:** This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

Return values

- **None:**

HAL_HCD_ResetPort

Function name

HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)

Function description

Reset the host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Start

Function name

HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)

Function description

Start the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Stop

Function name

HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)

Function description

Stop the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Suspend

Function name

HAL_StatusTypeDef HAL_HCD_Suspend (HCD_HandleTypeDef * hhcd)

Function description

Put the Device in suspend mode.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Resume

Function name

HAL_StatusTypeDef HAL_HCD_Resume (HCD_HandleTypeDef * hhcd)

Function description

Resume host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_ResumePort

Function name

HAL_StatusTypeDef HAL_HCD_ResumePort (HCD_HandleTypeDef * hhcd)

Function description

Resme the host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_GetState

Function name

HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef const * hhcd)

Function description

Return the HCD handle state.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** state

HAL_HCD_HC_GetURBState

Function name

HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return URB state for a channel.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL

HAL_HCD_HC_GetState

Function name

HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return the Host Channel state.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **Host:** channel state This parameter can be one of these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR

HAL_HCD_HC_GetXferCount

Function name

uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return the last host transfer size.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **last:** transfer size in byte

HAL_HCD_GetCurrentFrame

Function name

uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)

Function description

Return the current Host frame number.

Parameters

- **hhcd:** HCD handle

Return values

- **Current:** Host frame number

HAL_HCD_GetCurrentSpeed

Function name

uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)

Function description

Return the Host enumeration speed.

Parameters

- **hhcd:** HCD handle

Return values

- **speed:** : Device speed after Host enumeration This parameter can be one of these values:
  - HCD_DEVICE_SPEED_FULL: Full speed mode
  - HCD_DEVICE_SPEED_LOW: Low speed mode

HAL_HCD_PMAlloc

Function name

HAL_StatusTypeDef HAL_HCD_PMAlloc (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint16_t ch_kind, uint16_t mps)

Function description

Allocate PMA buffer for Channel This API will fetch a free space.

Parameters

- **hhcd:** Host instance
- **ch_num:** Channel number
- **ch_kind:** endpoint Kind USB_SNG_BUF Single Buffer used USB_DBL_BUF Double Buffer used
- **mps:** Channel Max Packet Size

Return values

- **HAL:** status

HAL_HCD_PMADeAlloc

Function name

HAL_StatusTypeDef HAL_HCD_PMADeAlloc (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

PMA De-Allocation for Channel Free the reserved block in the PMA-LookupTable.

Parameters

- **hhcd:** Host instance
- **ch_num:** Channel number

Return values

- **HAL:** status

HAL_HCD_PMAReset

Function name

HAL_StatusTypeDef HAL_HCD_PMAReset (HCD_HandleTypeDef * hhcd)

Function description

PMA Reset.

Parameters

- **hhcd:** Host instance

Return values

- **HAL:** status

HCD_GET_CH_RX_CNT

Function name

__STATIC_INLINE uint16_t HCD_GET_CH_RX_CNT (HCD_TypeDef * Instance, uint16_t bChNum)

Function description

gets counter of the rx buffer.

Parameters

- **Instance:** USB peripheral instance register address.
- **bChNum:** channel Number.

Return values

- **Counter:** value

HCD_GET_CH_DBUF0_CNT

Function name

__STATIC_INLINE uint16_t HCD_GET_CH_DBUF0_CNT (const HCD_TypeDef * Instance, uint16_t bChNum)

Function description

gets counter of the rx buffer0.

Parameters

- **Instance:** USB peripheral instance register address.
- **bChNum:** channel Number.

Return values

- **Counter:** value

HCD_GET_CH_DBUF1_CNT

Function name

__STATIC_INLINE uint16_t HCD_GET_CH_DBUF1_CNT (const HCD_TypeDef * Instance, uint16_t bChNum)

Function description

gets counter of the rx buffer1.

Parameters

- **Instance:** USB peripheral instance register address.
- **bChNum:** channel Number.

Return values

- **Counter:** value

35.3 HCD Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 HCD

HCD

HCD Device Speed

HCD_DEVICE_SPEED_HIGH

HCD_DEVICE_SPEED_FULL

HCD_DEVICE_SPEED_LOW

HCD Endpoint Kind

HCD_SNG_BUF

HCD_DBL_BUF

HCD Error Code definition

HAL_HCD_ERROR_INVALID_CALLBACK

Invalid Callback error

HCD Exported Macros

__HAL_HCD_ENABLE

__HAL_HCD_DISABLE

__HAL_HCD_GET_FLAG

__HAL_HCD_CLEAR_FLAG

__HAL_HCD_IS_INVALID_INTERRUPT

__HAL_HCD_GET_CHNUM

__HAL_HCD_GET_CHDIR

HCD State Structure definition

PMA_BLOCKS

HCD Logical Channel

HCD_LOGICAL_CH_NOT_OPENED

HCD_FREE_CH_NOT_FOUND

HCD PHY Module

HCD_PHY_ULPI

HCD_PHY_EMBEDDED

HCD Speed

HCD_SPEED_FULL

HCD_SPEED_LOW

36 HAL I2C Generic Driver

36.1 I2C Firmware driver registers structures

36.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32h5xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

36.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32h5xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_StatusTypeDef(* XferISR***
- ***DMA_HandleTypeDef * hdmatx***

- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO HAL_I2C_ModeTypeDef Mode***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t AddrEventCount***
- ***__IO uint32_t Devaddress***
- ***__IO uint32_t Memaddress***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* MemTxCpltCallback***
- ***void(* MemRxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of ***I2C_XFEROPTIONS***
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

- ***__IO uint32_t __I2C_HandleTypeDef::Devaddress***
  I2C Target device address
- ***__IO uint32_t __I2C_HandleTypeDef::Memaddress***
  I2C Target memory address
- ***void(* __I2C_HandleTypeDef::MasterTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MasterRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ListenCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Listen Complete callback
- ***void(* __I2C_HandleTypeDef::MemTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MemRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ErrorCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Error callback
- ***void(* __I2C_HandleTypeDef::AbortCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Abort callback
- ***void(* __I2C_HandleTypeDef::AddrCallback)(struct __I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  I2C Slave Address Match callback
- ***void(* __I2C_HandleTypeDef::MspInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp Init callback
- ***void(* __I2C_HandleTypeDef::MspDeInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp DeInit callback

36.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

36.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx channel
      - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel

3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode or DMA mode IO sequential operation

Note: These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer

- Option field values are defined through I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functional is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Seq_Transmit_IT() then HAL_I2C_Master_Seq_Transmit_IT() or HAL_I2C_Master_Seq_Transmit_DMA() then HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME. Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the opposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_FRAME then I2C_OTHER_FRAME. Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Different sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Transmit_IT() or using HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Receive_IT() or using HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
  - Abort a master IT or DMA I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Transmit_IT() or using HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Receive_IT() or using HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
  - Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()

- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Callback registration

The compilation flag USE_HAL_I2C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2C_RegisterCallback() or HAL_I2C_RegisterAddrCallback() to register an interrupt callback.

Function HAL_I2C_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.

- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_I2C_RegisterAddrCallback().

Use function HAL_I2C_UnRegisterCallback to reset a callback to the default weak function. HAL_I2C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_I2C_UnRegisterAddrCallback().

By default, after the HAL_I2C_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2C_MasterTxCpltCallback(), HAL_I2C_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2C_Init()/ HAL_I2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2C_Init()/ HAL_I2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2C_RegisterCallback() before calling HAL_I2C_DeInit() or HAL_I2C_Init() function.

When the compilation flag USE_HAL_I2C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I2C HAL driver header file for more useful macros

36.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- *HAL_I2C_Init()*
- *HAL_I2C_DeInit()*
- *HAL_I2C_MspInit()*
- *HAL_I2C_MspDeInit()*
- *HAL_I2C_RegisterCallback()*
- *HAL_I2C_UnRegisterCallback()*
- *HAL_I2C_RegisterAddrCallback()*
- *HAL_I2C_UnRegisterAddrCallback()*

36.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2C_Master_Transmit()
   - HAL_I2C_Master_Receive()
   - HAL_I2C_Slave_Transmit()
   - HAL_I2C_Slave_Receive()
   - HAL_I2C_Mem_Write()
   - HAL_I2C_Mem_Read()
   - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2C_Master_Transmit_IT()
   - HAL_I2C_Master_Receive_IT()
   - HAL_I2C_Slave_Transmit_IT()
   - HAL_I2C_Slave_Receive_IT()
   - HAL_I2C_Mem_Write_IT()
   - HAL_I2C_Mem_Read_IT()
   - HAL_I2C_Master_Seq_Transmit_IT()
   - HAL_I2C_Master_Seq_Receive_IT()
   - HAL_I2C_Slave_Seq_Transmit_IT()
   - HAL_I2C_Slave_Seq_Receive_IT()
   - HAL_I2C_EnableListen_IT()
   - HAL_I2C_DisableListen_IT()
   - HAL_I2C_Master_Abort_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2C_Master_Transmit_DMA()
   - HAL_I2C_Master_Receive_DMA()
   - HAL_I2C_Slave_Transmit_DMA()
   - HAL_I2C_Slave_Receive_DMA()
   - HAL_I2C_Mem_Write_DMA()
   - HAL_I2C_Mem_Read_DMA()
   - HAL_I2C_Master_Seq_Transmit_DMA()
   - HAL_I2C_Master_Seq_Receive_DMA()
   - HAL_I2C_Slave_Seq_Transmit_DMA()
   - HAL_I2C_Slave_Seq_Receive_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2C_MasterTxCpltCallback()
   - HAL_I2C_MasterRxCpltCallback()
   - HAL_I2C_SlaveTxCpltCallback()
   - HAL_I2C_SlaveRxCpltCallback()
   - HAL_I2C_MemTxCpltCallback()
   - HAL_I2C_MemRxCpltCallback()
   - HAL_I2C_AddrCallback()
   - HAL_I2C_ListenCpltCallback()
   - HAL_I2C_ErrorCallback()
   - HAL_I2C_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_I2C_Master_Transmit()***
- ***HAL_I2C_Master_Receive()***
- ***HAL_I2C_Slave_Transmit()***
- ***HAL_I2C_Slave_Receive()***
- ***HAL_I2C_Master_Transmit_IT()***
- ***HAL_I2C_Master_Receive_IT()***
- ***HAL_I2C_Slave_Transmit_IT()***
- ***HAL_I2C_Slave_Receive_IT()***
- ***HAL_I2C_Master_Transmit_DMA()***
- ***HAL_I2C_Master_Receive_DMA()***
- ***HAL_I2C_Slave_Transmit_DMA()***
- ***HAL_I2C_Slave_Receive_DMA()***
- ***HAL_I2C_Mem_Write()***
- ***HAL_I2C_Mem_Read()***
- ***HAL_I2C_Mem_Write_IT()***
- ***HAL_I2C_Mem_Read_IT()***
- ***HAL_I2C_Mem_Write_DMA()***
- ***HAL_I2C_Mem_Read_DMA()***
- ***HAL_I2C_IsDeviceReady()***
- ***HAL_I2C_Master_Seq_Transmit_IT()***
- ***HAL_I2C_Master_Seq_Transmit_DMA()***
- ***HAL_I2C_Master_Seq_Receive_IT()***
- ***HAL_I2C_Master_Seq_Receive_DMA()***
- ***HAL_I2C_Slave_Seq_Transmit_IT()***
- ***HAL_I2C_Slave_Seq_Transmit_DMA()***
- ***HAL_I2C_Slave_Seq_Receive_IT()***
- ***HAL_I2C_Slave_Seq_Receive_DMA()***
- ***HAL_I2C_EnableListen_IT()***
- ***HAL_I2C_DisableListen_IT()***
- ***HAL_I2C_Master_Abort_IT()***

36.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_I2C_GetState()***
- ***HAL_I2C_GetMode()***
- ***HAL_I2C_GetError()***

36.2.5 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C peripheral.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

Initialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MspDeInit

Function name

void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID, pI2C_CallbackTypeDef pCallback)

Function description

Register a User I2C Callback To be used instead of the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  - HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  - HAL_I2C_ERROR_CB_ID Error callback ID
  - HAL_I2C_ABORT_CB_ID Abort callback ID
  - HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  - HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_I2C_RegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2C Callback I2C callback is redirected to the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  - HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  - HAL_I2C_ERROR_CB_ID Error callback ID
  - HAL_I2C_ABORT_CB_ID Abort callback ID
  - HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  - HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_I2C_UnRegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to un-register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterAddrCallback (I2C_HandleTypeDef * hi2c, pI2C_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match I2C Callback To be used instead of the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_I2C_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterAddrCallback (I2C_HandleTypeDef * hi2c)

Function description

UnRegister the Slave Address Match I2C Callback Info Ready I2C Callback is redirected to the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Write

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Write an amount of data in blocking mode to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Checks if target device is ready for communication.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C

Return values

- **HAL:** status

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

Abort a master I2C IT or DMA process communication with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Reads an amount of data in non-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EV_IRQHandler

Function name

void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C event interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ER_IRQHandler

Function name

void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C error interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C Transfer Direction Master Point of View
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Listen Complete callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemTxCpltCallback

Function name

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemRxCpltCallback

Function name

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ErrorCallback

Function name

void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C error callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AbortCpltCallback

Function name

void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C abort callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_GetState

Function name

HAL_I2C_StateTypeDef HAL_I2C_GetState (const I2C_HandleTypeDef * hi2c)

Function description

Return the I2C handle state.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** state

HAL_I2C_GetMode

Function name

HAL_I2C_ModeTypeDef HAL_I2C_GetMode (const I2C_HandleTypeDef * hi2c)

Function description

Returns the I2C Master, Slave, Memory or no mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values

- **HAL:** mode

HAL_I2C_GetError

Function name

uint32_t HAL_I2C_GetError (const I2C_HandleTypeDef * hi2c)

Function description

Return the I2C error code.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **I2C:** Error Code

36.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

36.3.1 I2C

I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE

No error

HAL_I2C_ERROR_BERR

BERR error

HAL_I2C_ERROR_ARLO

ARLO error

HAL_I2C_ERROR_AF

ACKF error

HAL_I2C_ERROR_OVR

OVR error

HAL_I2C_ERROR_DMA

DMA transfer error

HAL_I2C_ERROR_TIMEOUT

Timeout error

HAL_I2C_ERROR_SIZE

Size Management error

HAL_I2C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I2C_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_I2C_ERROR_INVALID_PARAM

Invalid Parameters error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

I2C_FLAG_MASK

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert
  - I2C_FLAG_BUSY Bus busy
  - I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_GET_FLAG

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

I2C_LAST_FRAME_NO_STOP

I2C_OTHER_FRAME

I2C_OTHER_AND_LAST_FRAME

37 HAL I2C Extension Driver

37.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

37.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32H5xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

37.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   - HAL_I2CEx_ConfigFastModePlus()

37.1.3 Filter Mode Functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter()***
- ***HAL_I2CEx_ConfigDigitalFilter()***

37.1.4 WakeUp Mode Functions

This section provides functions allowing to:

- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_I2CEx_EnableWakeUp()***
- ***HAL_I2CEx_DisableWakeUp()***

37.1.5 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_I2CEx_ConfigFastModePlus()***

37.1.6 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)

Function description

Configure I2C Analog noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** New state of the Analog filter.

Return values

- **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

Configure I2C Digital noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Enable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Disable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigFastModePlus (I2C_HandleTypeDef * hi2c, uint32_t FastModePlus)

Function description

Configure I2C Fast Mode Plus.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **FastModePlus:** New state of the Fast Mode Plus.

Return values

- **HAL:** status

37.2 I2CEx Firmware driver defines

The following section lists the various define and macros of the module.

37.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FASTMODEPLUS_ENABLE

Enable Fast Mode Plus

I2C_FASTMODEPLUS_DISABLE

Disable Fast Mode Plus

38 HAL I2S Generic Driver

38.1 I2S Firmware driver registers structures

38.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32h5xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***
- ***uint32_t FirstBit***
- ***uint32_t WSInversion***
- ***uint32_t Data24BitAlignment***
- ***uint32_t MasterKeepIOState***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***
- ***uint32_t I2S_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***I2S_MSB_LSB_Transmission***
- ***uint32_t I2S_InitTypeDef::WSInversion***
  Control the Word Select Inversion. This parameter can be a value of ***I2S_WSInversion***
- ***uint32_t I2S_InitTypeDef::Data24BitAlignment***
  Specifies the Data Padding for 24 bits data length This parameter can be a value of ***I2S_Data_24Bit_Alignment***
- ***uint32_t I2S_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***I2S_Master_Keep_IO_State***

38.1.2 __I2S_HandleTypeDef

__I2S_HandleTypeDef is defined in the stm32h5xx_hal_i2s.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***I2S_InitTypeDef Init***
- ***const uint16_t * pTxBuffPtr***
- ***__IO uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***

- *uint16_t * pRxBuffPtr*
- *__IO uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *void(* RxISR*
- *void(* TxISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *__IO HAL_LockTypeDef Lock*
- *__IO HAL_I2S_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *void(* TxCpltCallback*
- *void(* RxCpltCallback*
- *void(* TxRxCpltCallback*
- *void(* TxHalfCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* TxRxHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SPI_TypeDef* __I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef __I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***const uint16_t* __I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* __I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***void(* __I2S_HandleTypeDef::RxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Rx ISR
- ***void(* __I2S_HandleTypeDef::TxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t __I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of *I2S_Error*
- ***void(* __I2S_HandleTypeDef::TxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Completed callback

- ***void(* __I2S_HandleTypeDef::RxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Completed callback
- ***void(* __I2S_HandleTypeDef::TxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Half Completed callback
- ***void(* __I2S_HandleTypeDef::RxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Half Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Half Completed callback
- ***void(* __I2S_HandleTypeDef::ErrorCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Error callback
- ***void(* __I2S_HandleTypeDef::MspInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp Init callback
- ***void(* __I2S_HandleTypeDef::MspDeInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp DeInit callback

38.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

38.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      - Enable the clock for the I2S GPIOs.
      - Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      - Configure the I2Sx interrupt priority.
      - Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      - Declare a DMA handle structure for the Tx/Rx Stream/Channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream/Channel.
      - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream/Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

– *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32h5xx_hal_conf.h file.*

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()

- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

Note: You can refer to the I2S HAL driver header file for more useful macros

I2S HAL driver macros list

Callback registration:

1. The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks: (+) TxCpltCallback : I2S Tx Completed callback (+) RxCpltCallback : I2S Rx Completed callback (+) TxRxCpltCallback : I2S TxRx Completed callback (+) TxHalfCpltCallback : I2S Tx Half Completed callback (+) RxHalfCpltCallback : I2S Rx Half Completed callback (+) TxRxHalfCpltCallback : I2S TxRx Half Completed callback (+) ErrorCallback : I2S Error callback (+) MspInitCallback : I2S Msp Init callback (+) MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

2. Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) TxCpltCallback : I2S Tx Completed callback (+) RxCpltCallback : I2S Rx Completed callback (+) TxRxCpltCallback : I2S TxRx Completed callback (+) TxHalfCpltCallback : I2S Tx Half Completed callback (+) RxHalfCpltCallback : I2S Rx Half Completed callback (+) TxRxHalfCpltCallback : I2S TxRx Half Completed callback (+) ErrorCallback : I2S Error callback (+) MspInitCallback : I2S Msp Init callback (+) MspDeInitCallback : I2S Msp DeInit callback By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function. When The compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

Callback registration: (#) The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : I2S Tx Completed callback
- RxCpltCallback : I2S Rx Completed callback
- TxRxCpltCallback : I2S TxRx Completed callback
- TxHalfCpltCallback : I2S Tx Half Completed callback
- RxHalfCpltCallback : I2S Rx Half Completed callback
- TxRxHalfCpltCallback : I2S TxRx Half Completed callback
- ErrorCallback : I2S Error callback
- MspInitCallback : I2S Msp Init callback
- MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- TxCpltCallback : I2S Tx Completed callback
- RxCpltCallback : I2S Rx Completed callback
- TxRxCpltCallback : I2S TxRx Completed callback
- TxHalfCpltCallback : I2S Tx Half Completed callback
- RxHalfCpltCallback : I2S Rx Half Completed callback
- TxRxHalfCpltCallback : I2S TxRx Half Completed callback
- ErrorCallback : I2S Error callback
- MspInitCallback : I2S Msp Init callback
- MspDeInitCallback : I2S Msp DeInit callback By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function. When The compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

38.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- ***HAL_I2S_Init()***
- ***HAL_I2S_DeInit()***
- ***HAL_I2S_MspInit()***
- ***HAL_I2S_MspDeInit()***
- ***HAL_I2S_RegisterCallback()***
- ***HAL_I2S_UnRegisterCallback()***

38.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2S_Transmit()
   - HAL_I2S_Receive()
   - HAL_I2SEx_TransmitReceive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_Transmit_IT()
   - HAL_I2S_Receive_IT()
   - HAL_I2SEx_TransmitReceive_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2S_Transmit_DMA()
   - HAL_I2S_Receive_DMA()
   - HAL_I2SEx_TransmitReceive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2S_TxCpltCallback()
   - HAL_I2S_RxCpltCallback()
   - HAL_I2SEx_TxRxCpltCallback()
   - HAL_I2S_ErrorCallback()

This section contains the following APIs:

- ***HAL_I2S_Transmit()***
- ***HAL_I2S_Receive()***
- ***HAL_I2SEx_TransmitReceive()***
- ***HAL_I2S_Transmit_IT()***

- *HAL_I2S_Receive_IT()*
- *HAL_I2SEx_TransmitReceive_IT()*
- *HAL_I2S_Transmit_DMA()*
- *HAL_I2S_Receive_DMA()*
- *HAL_I2SEx_TransmitReceive_DMA()*
- *HAL_I2S_DMAPause()*
- *HAL_I2S_DMAResume()*
- *HAL_I2S_DMAStop()*
- *HAL_I2S_IRQHandler()*
- *HAL_I2S_TxHalfCpltCallback()*
- *HAL_I2S_TxCpltCallback()*
- *HAL_I2S_RxHalfCpltCallback()*
- *HAL_I2S_RxCpltCallback()*
- *HAL_I2SEx_TxRxHalfCpltCallback()*
- *HAL_I2SEx_TxRxCpltCallback()*
- *HAL_I2S_ErrorCallback()*

38.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_I2S_GetState()*
- *HAL_I2S_GetError()*

38.2.5 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DeInit

Function name

HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)

Function description

DeInitializes the I2S peripheral.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP Init.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_MspDeInit

Function name

void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP DeInit.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_RegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID, pI2S_CallbackTypeDef pCallback)

Function description

Register a User I2S Callback To be used instead of the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_I2S_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_UnRegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2S Callback I2S callback is redirected to the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

HAL_I2S_Transmit

Function name

HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive

Function name

HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

HAL_I2SEx_TransmitReceive

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Transmit/Receive data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronization between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2SEx_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit/Receive data in non-blocking mode using Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_IRQHandler

Function name

void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)

Function description

This function handles I2S interrupt request.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Transmit data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2SEx_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit/Receive data in non-blocking mode using DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)

Function description

Pauses the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

Resumes the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

Stops the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name

void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer Half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_TxCpltCallback

Function name

void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxHalfCpltCallback

Function name

void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxCpltCallback

Function name

void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2SEx_TxRxHalfCpltCallback

Function name

void HAL_I2SEx_TxRxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2SEx_TxRxCpltCallback

Function name

void HAL_I2SEx_TxRxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_ErrorCallback

Function name

void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)

Function description

I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_GetState

Function name

HAL_I2S_StateTypeDef HAL_I2S_GetState (const I2S_HandleTypeDef * hi2s)

Function description

Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** state

HAL_I2S_GetError

Function name

uint32_t HAL_I2S_GetError (const I2S_HandleTypeDef * hi2s)

Function description

Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **I2S:** Error Code

38.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

38.3.1 I2S

I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S FullDuplex Mode

I2S_CPOL_LOW

I2S_CPOL_HIGH

Data Padding 24Bit

I2S_DATA_24BIT_ALIGNMENT_RIGHT

I2S_DATA_24BIT_ALIGNMENT_LEFT

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error

HAL_I2S_ERROR_NONE
No error

HAL_I2S_ERROR_TIMEOUT
Timeout error

HAL_I2S_ERROR_OVR
OVR error

HAL_I2S_ERROR_UDR
UDR error

HAL_I2S_ERROR_DMA
DMA transfer error

HAL_I2S_ERROR_PRESCALER
Prescaler Calculation error

HAL_I2S_ERROR_FRE
FRE error

HAL_I2S_ERROR_NO_OGT
No On Going Transfer error

HAL_I2S_ERROR_NOT_SUPPORTED
Requested operation not supported

HAL_I2S_ERROR_INVALID_CALLBACK
Invalid Callback error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

- Check if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_DXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_I2S_GET_FLAG

Description:

- Check whether the specified I2S flag is set or not.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2S_FLAG_RXP : Rx-Packet available flag
  - I2S_FLAG_TXP : Tx-Packet space available flag
  - I2S_FLAG_UDR : Underrun flag
  - I2S_FLAG_OVR : Overrun flag
  - I2S_FLAG_FRE : TI mode frame format error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clear the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clear the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_TIFREFLAG

Description:

- Clear the I2S FRE pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flags Definition

I2S_FLAG_RXP

I2S_FLAG_TXP

I2S_FLAG_DXP

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_MASK

I2S Interrupts Definition

I2S_IT_RXP

I2S_IT_TXP

I2S_IT_DXP

I2S_IT_UDR

I2S_IT_OVR

I2S_IT_FRE

I2S_IT_ERR

Keep IO State

I2S_MASTER_KEEP_IO_STATE_DISABLE

I2S_MASTER_KEEP_IO_STATE_ENABLE

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S_MODE_SLAVE_FULLDUPLEX

I2S_MODE_MASTER_FULLDUPLEX

I2S MSB LSB Transmission

I2S_FIRSTBIT_MSB

I2S_FIRSTBIT_LSB

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

I2S Word Select Inversion

I2S_WS_INVERSION_DISABLE

I2S_WS_INVERSION_ENABLE

39 HAL I3C Generic Driver

39.1 I3C Firmware driver registers structures

39.1.1 I3C_InitTypeDef

I3C_InitTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***LL_I3C_CtrlBusConfTypeDef CtrlBusCharacteristic***
- ***LL_I3C_TgtBusConfTypeDef TgtBusCharacteristic***

Field Documentation

- ***LL_I3C_CtrlBusConfTypeDef I3C_InitTypeDef::CtrlBusCharacteristic***
  Specifies the I3C controller bus characteristic configuration when Controller mode
- ***LL_I3C_TgtBusConfTypeDef I3C_InitTypeDef::TgtBusCharacteristic***
  Specifies the I3C target bus characteristic configuration when Target mode

39.1.2 I3C_FifoConfTypeDef

I3C_FifoConfTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint32_t RxFifoThreshold***
- ***uint32_t TxFifoThreshold***
- ***uint32_t ControlFifo***
- ***uint32_t StatusFifo***

Field Documentation

- ***uint32_t I3C_FifoConfTypeDef::RxFifoThreshold***
  Specifies the I3C Rx FIFO threshold level. This parameter must be a value of ***I3C_RX_FIFO_THRESHOLD***
- ***uint32_t I3C_FifoConfTypeDef::TxFifoThreshold***
  Specifies the I3C Tx FIFO threshold level. This parameter must be a value of ***I3C_TX_FIFO_THRESHOLD***
- ***uint32_t I3C_FifoConfTypeDef::ControlFifo***
  Specifies the I3C control FIFO enable/disable state. This parameter is configured only with controller mode and it must be a value of ***I3C_CONTROL_FIFO_STATE***
- ***uint32_t I3C_FifoConfTypeDef::StatusFifo***
  Specifies the I3C status FIFO enable/disable state. This parameter is configured only with controller mode and it must be a value of ***I3C_STATUS_FIFO_STATE***

39.1.3 I3C_CtrlConfTypeDef

I3C_CtrlConfTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t DynamicAddr***
- ***uint8_t StallTime***
- ***FunctionalState HotJoinAllowed***
- ***FunctionalState ACKStallState***
- ***FunctionalState CCCStallState***
- ***FunctionalState TxStallState***
- ***FunctionalState RxStallState***
- ***FunctionalState HighKeeperSDA***

Field Documentation

- ***uint8_t I3C_CtrlConfTypeDef::DynamicAddr***
  Specifies the dynamic address of the controller when goes in target mode. This parameter must be a number between Min_Data=0x00 and Max_Data=0x7F

- ***uint8_t I3C_CtrlConfTypeDef::StallTime***
  Specifies the controller clock stall time in number of kernel clock cycles. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***FunctionalState I3C_CtrlConfTypeDef::HotJoinAllowed***
  Specifies the Enable/Disable state of the controller Hot Join acknowledgement when receiving a hot join request from target. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::ACKStallState***
  Specifies the Enable/Disable state of the controller clock stall on the ACK phase. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::CCCStallState***
  Specifies the Enable/Disable state of the controller clock stall on the T bit phase of a CCC communication to allow the target to decode command. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::TxStallState***
  Specifies the Enable/Disable state of the controller clock stall on parity phase of data to allow the target to read received data. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::RxStallState***
  Specifies the Enable/Disable state of the controller clock stall on the T bit phase of data enable to allow the target to prepare data to be sent. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_CtrlConfTypeDef::HighKeeperSDA***
  Specifies the Enable/Disable state of the controller SDA high keeper. This parameter can be set to ENABLE or DISABLE

39.1.4 I3C_TgtConfTypeDef

I3C_TgtConfTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t Identifier***
- ***uint8_t MIPIIdentifier***
- ***FunctionalState CtrlRoleRequest***
- ***FunctionalState HotJoinRequest***
- ***FunctionalState IBIRequest***
- ***FunctionalState IBIPayload***
- ***uint32_t IBIPayloadSize***
- ***uint16_t MaxReadDataSize***
- ***uint16_t MaxWriteDataSize***
- ***FunctionalState CtrlCapability***
- ***FunctionalState GroupAddrCapability***
- ***uint32_t DataTurnAroundDuration***
- ***uint8_t MaxReadTurnAround***
- ***uint32_t MaxDataSpeed***
- ***FunctionalState MaxSpeedLimitation***
- ***uint32_t HandOffActivityState***
- ***FunctionalState HandOffDelay***
- ***FunctionalState PendingReadMDB***

Field Documentation

- ***uint8_t I3C_TgtConfTypeDef::Identifier***
  Specifies the target characteristic ID (MIPI named reference DCR). This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***uint8_t I3C_TgtConfTypeDef::MIPIIdentifier***
  Specifies the bits [12-15] of the 48-provisioned ID (MIPI named reference PID), other 48-provisioned ID are hardcoded. This parameter must be a number between Min_Data=0x00 and Max_Data=0x0F
- ***FunctionalState I3C_TgtConfTypeDef::CtrlRoleRequest***
  Specifies the Enable/Disable state of the target authorization request for a second master Chip. This parameter can be set to ENABLE or DISABLE

- ***FunctionalState I3C_TgtConfTypeDef::HotJoinRequest***
  Specifies the Enable/Disable state of the target hot join authorization request. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::IBIRequest***
  Specifies the Enable/Disable state of the target in Band Interrupt authorization request. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::IBIPayload***
  Specifies the Enable/Disable state of sending data payload after an accepted IBI. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::IBIPayloadSize***
  Specifies the I3C target payload data size. This parameter must be a value of ***I3C_PAYLOAD_SIZE***
- ***uint16_t I3C_TgtConfTypeDef::MaxReadDataSize***
  Specifies the numbers of data bytes that the target can read at maximum. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFFFF
- ***uint16_t I3C_TgtConfTypeDef::MaxWriteDataSize***
  Specifies the numbers of data bytes that the target can write at maximum. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFFFF
- ***FunctionalState I3C_TgtConfTypeDef::CtrlCapability***
  Specifies the Enable/Disable state of the target controller capability. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::GroupAddrCapability***
  Specifies the Enable/Disable state of the target support of group address after a controller role hand-off. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::DataTurnAroundDuration***
  Specifies the I3C target clock-to-data turnaround time. This parameter must be a value of ***I3C_TURNAROUND_TIME_TSCO***
- ***uint8_t I3C_TgtConfTypeDef::MaxReadTurnAround***
  Specifies the target maximum read turnaround byte. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF
- ***uint32_t I3C_TgtConfTypeDef::MaxDataSpeed***
  Specifies the I3C target returned GETMXDS CCC format. This parameter must be a value of ***I3C_GETMXDS_FORMAT***
- ***FunctionalState I3C_TgtConfTypeDef::MaxSpeedLimitation***
  Specifies the Enable/Disable state of the target max data speed limitation. This parameter can be set to ENABLE or DISABLE
- ***uint32_t I3C_TgtConfTypeDef::HandOffActivityState***
  Specifies the I3C target activity state when becoming controller. This parameter must be a value of ***I3C_HANDOFF_ACTIVITY_STATE***
- ***FunctionalState I3C_TgtConfTypeDef::HandOffDelay***
  Specifies the Enable/Disable state of the target need of delay to process the controller role hand-off. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_TgtConfTypeDef::PendingReadMDB***
  Specifies the Enable/Disable state of the transmission of a mandatory data bytes indicating a pending read notification for GETCAPR CCC command. This parameter can be set to ENABLE or DISABLE

39.1.5 I3C_DeviceConfTypeDef

I3C_DeviceConfTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t DeviceIndex***
- ***uint8_t TargetDynamicAddr***
- ***FunctionalState IBIAck***
- ***FunctionalState IBIPayload***
- ***FunctionalState CtrlRoleReqAck***
- ***FunctionalState CtrlStopTransfer***

Field Documentation

- ***uint8_t I3C_DeviceConfTypeDef::DeviceIndex***
  Specifies the index value of the device in the DEVRx register. This parameter must be a number between Min_Data=1 and Max_Data=4
- ***uint8_t I3C_DeviceConfTypeDef::TargetDynamicAddr***
  Specifies the dynamic address of the target x (1 to 4) connected on the bus. This parameter must be a number between Min_Data=0x00 and Max_Data=0x7F
- ***FunctionalState I3C_DeviceConfTypeDef::IBIAck***
  Specifies the Enable/Disable state of the controller's ACK when receiving an IBI from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::IBIPayload***
  Specifies the Enable/Disable state of the controller's receiving IBI payload after acknowledging an IBI requested from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::CtrlRoleReqAck***
  Specifies the Enable/Disable state of the controller's ACK when receiving a control request from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState I3C_DeviceConfTypeDef::CtrlStopTransfer***
  Specifies the Enable/Disable state of the controller's stop transfer after receiving an IBI request from a target x (1 to 4) connected on the bus. This parameter can be set to ENABLE or DISABLE

39.1.6 I3C_CCCInfoTypeDef

I3C_CCCInfoTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint32_t DynamicAddrValid***
- ***uint32_t DynamicAddr***
- ***uint32_t MaxWriteLength***
- ***uint32_t MaxReadLength***
- ***uint32_t ResetAction***
- ***uint32_t ActivityState***
- ***uint32_t HotJoinAllowed***
- ***uint32_t InBandAllowed***
- ***uint32_t CtrlRoleAllowed***
- ***uint32_t IBICRTgtAddr***
- ***uint32_t IBITgtNbPayload***
- ***uint32_t IBITgtPayload***

Field Documentation

- ***uint32_t I3C_CCCInfoTypeDef::DynamicAddrValid***
  I3C target Dynamic Address Valid (updated during ENTDAA/RSTDAA/SETNEWDA CCC) This parameter can be Valid=1U or Not Valid=0U
- ***uint32_t I3C_CCCInfoTypeDef::DynamicAddr***
  I3C target Dynamic Address (updated during ENTDAA/RSTDAA/SETNEWDA CCC)
- ***uint32_t I3C_CCCInfoTypeDef::MaxWriteLength***
  I3C target Maximum Write Length (updated during SETMWL CCC)
- ***uint32_t I3C_CCCInfoTypeDef::MaxReadLength***
  I3C target Maximum Read Length (updated during SETMRL CCC)
- ***uint32_t I3C_CCCInfoTypeDef::ResetAction***
  I3C target Reset Action level (updated during RSTACT CCC)
- ***uint32_t I3C_CCCInfoTypeDef::ActivityState***
  I3C target Activity State (updated during ENTASx CCC)
- ***uint32_t I3C_CCCInfoTypeDef::HotJoinAllowed***
  I3C target Hot Join (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U
- ***uint32_t I3C_CCCInfoTypeDef::InBandAllowed***
  I3C target In Band Interrupt (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U

- ***uint32_t I3C_CCCInfoTypeDef::CtrlRoleAllowed***
  I3C target Controller Role Request (updated during ENEC/DISEC CCC) This parameter can be Allowed=1U or Not Allowed=0U
- ***uint32_t I3C_CCCInfoTypeDef::IBICRTgtAddr***
  I3C controller receive Target Address during IBI or Controller Role Request event
- ***uint32_t I3C_CCCInfoTypeDef::IBITgtNbPayload***
  I3C controller get Number of Data Payload after an IBI event
- ***uint32_t I3C_CCCInfoTypeDef::IBITgtPayload***
  I3C controller receive IBI Payload after an IBI event

39.1.7 I3C_ControlTypeDef

I3C_ControlTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint32_t * pBuffer***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* I3C_ControlTypeDef::pBuffer***
  Pointer to the buffer containing the control or status register values
- ***uint32_t I3C_ControlTypeDef::Size***
  The size of pBuffer in words

39.1.8 I3C_DataTypeDef

I3C_DataTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t * pBuffer***
- ***uint32_t Size***

Field Documentation

- ***uint8_t* I3C_DataTypeDef::pBuffer***
  Pointer to the buffer containing all data values to transfer
- ***uint32_t I3C_DataTypeDef::Size***
  The size of pBuffer in bytes

39.1.9 I3C_CCCTypeDef

I3C_CCCTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t TargetAddr***
- ***uint8_t CCC***
- ***I3C_DataTypeDef CCCBuf***
- ***uint32_t Direction***

Field Documentation

- ***uint8_t I3C_CCCTypeDef::TargetAddr***
  Dynamic or Static target Address
- ***uint8_t I3C_CCCTypeDef::CCC***
  CCC value code
- ***I3C_DataTypeDef I3C_CCCTypeDef::CCCBuf***
  Contain size of associated data and size of defining byte if any. Contain pointer to CCC associated data
- ***uint32_t I3C_CCCTypeDef::Direction***
  CCC read and/or write direction message

39.1.10 I3C_PrivateTypeDef

I3C_PrivateTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***uint8_t TargetAddr***
- ***I3C_DataTypeDef TxBuf***
- ***I3C_DataTypeDef RxBuf***
- ***uint32_t Direction***

Field Documentation

- ***uint8_t I3C_PrivateTypeDef::TargetAddr***
  Dynamic or Static target Address
- ***I3C_DataTypeDef I3C_PrivateTypeDef::TxBuf***
  Buffer structure containing the data to transmit (little endian)
- ***I3C_DataTypeDef I3C_PrivateTypeDef::RxBuf***
  Buffer structure containing the data to receive (little endian)
- ***uint32_t I3C_PrivateTypeDef::Direction***
  Read and/or write message

39.1.11 I3C_XferTypeDef

I3C_XferTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***I3C_ControlTypeDef CtrlBuf***
- ***I3C_ControlTypeDef StatusBuf***
- ***I3C_DataTypeDef TxBuf***
- ***I3C_DataTypeDef RxBuf***

Field Documentation

- ***I3C_ControlTypeDef I3C_XferTypeDef::CtrlBuf***
  Buffer structure containing the control register values
- ***I3C_ControlTypeDef I3C_XferTypeDef::StatusBuf***
  Buffer structure containing the status register values
- ***I3C_DataTypeDef I3C_XferTypeDef::TxBuf***
  Buffer structure containing the data to transmit
- ***I3C_DataTypeDef I3C_XferTypeDef::RxBuf***
  Buffer structure containing the data to receive

39.1.12 __I3C_HandleTypeDef

__I3C_HandleTypeDef is defined in the stm32h5xx_hal_i3c.h

Data Fields

- ***I3C_TypeDef * Instance***
- ***I3C_InitTypeDef Init***
- ***HAL_I3C_ModeTypeDef Mode***
- ***I3C_XferTypeDef * pXferData***
- ***const I3C_CCCTypeDef * pCCCDesc***
- ***const I3C_PrivateTypeDef * pPrivateDesc***
- ***uint32_t ControlXferCount***
- ***uint32_t RxXferCount***
- ***uint32_t TxXferCount***
- ***DMA_HandleTypeDef * hdmacr***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***DMA_HandleTypeDef * hdmasr***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_I3C_StateTypeDef State***
- ***__IO HAL_I3C_StateTypeDef PreviousState***
- ***__IO uint32_t ErrorCode***

- ***HAL_StatusTypeDef(* XferISR***
- ***void(* ptrTxFunc***
- ***void(* ptrRxFunc***
- ***void(* CtrlTxCpltCallback***
- ***void(* CtrlRxCpltCallback***
- ***void(* CtrlMultipleXferCpltCallback***
- ***void(* CtrlDAACpltCallback***
- ***void(* TgtReqDynamicAddrCallback***
- ***void(* TgtTxCpltCallback***
- ***void(* TgtRxCpltCallback***
- ***void(* TgtHotJoinCallback***
- ***void(* NotifyCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I3C_TypeDef* __I3C_HandleTypeDef::Instance***
  I3C registers base address
- ***I3C_InitTypeDef __I3C_HandleTypeDef::Init***
  I3C communication parameters
- ***HAL_I3C_ModeTypeDef __I3C_HandleTypeDef::Mode***
  I3C communication mode. This parameter must be a value of
  Section 39.3.1 I3C_mode_structure_definition
- ***I3C_XferTypeDef* __I3C_HandleTypeDef::pXferData***
  I3C transfer buffers pointer
- ***const I3C_CCCTypeDef* __I3C_HandleTypeDef::pCCCDesc***
  I3C CCC descriptor pointer
- ***const I3C_PrivateTypeDef* __I3C_HandleTypeDef::pPrivateDesc***
  I3C private transfer descriptor pointer
- ***uint32_t __I3C_HandleTypeDef::ControlXferCount***
  I3C counter indicating the remaining control data bytes to write in the control register
- ***uint32_t __I3C_HandleTypeDef::RxXferCount***
  I3C counter indicating the remaining data bytes to receive
- ***uint32_t __I3C_HandleTypeDef::TxXferCount***
  I3C counter indicating the remaining data bytes to transmit
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmacr***
  I3C control DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmatx***
  I3C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmarx***
  I3C Rx DMA handle parameters
- ***DMA_HandleTypeDef* __I3C_HandleTypeDef::hdmasr***
  I3C status DMA handle parameters
- ***HAL_LockTypeDef __I3C_HandleTypeDef::Lock***
  I3C locking object
- ***__IO HAL_I3C_StateTypeDef __I3C_HandleTypeDef::State***
  I3C communication state
- ***__IO HAL_I3C_StateTypeDef __I3C_HandleTypeDef::PreviousState***
  I3C communication previous state
- ***__IO uint32_t __I3C_HandleTypeDef::ErrorCode***
  I3C Error code

- ***HAL_StatusTypeDef(* __I3C_HandleTypeDef::XferISR)(struct __I3C_HandleTypeDef *hi3c, uint32_t itFlags, uint32_t itSources)***
  I3C transfer IRQ handler function pointer
- ***void(* __I3C_HandleTypeDef::ptrTxFunc)(struct __I3C_HandleTypeDef *hi3c)***
  I3C transmit function pointer
- ***void(* __I3C_HandleTypeDef::ptrRxFunc)(struct __I3C_HandleTypeDef *hi3c)***
  I3C receive function pointer
- ***void(* __I3C_HandleTypeDef::CtrlTxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller private data and CCC Tx Transfer complete callback
- ***void(* __I3C_HandleTypeDef::CtrlRxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller private data and CCC Rx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::CtrlMultipleXferCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller multiple Direct CCC, I3C private or I2C Transfer completed callback
- ***void(* __I3C_HandleTypeDef::CtrlDAACpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Controller Dynamic Address Assignment completed callback
- ***void(* __I3C_HandleTypeDef::TgtReqDynamicAddrCallback)(struct __I3C_HandleTypeDef *hi3c, uint64_t targetPayload)***
  I3C Controller request dynamic address callback during Dynamic Address Assignment processus
- ***void(* __I3C_HandleTypeDef::TgtTxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Target private data Tx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::TgtRxCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Target private data Rx Transfer completed callback
- ***void(* __I3C_HandleTypeDef::TgtHotJoinCallback)(struct __I3C_HandleTypeDef *hi3c, uint8_t dynamicAddress)***
  I3C Target Hot-Join callback
- ***void(* __I3C_HandleTypeDef::NotifyCallback)(struct __I3C_HandleTypeDef *hi3c, uint32_t eventId)***
  I3C Target or Controller asynchronous events callback
- ***void(* __I3C_HandleTypeDef::ErrorCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Error callback
- ***void(* __I3C_HandleTypeDef::AbortCpltCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Abort complete callback
- ***void(* __I3C_HandleTypeDef::MspInitCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Msp Init callback
- ***void(* __I3C_HandleTypeDef::MspDeInitCallback)(struct __I3C_HandleTypeDef *hi3c)***
  I3C Msp DeInit callback

39.2 I3C Firmware driver API description

The following section lists the various functions of the I3C library.

39.2.1 How to use this driver

The I3C HAL driver can be used as follows:

1. Declare a I3C_HandleTypeDef handle structure, for example: I3C_HandleTypeDef hi3c;
2. Declare a I3C_XferTypeDef transfer descriptor structure, for example: I3C_XferTypeDef ContextBuffers;

3. Initialize the I3C low level resources by implementing the HAL_I3C_MspInit() API:
   a. Enable the I3Cx interface clock
   b. I3C pins configuration
      - Enable the clock for the I3C GPIOs
      - Configure I3C pins as alternate function push-pull with no-pull
   c. NVIC configuration if you need to use interrupt process
      - Configure the I3Cx interrupt priority
      - Enable the NVIC I3C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the Command Common Code (CCC) management channel
      - Declare a DMA_HandleTypeDef handle structure for the transmit channel
      - Declare a DMA_HandleTypeDef handle structure for the receive channel
      - Declare a DMA_HandleTypeDef handle structure for the status channel
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Command Common Code (CCC) channel
      - Configure the DMA Tx channel
      - Configure the DMA Rx channel
      - Configure the DMA Status channel
      - Associate the initialized DMA handle to the hi3c DMA CCC, Tx, Rx or Status handle as necessary
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CCC, Tx, Rx or Status instance
4. Configure the HAL I3C Communication Mode as Controller or Target in the hi3c Init structure.
5. Configure the Controller Communication Bus characterics for Controller mode. This mean, configure the parameters SDAHoldTime, WaitTime, SCLPPLowDuration, SCLI3CHighDuration, SCLODLowDuration, SCLI2CHighDuration, BusFreeDuration, BusIdleDuration in the LL_I3C_CtrlBusConfTypeDef structure through h3c Init structure.
6. Configure the Target Communication Bus characterics for Target mode. This mean, configure the parameter BusAvailableDuration in the LL_I3C_TgtBusConfTypeDef structure through h3c Init structure. All these parameters for Controller or Target can be configured directly in user code or by using CubeMx generation. To help the computation of the different parameters, the recommendation is to use CubeMx. Those parameters can be modified after the hi3c initialization by using HAL_I3C_Ctrl_BusCharacteristicConfig() for controller and HAL_I3C_Tgt_BusCharacteristicConfig() for target.
7. Initialize the I3C registers by calling the HAL_I3C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I3C_MspInit(&hi3c) API.
8. Configure the different FIFO parameters in I3C_FifoConfTypeDef structure as RxFifoThreshold, TxFifoThreshold for Controller or Target mode. And enable/disable the Control or Status FIFO only for Controller Mode. Use HAL_I3C_SetConfigFifo() function to finalize the configuration, and HAL_I3C_GetConfigFifo() to retrieve FIFO configuration. Possibility to clear the FIFO configuration by using HAL_I3C_ClearConfigFifo() which reset the configuration FIFO to their default hardware value
9. Configure the different additional Controller configuration in I3C_CtrlConfTypeDef structure as DynamicAddr, StallTime, HotJoinAllowed, ACKStallState, CCCStallState, TxStallState, RxStallState, HighKeeperSDA. Use HAL_I3C_Ctrl_Config() function to finalize the Controller configuration.
10. Configure the different additional Target configuration in I3C_TgtConfTypeDef structure as Identifier, MIPIIdentifier, CtrlRoleRequest, HotJoinRequest, IBIRequest, IBIPayload, IBIPayloadSize, MaxReadDataSize, MaxWriteDataSize, CtrlCapability, GroupAddrCapability, DataTurnAroundDuration, MaxReadTurnAround, MaxDataSpeed, MaxSpeedLimitation, HandOffActivityState, HandOffDelay, PendingReadMDB. Use HAL_I3C_Tgt_Config() function to finalize the Target configuration.

11. Before initiate any IO operation, the application must launch an assignment of the different Target dynamic address by using HAL_I3C_Ctrl_DynAddrAssign() in polling mode or HAL_I3C_Ctrl_DynAddrAssign_IT() in interrupt mode. This procedure is named Enter Dynamic Address Assignment (ENTDAA CCC command). For the initiation of ENTDAA procedure from the controller, each target connected and powered on the I3C bus must repond to this particular Command Common Code by sending its proper Payload (a amount of 48bits which contain the target characteristics) Each time a target responds to ENTDAA sequence, the application is informed through HAL_I3C_TgtReqDynamicAddrCallback() of the reception of the target paylaod. And then application must send a associated dynamic address through HAL_I3C_Ctrl_SetDynAddr(). This procedure in loop automatically in hardware side until a target respond to repeated ENTDAA sequence. The application is informed of the end of the procedure at reception of HAL_I3C_CtrlDAACpltCallback(). At the end of procedure, the function HAL_I3C_Ctrl_ConfigBusDevices() must be called to store in hardware register part the target capabilities as Dynamic address, IBI support with or without additional data byte, Controller role request support, Controller stop transfer after IBI through I3C_DeviceConfTypeDef structure.
12. Other action to be done, before initiate any IO operation, the application must prepare the different frame descriptor with its associated buffer allocation in their side. Configure the different information related to CCC transfer through I3C_CCCTypeDef structure Configure the different information related to Private or I2C transfer through I3C_PrivateTypeDef structure Configure the different buffer pointers and associated size needed for the driver communication through I3C_XferTypeDef structure The I3C_XferTypeDef structure contains different parameters about Control, Status buffer, and Transmit and Receive buffer. Use HAL_I3C_AddDescToFrame() function each time application add a descriptor in the frame before call an IO operation interface One element of the frame descriptor correspond to one frame to manage through IO operation.
13. To check if I3C target device is ready for communication, use the function HAL_I3C_Ctrl_IsDeviceI3C_Ready()
14. To check if I2C target device is ready for communication, use the function HAL_I3C_Ctrl_IsDeviceI2C_Ready()
15. For I3C IO operations, three operation modes are available within this driver :

Polling mode IO operation

- Activate asynchronous event in controller or target mode a Common Command Code in a broadcast or a direct communication in blocking mode using HAL_I3C_Ctrl_TransmitCCC()
- Transmit in controller mode a Common Command Code in a broadcast or a direct communication in blocking mode using HAL_I3C_Ctrl_TransmitCCC()
- Receive in controller mode a Common Command Code in a direct communication in blocking mode using HAL_I3C_Ctrl_ReceiveCCC()
- Transmit in controller mode an amount of private data in an I3C or an I2C communication in blocking mode using HAL_I3C_Ctrl_Transmit()
- Receive in controller mode an amount of private data in an I3C or an I2C communication in blocking mode using HAL_I3C_Ctrl_Receive()
- Transmit in target mode an amount of private data in an I3C communication in blocking mode using HAL_I3C_Tgt_Transmit()
- Receive in target mode an amount of private data in an I3C communication in blocking mode using HAL_I3C_Tgt_Receive()
- At the end of a transfer, the different FIFO can be flushed if necessary by using HAL_I3C_FlushAllFifo() for flush all the FIFO, or flush individually y using HAL_I3C_FlushTxFifo(), HAL_I3C_FlushRxFifo(), HAL_I3C_FlushControlFifo(), HAL_I3C_FlushStatusFifo().
- Request a HotJoin in target mode in blocking mode using HAL_I3C_Tgt_HotJoinReq()
- Request a In Band Interrupt in target mode in blocking mode using HAL_I3C_Tgt_IBIReq()
- Request a Controller Role in target mode in blocking mode using HAL_I3C_Tgt_ControlRoleReq()

DMA and Interrupt mode IO operation

- Transmit in controller mode a Common Command Code in a broadcast or a direct communication in non-blocking mode using HAL_I3C_Ctrl_TransmitCCC_IT() or HAL_I3C_Ctrl_TransmitCCC_DMA()
- At transmission end of transfer, HAL_I3C_CtrlTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlTxCpltCallback()

- Receive in controller mode a Common Command Code in a direct communication in non-blocking mode using HAL_I3C_Ctrl_ReceiveCCC_IT() or HAL_I3C_Ctrl_ReceiveCCC_DMA()
- At reception end of transfer, HAL_I3C_CtrlRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlRxCpltCallback()
- Transmit in controller mode an amount of private data in an I3C or an I2C communication in non-blocking mode using HAL_I3C_Ctrl_Transmit_IT() or HAL_I3C_Ctrl_Transmit_DMA()
- At transmission end of transfer, HAL_I3C_CtrlTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlTxCpltCallback()
- Receive in controller mode an amount of private data in an I3C or an I2C communication in non-blocking mode using HAL_I3C_Ctrl_Receive_IT() or HAL_I3C_Ctrl_Receive_DMA()
- At reception end of transfer, HAL_I3C_CtrlRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlRxCpltCallback()
- Transfer in multiple direction (transmit/receive) in controller mode a Common Command Code in a direct communication or an amount of private data in an I2C or I3C communication in non-blocking mode using HAL_I3C_Ctrl_MultipleTransfer_IT() or HAL_I3C_Ctrl_MultipleTransfer_DMA()
- At the end of transfer, HAL_I3C_CtrlMultipleXferCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_CtrlMultipleXferCpltCallback()
- Transmit in target mode an amount of private data in an I3C communication in non-blocking mode using HAL_I3C_Tgt_Transmit_IT() or HAL_I3C_Tgt_Transmit_DMA()
- At transmission end of transfer, HAL_I3C_TgtTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtTxCpltCallback()
- Receive in target mode an amount of private data in an I3C communication in non-blocking mode using HAL_I3C_Tgt_Receive_IT() or HAL_I3C_Tgt_Receive_DMA()
- At reception end of transfer, HAL_I3C_TgtRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtRxCpltCallback()
- To treat asynchronous event, HAL_I3C_ActivateNotification() or HAL_I3C_DeactivateNotification() function is used for enable or disable one or more notification related to specific asynchronous event. Each time one or more event detected by hardware the associated HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback(). Then application can easily retrieve some specific associated event data through HAL_I3C_GetCCCInfo() function
- At the end of a transfer, the different FIFO can be flushed if necessary by using HAL_I3C_FlushAllFifo() for flush all the FIFO, or flush individually y using HAL_I3C_FlushTxFifo(), HAL_I3C_FlushRxFifo(), HAL_I3C_FlushControlFifo(), HAL_I3C_FlushStatusFifo().
- Request a HotJoin in target mode in non-blocking mode using HAL_I3C_Tgt_HotJoinReq_IT
- At completion, HAL_I3C_TgtHotJoinCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_TgtHotJoinCallback()
- Request an In Band Interrupt in target mode in non-blocking mode using HAL_I3C_Tgt_IBIReq_IT()
- At completion, HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback()
- Request a Controller Role in target mode in non-blocking mode using HAL_I3C_Tgt_ControlRoleReq_IT()
- At completion, HAL_I3C_NotifyCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_NotifyCallback()
- To manage the wakeup capability, HAL_I3C_ActivateNotification() or HAL_I3C_DeactivateNotification() function is used for enable or disable Wake Up interrupt. At wakeup detection the associated HAL_I3C_NotifyCallback() is executed.
- In case of transfer Error, HAL_I3C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I3C_ErrorCallback()
- Abort an I3C process communication with Interrupt using HAL_I3C_Abort_IT()
- End of abort process, HAL_I3C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I3C_AbortCpltCallback()

I3C HAL driver macros list

Below the list of most used macros in I3C HAL driver.

- __HAL_I3C_ENABLE: Enable the I3C peripheral
- __HAL_I3C_DISABLE: Disable the I3C peripheral

- __HAL_I3C_RESET_HANDLE_STATE: Reset the I3C handle state
- __HAL_I3C_GET_FLAG: Check whether the specified I3C flag is set or not

Callback registration

The compilation flag USE_HAL_I3C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I3C_RegisterCallback() or HAL_I3C_RegisterNotifyCallback() or HAL_I3C_RegisterTgtReqDynamicAddrCallback() or HAL_I3C_RegisterTgtHotJoinCallback() to register an interrupt callback.

Function HAL_I3C_RegisterCallback() allows to register following callbacks:

- CtrlTxCpltCallback : callback for Controller transmission CCC, I3C private or I2C end of transfer.
- CtrlRxCpltCallback : callback for Controller reception CCC, I3C private or I2C end of transfer.
- CtrlMultipleXferCpltCallback : callback for Controller multiple Direct CCC, I3C private or I2C end of transfer.
- CtrlDAACpltCallback : callback for Controller Enter Dynamic Address Assignment end of transfer.
- TgtTxCpltCallback : callback for Target transmission I3C private end of transfer.
- TgtRxCpltCallback : callback for Target reception I3C private end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback NotifyCallback use dedicated register callbacks : HAL_I3C_RegisterNotifyCallback().

For specific callback TgtReqDynamicAddrCallback use dedicated register callbacks : HAL_I3C_RegisterTgtReqDynamicAddrCallback().

For specific callback TgtHotJoinCallback use dedicated register callbacks : HAL_I3C_RegisterTgtHotJoinCallback().

Use function HAL_I3C_UnRegisterCallback to reset a callback to the default weak function. HAL_I3C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- CtrlTxCpltCallback : callback for Controller transmission CCC, I3C private or I2C end of transfer.
- CtrlRxCpltCallback : callback for Controller reception CCC, I3C private or I2C end of transfer.
- CtrlMultipleXferCpltCallback : callback for Controller multiple Direct CCC, I3C private or I2C end of transfer.
- CtrlDAACpltCallback : callback for Controller Enter Dynamic Address Assignment end of transfer.
- TgtTxCpltCallback : callback for Target transmission I3C private end of transfer.
- TgtRxCpltCallback : callback for Target reception I3C private end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.
- NotifyCallback : callback for Controller and Target notification process.
- TgtReqDynamicAddrCallback : callback for controller application when a target sent its payload to the controller during Dynamic Address Assignment process.
- TgtHotJoinCallback : callback for Target Hotjoin completion process.

By default, after the HAL_I3C_Init() and when the state is HAL_I3C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I3C_CtrlTxCpltCallback(), HAL_I3C_CtrlRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I3C_Init()/ HAL_I3C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I3C_Init()/ HAL_I3C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I3C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I3C_STATE_READY or HAL_I3C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I3C_RegisterCallback() before calling HAL_I3C_DeInit() or HAL_I3C_Init() function.

When the compilation flag USE_HAL_I3C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I3C HAL driver header file for more useful macros

39.2.2 Detailed description of functions

HAL_I3C_Init

Function name

HAL_StatusTypeDef HAL_I3C_Init (I3C_HandleTypeDef * hi3c)

Function description

Initializes the I3C instance by activating the low-level hardware and configuring the bus characteristic according to the specified parameters in the I3C_InitTypeDef.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_DeInit

Function name

HAL_StatusTypeDef HAL_I3C_DeInit (I3C_HandleTypeDef * hi3c)

Function description

DeInitialize the I3C peripheral.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_MspInit

Function name

void HAL_I3C_MspInit (I3C_HandleTypeDef * hi3c)

Function description

Initialize the I3C MSP.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_MspDeInit

Function name

void HAL_I3C_MspDeInit (I3C_HandleTypeDef * hi3c)

Function description

DeInitialize the I3C MSP.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterCallback (I3C_HandleTypeDef * hi3c, HAL_I3C_CallbackIDTypeDef callbackID, pI3C_CallbackTypeDef pCallback)

Function description

Register a User I3C Callback to be used instead of the weak predefined callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **callbackID:** : [IN] ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_I3C_CTRL_TX_COMPLETE_CB_ID
  - HAL_I3C_CTRL_RX_COMPLETE_CB_ID
  - HAL_I3C_CTRL_MULTIPLE_XFER_COMPLETE_CB_ID
  - HAL_I3C_CTRL_DAA_COMPLETE_CB_ID
  - HAL_I3C_TGT_TX_COMPLETE_CB_ID
  - HAL_I3C_TGT_RX_COMPLETE_CB_ID
  - HAL_I3C_ERROR_CB_ID
  - HAL_I3C_ABORT_CB_ID
  - HAL_I3C_MSPINIT_CB_ID
  - HAL_I3C_MSPDEINIT_CB_ID
- **pCallback:** : [IN] Pointer to the Callback function.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The HAL_I3C_RegisterCallback() may be called before HAL_I3C_Init() in HAL_I3C_STATE_RESET to register callbacks for HAL_I3C_MSPINIT_CB_ID and HAL_I3C_MSPDEINIT_CB_ID

HAL_I3C_RegisterNotifyCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterNotifyCallback (I3C_HandleTypeDef * hi3c, pI3C_NotifyCallbackTypeDef pNotifyCallback)

Function description

Register a User I3C Notify Callback to be used instead of the weak predefined callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pNotifyCallback:** : [IN] Pointer to the Callback function.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_RegisterTgtReqDynamicAddrCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterTgtReqDynamicAddrCallback (I3C_HandleTypeDef * hi3c, pI3C_TgtReqDynamicAddrCallbackTypeDef pTgtReqAddrCallback)

Function description

Register a User I3C dynamic address Callback to be used instead of the weak predefined callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pTgtReqAddrCallback:** : [IN] Pointer to the Callback function.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_RegisterTgtHotJoinCallback

Function name

HAL_StatusTypeDef HAL_I3C_RegisterTgtHotJoinCallback (I3C_HandleTypeDef * hi3c, pI3C_TgtHotJoinCallbackTypeDef pTgtHotJoinCallback)

Function description

Register a User I3C hot join Callback to be used instead of the weak predefined callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pTgtHotJoinCallback:** : [IN] Pointer to the Callback function.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I3C_UnRegisterCallback (I3C_HandleTypeDef * hi3c, HAL_I3C_CallbackIDTypeDef callbackID)

Function description

Unregister a user I3C Callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **callbackID:** : [IN] ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_I3C_CTRL_TX_COMPLETE_CB_ID
  - HAL_I3C_CTRL_RX_COMPLETE_CB_ID
  - HAL_I3C_CTRL_MULTIPLE_XFER_COMPLETE_CB_ID
  - HAL_I3C_CTRL_DAA_COMPLETE_CB_ID
  - HAL_I3C_TGT_REQ_DYNAMIC_ADDR_CB_ID
  - HAL_I3C_TGT_TX_COMPLETE_CB_ID
  - HAL_I3C_TGT_RX_COMPLETE_CB_ID
  - HAL_I3C_TGT_HOTJOIN_CB_ID
  - HAL_I3C_NOTIFY_CB_ID
  - HAL_I3C_ERROR_CB_ID
  - HAL_I3C_ABORT_CB_ID
  - HAL_I3C_MSPINIT_CB_ID
  - HAL_I3C_MSPDEINIT_CB_ID

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The HAL_I3C_UnRegisterCallback() may be called before HAL_I3C_Init() in HAL_I3C_STATE_RESET to un-register callbacks for HAL_I3C_MSPINIT_CB_ID and HAL_I3C_MSPDEINIT_CB_ID

HAL_I3C_ActivateNotification

Function name

HAL_StatusTypeDef HAL_I3C_ActivateNotification (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t interruptMask)

Function description

This function permits to activate the I3C notifications.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN/OUT] Pointer to an I3C_XferTypeDef structure that contains the reception buffer to retrieve data during broadcast CCC DEFTGTS and DEFGRPA when Target mode only.
- **interruptMask:** : [IN] Parameter indicates which interrupts will be enabled. This parameter can be any combination of
  - I3C_TARGET_INTERRUPT when the I3C is in target mode or a combination of
  - I3C_CONTROLLER_INTERRUPT when it is in controller mode.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_I3C_DeactivateNotification (I3C_HandleTypeDef * hi3c, uint32_t interruptMask)

Function description

This function permits to deactivate the I3C notifications.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **interruptMask:** : [IN] Parameter indicates which interrupts will be disabled. This parameter can be any combination of
  - I3C_TARGET_INTERRUPT when the I3C is in target mode or a combination of
  - I3C_CONTROLLER_INTERRUPT when it is in controller mode.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_CtrlTxCpltCallback

Function name

void HAL_I3C_CtrlTxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Controller Transmission Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_CtrlRxCpltCallback

Function name

void HAL_I3C_CtrlRxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Controller Reception Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_CtrlMultipleXferCpltCallback

Function name

void HAL_I3C_CtrlMultipleXferCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Controller multiple Direct CCC Command, I3C private or I2C transfer Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_CtrlDAACpltCallback

Function name

void HAL_I3C_CtrlDAACpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Controller dynamic address assignment Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_TgtReqDynamicAddrCallback

Function name

void HAL_I3C_TgtReqDynamicAddrCallback (I3C_HandleTypeDef * hi3c, uint64_t targetPayload)

Function description

Target Request Dynamic Address callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **targetPayload:** : [IN] Parameter indicates the target payload.

Return values

- **None:**

HAL_I3C_TgtTxCpltCallback

Function name

void HAL_I3C_TgtTxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Target Transmission Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_TgtRxCpltCallback

Function name

void HAL_I3C_TgtRxCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Target Reception Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_TgtHotJoinCallback

Function name

void HAL_I3C_TgtHotJoinCallback (I3C_HandleTypeDef * hi3c, uint8_t dynamicAddress)

Function description

Target Hot join process Complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **dynamicAddress:** : [IN] The returned dynamic address value after the hot join process.

Return values

- **None:**

HAL_I3C_NotifyCallback

Function name

void HAL_I3C_NotifyCallback (I3C_HandleTypeDef * hi3c, uint32_t eventId)

Function description

Target/Controller Notification event callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **eventId:** : [IN] Parameter indicates which notification is signaled. It can be a combination value of I3C Notification ID definition.

Return values

- **None:**

HAL_I3C_AbortCpltCallback

Function name

void HAL_I3C_AbortCpltCallback (I3C_HandleTypeDef * hi3c)

Function description

Abort complete callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_ErrorCallback

Function name

void HAL_I3C_ErrorCallback (I3C_HandleTypeDef * hi3c)

Function description

Error callback.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_ER_IRQHandler

Function name

void HAL_I3C_ER_IRQHandler (I3C_HandleTypeDef * hi3c)

Function description

This function handles I3C error interrupt request.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_EV_IRQHandler

Function name

void HAL_I3C_EV_IRQHandler (I3C_HandleTypeDef * hi3c)

Function description

This function handles I3C event interrupt request.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **None:**

HAL_I3C_Ctrl_BusCharacteristicConfig

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_BusCharacteristicConfig (I3C_HandleTypeDef * hi3c, const LL_I3C_CtrlBusConfTypeDef * pConfig)

Function description

Configure the Controller Bus characterics.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN] Pointer to an LL_I3C_CtrlBusConfTypeDef structure contains controller bus configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_BusCharacteristicConfig

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_BusCharacteristicConfig (I3C_HandleTypeDef * hi3c, const LL_I3C_TgtBusConfTypeDef * pConfig)

Function description

Configure the target Bus characterics.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN] Pointer to an LL_I3C_TgtBusConfTypeDef structure contains target bus configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_SetConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_SetConfigFifo (I3C_HandleTypeDef * hi3c, const I3C_FifoConfTypeDef * pConfig)

Function description

Set I3C FIFOs configuration.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN] Pointer to an I3C_FifoConfTypeDef structure contains FIFOs configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_Config

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Config (I3C_HandleTypeDef * hi3c, const I3C_CtrlConfTypeDef * pConfig)

Function description

Set I3C controller configuration.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN] Pointer to an I3C_CtrlConfTypeDef structure that contains controller configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- This function is called only when the I3C instance is initialized as controller.

HAL_I3C_Tgt_Config

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Config (I3C_HandleTypeDef * hi3c, const I3C_TgtConfTypeDef * pConfig)

Function description

Set I3C target configuration.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN] Pointer to an I3C_TgtConfTypeDef structure that contains target configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- This function is called only when the I3C instance is initialized as target.

HAL_I3C_Ctrl_ConfigBusDevices

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ConfigBusDevices (I3C_HandleTypeDef * hi3c, const I3C_DeviceConfTypeDef * pDesc, uint8_t nbDevice)

Function description

Set I3C bus devices configuration.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pDesc:** : [IN] Pointer to an I3C_DeviceConfTypeDef descriptor that contains the bus devices configurations.
- **nbDevice:** : [IN] Value specifies the number of devices to be treated. This parameter must be a number between Min_Data=1U and Max_Data=4U.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- This function is called only when the I3C instance is initialized as controller. This function can be called by the controller application to help the automatic treatment when target have capability of IBI and/or Control-Role.

HAL_I3C_AddDescToFrame

Function name

HAL_StatusTypeDef HAL_I3C_AddDescToFrame (I3C_HandleTypeDef * hi3c, const I3C_CCCTypeDef * pCCCDesc, const I3C_PrivateTypeDef * pPrivateDesc, I3C_XferTypeDef * pXferData, uint8_t nbFrame, uint32_t option)

Function description

Add Private or CCC descriptor in the user data transfer descriptor.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pCCCDesc:** : [IN] Pointer to an I3C_CCCTypeDef structure that contains the CCC descriptor information.
- **pPrivateDesc:** : [IN] Pointer to an I3C_PrivateTypeDef structure that contains the transfer descriptor.
- **pXferData:** : [IN/OUT] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer).
- **nbFrame:** : [IN] The number of CCC commands or the number of device to treat.
- **option:** : [IN] Value indicates the transfer option. It can be one value of OPTION DEFINITION

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- This function must be called before initiate any communication transfer. This function help the preparation of the full transfer usecase in a transfer descriptor which contained different buffer pointers and their associated size through I3C_XferTypeDef.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.

HAL_I3C_FlushAllFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushAllFifo (I3C_HandleTypeDef * hi3c)

Function description

Flush all I3C FIFOs content.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_FlushTxFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushTxFifo (I3C_HandleTypeDef * hi3c)

Function description

Flush I3C Tx FIFO content.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushRxFifo (I3C_HandleTypeDef * hi3c)

Function description

Flush I3C Rx FIFO content.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_FlushControlFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushControlFifo (I3C_HandleTypeDef * hi3c)

Function description

Flush I3C control FIFO content.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_FlushStatusFifo

Function name

HAL_StatusTypeDef HAL_I3C_FlushStatusFifo (I3C_HandleTypeDef * hi3c)

Function description

Flush I3C status FIFO content.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_ClearConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_ClearConfigFifo (I3C_HandleTypeDef * hi3c)

Function description

Clear I3C FIFOs configuration and set it to default values.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_GetConfigFifo

Function name

HAL_StatusTypeDef HAL_I3C_GetConfigFifo (I3C_HandleTypeDef * hi3c, I3C_FifoConfTypeDef * pConfig)

Function description

Get I3C FIFOs current configuration.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pConfig:** : [IN/OUT] Pointer to an I3C_FifoConfTypeDef structure that returns current FIFOs configuration.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_TransmitCCC

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Controller transmit direct write or a broadcast CCC command in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.

HAL_I3C_Ctrl_TransmitCCC_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller transmit direct write or a broadcast CCC command in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.

HAL_I3C_Ctrl_TransmitCCC_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_TransmitCCC_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller transmit direct write or a broadcast CCC command in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.

HAL_I3C_Ctrl_ReceiveCCC

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Controller transmit direct read CCC command in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_ReceiveCCC_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller transmit direct read CCC command in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_ReceiveCCC_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_ReceiveCCC_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller transmit direct read CCC command in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.

HAL_I3C_Ctrl_Transmit

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Controller private write in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.
- The TxBuf.Size must be equal to the sum of all TxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller private write in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.
- The TxBuf.Size must be equal to the sum of all TxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Transmit_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller private write in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.
- The TxBuf.Size must be equal to the sum of all TxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_Receive

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Controller private read in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_Receive_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller private read in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_Receive_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller private read in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers (control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_MultipleTransfer_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_MultipleTransfer_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller multiple Direct CCC Command, I3C private or I2C transfer in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmit and receive buffers (control buffer, data buffers and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.
- This function must be called to transfer read/write I3C or I2C private data or a direct read/write CCC.
- The TxBuf.Size must be equal to the sum of all TxBuf.Size exist in the descriptor.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.

HAL_I3C_Ctrl_MultipleTransfer_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_MultipleTransfer_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Controller multiple Direct CCC Command, I3C private or I2C transfer in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmit and receive buffers(control buffer, data buffer and status buffer). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- The function HAL_I3C_AddDescToFrame() must be called before initiate a transfer.
- The Tx FIFO threshold HAL_I3C_TXFIFO_THRESHOLD_4_4 is not allowed when the transfer descriptor contains multiple transmission frames.
- The TxBuf.Size must be equal to the sum of all TxBuf.Size exist in the descriptor.
- The RxBuf.Size must be equal to the sum of all RxBuf.Size exist in the descriptor.
- This function must be called to transfer read/write private data or a direct read/write CCC command.

HAL_I3C_Ctrl_SetDynAddr

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_SetDynAddr (I3C_HandleTypeDef * hi3c, uint8_t devAddress)

Function description

Controller set dynamic address.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **devAddress:** : [IN] Value of the dynamic address to be assigned.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_DynAddrAssign_IT

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_DynAddrAssign_IT (I3C_HandleTypeDef * hi3c, uint32_t dynOption)

Function description

Controller assign dynamic address (send a broadcast ENTDAA CCC command) in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **dynOption:** : [IN] Parameter indicates the Dynamic address assignment option. It can be one value of I3C DYNAMIC ADDRESS OPTION DEFINITION.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_DynAddrAssign

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_DynAddrAssign (I3C_HandleTypeDef * hi3c, uint64_t * target_payload, uint32_t dynOption, uint32_t timeout)

Function description

Controller assign dynamic address (send a broadcast ENTDAA CCC command) in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **target_payload:** : [IN/OUT] Pointer to the returned target payload value.
- **dynOption:** : [IN] Parameter indicates the Dynamic address assignment option. It can be one value of I3C DYNAMIC ADDRESS OPTION DEFINITION.
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_IsDeviceI3C_Ready

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_IsDeviceI3C_Ready (I3C_HandleTypeDef * hi3c, uint8_t devAddress, uint32_t trials, uint32_t timeout)

Function description

Check if I3C target device is ready for communication.

Parameters

- **hi3c:** : [IN] Pointer to a I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **devAddress:** : [IN] Value of the device dynamic address.
- **trials:** : [IN] Number of trials
- **timeout:** : [IN] Timeout duration

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Ctrl_IsDeviceI2C_Ready

Function name

HAL_StatusTypeDef HAL_I3C_Ctrl_IsDeviceI2C_Ready (I3C_HandleTypeDef * hi3c, uint8_t devAddress, uint32_t trials, uint32_t timeout)

Function description

Check if I2C target device is ready for communication.

Parameters

- **hi3c:** : [IN] Pointer to a I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **devAddress:** : [IN] Value of the device dynamic address.
- **trials:** : [IN] Number of trials
- **timeout:** : [IN] Timeout duration

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_Transmit

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Target transmit private data in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers information (Pointer to the Tx buffer (TxBuf.pBuffer) and size of data to transmit in bytes (TxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

Notes

- Target FIFO preload data is forced within this API for timing purpose.

HAL_I3C_Tgt_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Target transmit private data in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers information (Pointer to the Tx buffer (TxBuf.pBuffer) and size of data to transmit in bytes (TxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Transmit_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Target transmit private data in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required transmission buffers information (Pointer to the Tx buffer (TxBuf.pBuffer) and size of data to transmit in bytes (TxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_Receive

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData, uint32_t timeout)

Function description

Target receive private data in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers information (Pointer to the Rx buffer (RxBuf.pBuffer) and size of data to be received in bytes (RxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_Receive_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive_IT (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Target receive private data in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers information (Pointer to the Rx buffer (RxBuf.pBuffer) and size of data to be received in bytes (RxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_Receive_DMA (I3C_HandleTypeDef * hi3c, I3C_XferTypeDef * pXferData)

Function description

Target receive private data in DMA mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pXferData:** : [IN] Pointer to an I3C_XferTypeDef structure that contains required reception buffers information (Pointer to the Rx buffer (RxBuf.pBuffer) and size of data to be received in bytes (RxBuf.Size)). This value contain transfer data after called HAL_I3C_AddDescToFrame().

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_ControlRoleReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_ControlRoleReq (I3C_HandleTypeDef * hi3c, uint32_t timeout)

Function description

Target send control role request in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_ControlRoleReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_ControlRoleReq_IT (I3C_HandleTypeDef * hi3c)

Function description

Target send control role request in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_HotJoinReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_HotJoinReq (I3C_HandleTypeDef * hi3c, uint8_t * pAddress, uint32_t timeout)

Function description

Target send hot join request in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pAddress:** : [IN/OUT] Pointer to the target own dynamic address assigned by the controller.
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_HotJoinReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_HotJoinReq_IT (I3C_HandleTypeDef * hi3c)

Function description

Target send hot join request in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_IBIReq

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_IBIReq (I3C_HandleTypeDef * hi3c, const uint8_t * pPayload, uint8_t payloadSize, uint32_t timeout)

Function description

Target send IBI request in polling mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pPayload:** : [IN] Pointer to the buffer contains the payload data.
- **payloadSize:** : [IN] Payload buffer size in bytes.
- **timeout:** : [IN] Timeout duration in millisecond.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Tgt_IBIReq_IT

Function name

HAL_StatusTypeDef HAL_I3C_Tgt_IBIReq_IT (I3C_HandleTypeDef * hi3c, const uint8_t * pPayload, uint8_t payloadSize)

Function description

Target send IBI request in interrupt mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **pPayload:** : [IN] Pointer to the buffer contains the payload data.
- **payloadSize:** : [IN] Payload buffer size in bytes.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_Abort_IT

Function name

HAL_StatusTypeDef HAL_I3C_Abort_IT (I3C_HandleTypeDef * hi3c)

Function description

Abort an I3C IT or DMA process communication with Interrupt.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Status : Value from HAL_StatusTypeDef enumeration.

HAL_I3C_GetState

Function name

HAL_I3C_StateTypeDef HAL_I3C_GetState (const I3C_HandleTypeDef * hi3c)

Function description

Return the I3C handle state.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** State : [OUT] Value from HAL_I3C_StateTypeDef enumeration.

HAL_I3C_GetMode

Function name

HAL_I3C_ModeTypeDef HAL_I3C_GetMode (const I3C_HandleTypeDef * hi3c)

Function description

Returns the I3C handle mode.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **HAL:** Mode : [OUT] Value from HAL_I3C_ModeTypeDef enumeration.

HAL_I3C_GetError

Function name

uint32_t HAL_I3C_GetError (const I3C_HandleTypeDef * hi3c)

Function description

Return the I3C error code.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.

Return values

- **I3C:** Error Code : [OUT] Value from ERROR CODE DEFINITION.

HAL_I3C_GetCCCInfo

Function name

HAL_StatusTypeDef HAL_I3C_GetCCCInfo (I3C_HandleTypeDef * hi3c, uint32_t notifyId, I3C_CCCInfoTypeDef * pCCCInfo)

Function description

Target/Controller Get Common Command Code Information updated after event.

Parameters

- **hi3c:** : [IN] Pointer to an I3C_HandleTypeDef structure that contains the configuration information for the specified I3C.
- **notifyId:** : [IN] Parameter indicates which notification is signaled. It can be a combination of value of I3C Notification ID definition.
- **pCCCInfo:** : [IN/OUT] Pointer to an I3C_CCCInfoTypeDef structure that contains the CCC information updated after CCC event.

Return values

- **None:**

39.3 I3C Firmware driver defines

The following section lists the various define and macros of the module.

39.3.1 I3C

I3C

I3C BCR IN PAYLOAD

HAL_I3C_BCR_IN_PAYLOAD_SHIFT

BCR field in target payload

LL_I3C_BCR_IN_PAYLOAD_SHIFT

BCR field in target payload

I3C COMMON INTERRUPT

HAL_I3C_IT_TXFNFIE

Tx FIFO not full interrupt enable

HAL_I3C_IT_RXFNEIE

Rx FIFO not empty interrupt enable

HAL_I3C_IT_FCIE

Frame complete interrupt enable

HAL_I3C_IT_ERRIE

Error interrupt enable

HAL_I3C_ALL_COMMON_ITS

I3C CONTROLLER INTERRUPT

HAL_I3C_IT_CFNFIE

Control FIFO not full interrupt enable

HAL_I3C_IT_SFNEIE

Status FIFO not empty interrupt enable

HAL_I3C_IT_HJIE

Hot-join interrupt enable

HAL_I3C_IT_CRIE

Controller-role request interrupt enable

HAL_I3C_IT_IBIIE

IBI request interrupt enable

HAL_I3C_IT_RXTGTENDIE

Target-initiated read end interrupt enable

HAL_I3C_ALL_CTRL_ITS

CONTROL FIFO STATE

HAL_I3C_CONTROLFIFO_DISABLE

Control FIFO mode disable

HAL_I3C_CONTROLFIFO_ENABLE

Control FIFO mode enable

DIRECTION

HAL_I3C_DIRECTION_WRITE

Write transfer

HAL_I3C_DIRECTION_READ

Read transfer

HAL_I3C_DIRECTION_BOTH

Read and Write transfer

I3C DYNAMIC ADDRESS OPTION DEFINITION

I3C_RSTDAA_THEN_ENTDAA

Initiate a RSTDAA before a ENTDAA procedure

I3C_ONLY_ENTDAA

Initiate a ENTDAA without RSTDAA

ERROR CODE DEFINITION

HAL_I3C_ERROR_NONE

No error

HAL_I3C_ERROR_CE0

Controller detected an illegally formatted CCC

HAL_I3C_ERROR_CE1

Controller detected that transmitted data on the bus is different than expected

HAL_I3C_ERROR_CE2

Controller detected that broadcast address 7'h7E has been nacked

HAL_I3C_ERROR_CE3

Controller detected that new Controller did not drive the bus after Controller-role handoff

HAL_I3C_ERROR_TE0

Target detected an invalid broadcast address

HAL_I3C_ERROR_TE1

Target detected an invalid CCC Code

HAL_I3C_ERROR_TE2

Target detected a parity error during a write data

HAL_I3C_ERROR_TE3

Target detected a parity error on assigned address during dynamic address arbitration

HAL_I3C_ERROR_TE4

Target detected 7'h7E missing after Restart during Dynamic Address Assignment procedure

HAL_I3C_ERROR_TE5

Target detected an illegally formatted CCC

HAL_I3C_ERROR_TE6

Target detected that transmitted data on the bus is different than expected

HAL_I3C_ERROR_DATA_HAND_OFF

I3C data error during controller-role hand-off process

HAL_I3C_ERROR_DATA_NACK

I3C data not acknowledged error

HAL_I3C_ERROR_ADDRESS_NACK

I3C address not acknowledged error

HAL_I3C_ERROR_COVR

I3C S FIFO Over-Run or C FIFO Under-Run error

HAL_I3C_ERROR_DOVR

I3C Rx FIFO Over-Run or Tx FIFO Under-Run error

HAL_I3C_ERROR_STALL

I3C SCL stall error

HAL_I3C_ERROR_DMA

DMA transfer error

HAL_I3C_ERROR_TIMEOUT

Timeout error

HAL_I3C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I3C_ERROR_INVALID_PARAM

Invalid Parameters error

HAL_I3C_ERROR_SIZE

I3C size management error

HAL_I3C_ERROR_NOT_ALLOWED

I3C operation is not allowed

HAL_I3C_ERROR_DYNAMIC_ADDR

I3C dynamic address error

HAL_I3C_ERROR_INVALID_CALLBACK

Invalid Callback error

I3C Exported Macros

__HAL_I3C_RESET_HANDLE_STATE

Description:

- Reset I3C handle state.

Parameters:

- __HANDLE__: specifies the I3C Handle.

Return value:

- None

__HAL_I3C_ENABLE_IT

Description:

- Enable the specified I3C interrupt.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one value or a combination of the following group's values:
  - I3C CONTROLLER INTERRUPT
  - I3C TARGET INTERRUPT
  - I3C COMMON INTERRUPT

Return value:

- None

__HAL_I3C_DISABLE_IT

Description:

- Disable the specified I3C interrupt.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one value or a combination of the following group's values:
  - I3C CONTROLLER INTERRUPT
  - I3C TARGET INTERRUPT
  - I3C COMMON INTERRUPT

Return value:

- None

__HAL_I3C_GET_FLAG

Description:

- Check whether the specified I3C flag is set or not.

Parameters:

- __HANDLE__: specifies the I3C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one value of the group
  - I3C FLAGS values.

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I3C_GET_BCR

Description:

- Get Bus Characterics in payload (64bits) receive during ENTDAA procedure.

Parameters:

- __PAYLOAD__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00(uint64_t) and Max_Data=0xFFFFFFFFFFFFFFFF.

Return value:

- The: value of BCR Return value between Min_Data=0x00 and Max_Data=0xFF.

__HAL_I3C_GET_IBI_CAPABLE

Description:

- Check IBI request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of IBI request capabilities (ENABLE or DISABLE).

__HAL_I3C_GET_IBI_PAYLOAD

Description:

- Check IBI additional data byte capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of IBI additional data byte capabilities (ENABLE or DISABLE).

__HAL_I3C_GET_CR_CAPABLE

Description:

- Check Controller role request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- The: state of Controller role request capabilities (ENABLE or DISABLE).

I3C FLAGS

HAL_I3C_FLAG_CFEF

Control FIFO not empty flag

HAL_I3C_FLAG_TXFEF

Tx FIFO empty flag

HAL_I3C_FLAG_CFNFF

Control FIFO not full flag

HAL_I3C_FLAG_SFNEF

Status FIFO not empty flag

HAL_I3C_FLAG_TXFNFF

Tx FIFO not full flag

HAL_I3C_FLAG_RXFNEF

Rx FIFO not empty flag

HAL_I3C_FLAG_RXLASTF

Last read data byte/word flag

HAL_I3C_FLAG_TXLASTF

Last written data byte/word flag

HAL_I3C_FLAG_FCF

Frame complete flag

HAL_I3C_FLAG_RXTGTENDF

Target-initiated read end flag

HAL_I3C_FLAG_ERRF

Error flag

HAL_I3C_FLAG_IBIF

IBI request flag

HAL_I3C_FLAG_IBIENDF

IBI end flag

HAL_I3C_FLAG_CRF

Controller-role request flag

HAL_I3C_FLAG_CRUPDF

Controller-role update flag

HAL_I3C_FLAG_HJF

Hot-join flag

HAL_I3C_FLAG_WKPF

Wakeup flag

HAL_I3C_FLAG_GETF

GETxxx CCC flag

HAL_I3C_FLAG_STAF

Format 1 GETSTATUS CCC flag

HAL_I3C_FLAG_DAUPDF

ENTDAA/RSTDAA/SETNEWDA CCC flag

HAL_I3C_FLAG_MWLUPDF

SETMWL CCC flag

HAL_I3C_FLAG_MRLUPDF

SETMRL CCC flag

HAL_I3C_FLAG_RSTF

Reset pattern flag

HAL_I3C_FLAG_ASUPDF

ENTASx CCC flag

HAL_I3C_FLAG_INTUPDF

ENEC/DISEC CCC flag

HAL_I3C_FLAG_DEFF

DEFTGTS CCC flag

HAL_I3C_FLAG_GRPF

DEFGRPA CCC flag

GETMXDS FORMAT

HAL_I3C_GETMXDS_FORMAT_1

GETMXDS CCC Format 1 is used, no MaxRdTurn field in response

HAL_I3C_GETMXDS_FORMAT_2_LSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, LSB = RDTURN[7:0]

HAL_I3C_GETMXDS_FORMAT_2_MID

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, Middle byte = RDTURN[7:0]

HAL_I3C_GETMXDS_FORMAT_2_MSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, MSB = RDTURN[7:0]

HANDOFF ACTIVITY STATE

HAL_I3C_HANDOFF_ACTIVITY_STATE_0

Activity state 0 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_1

Activity state 1 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_2

Activity state 2 after handoff

HAL_I3C_HANDOFF_ACTIVITY_STATE_3

Activity state 3 after handoff

OPTION DEFINITION

I3C_DIRECT_WITH_DEFBYTE_RESTART

Restart between each Direct Command then Stop request for last command. Each Command have an associated defining byte

I3C_DIRECT_WITH_DEFBYTE_STOP

Stop between each Direct Command. Each Command have an associated defining byte

I3C_DIRECT_WITHOUT_DEFBYTE_RESTART

Restart between each Direct Command then Stop request for last command. Each Command have not an associated defining byte

I3C_DIRECT_WITHOUT_DEFBYTE_STOP

Stop between each Direct Command. Each Command have not an associated defining byte

I3C_BROADCAST_WITH_DEFBYTE_RESTART

Restart between each Broadcast Command then Stop request for last command. Each Command have an associated defining byte

I3C_BROADCAST_WITH_DEFBYTE_STOP

Stop between each Broadcast Command. Each Command have an associated defining byte

I3C_BROADCAST_WITHOUT_DEFBYTE_RESTART

Restart between each Broadcast Command then Stop request for last command. Each Command have not an associated defining byte

I3C_BROADCAST_WITHOUT_DEFBYTE_STOP

Stop between each Broadcast Command. Each Command have not an associated defining byte

I3C_PRIVATE_WITH_ARB_RESTART

Restart between each I3C Private message then Stop request for last message. Each Message start with an arbitration header after start bit condition

I3C_PRIVATE_WITH_ARB_STOP

Stop between each I3C Private message. Each Message start with an arbitration header after start bit condition

I3C_PRIVATE_WITHOUT_ARB_RESTART

Restart between each I3C message then Stop request for last message. Each Message start with Target address after start bit condition

I3C_PRIVATE_WITHOUT_ARB_STOP

Stop between each I3C Private message. Each Message start with Target address after start bit condition

I2C_PRIVATE_WITH_ARB_RESTART

Restart between each I2C Private message then Stop request for last message. Each Message start with an arbitration header after start bit condition

I2C_PRIVATE_WITH_ARB_STOP

Stop between each I2C Private message. Each Message start with an arbitration header after start bit condition

I2C_PRIVATE_WITHOUT_ARB_RESTART

Restart between each I2C message then Stop request for last message. Each Message start with Target address after start bit condition

I2C_PRIVATE_WITHOUT_ARB_STOP

Stop between each I2C Private message. Each Message start with Target address after start bit condition

OWN ACTIVITY STATE

HAL_I3C_OWN_ACTIVITY_STATE_0

Own Controller Activity state 0

HAL_I3C_OWN_ACTIVITY_STATE_1

Own Controller Activity state 1

HAL_I3C_OWN_ACTIVITY_STATE_2

Own Controller Activity state 2

HAL_I3C_OWN_ACTIVITY_STATE_3

Own Controller Activity state 3

PAYLOAD SIZE

HAL_I3C_PAYLOAD_EMPTY

Empty payload, no additional data after IBI acknowledge

HAL_I3C_PAYLOAD_1_BYTE

One additional data byte after IBI acknowledge

HAL_I3C_PAYLOAD_2_BYTES

Two additional data bytes after IBI acknowledge

HAL_I3C_PAYLOAD_3_BYTES

Three additional data bytes after IBI acknowledge

HAL_I3C_PAYLOAD_4_BYTES

Four additional data bytes after IBI acknowledge

RX FIFO THRESHOLD

HAL_I3C_RXFIFO_THRESHOLD_1_4

Rx Fifo Threshold is 1 byte

HAL_I3C_RXFIFO_THRESHOLD_4_4

Rx Fifo Threshold is 4 bytes

SDA HOLD TIME

HAL_I3C_SDA_HOLD_TIME_0_5

SDA hold time equal to 0.5 x ti3cclk

HAL_I3C_SDA_HOLD_TIME_1_5

SDA hold time equal to 1.5 x ti3cclk

STATUS FIFO STATE

HAL_I3C_STATUSFIFO_DISABLE

Status FIFO mode disable

HAL_I3C_STATUSFIFO_ENABLE

Status FIFO mode enable

I3C TARGET INTERRUPT

HAL_I3C_IT_IBIENDIE

IBI end interrupt enable

HAL_I3C_IT_CRUPDIE

controller-role update interrupt enable

HAL_I3C_IT_WKPIE

wakeup interrupt enable

HAL_I3C_IT_GETIE

GETxxx CCC interrupt enable

HAL_I3C_IT_STAIE

GETSTATUS CCC interrupt enable

HAL_I3C_IT_DAUPDIE

ENTDAA/RSTDAA/SETNEWDA CCC interrupt enable

HAL_I3C_IT_MWLUPDIE

SETMWL CCC interrupt enable

HAL_I3C_IT_MRLUPDIE

SETMRL CCC interrupt enable

HAL_I3C_IT_RSTIE

reset pattern interrupt enable

HAL_I3C_IT_ASUPDIE

ENTASx CCC interrupt enable

HAL_I3C_IT_INTUPDIE

ENEC/DISEC CCC interrupt enable

HAL_I3C_IT_DEFIE

DEFTGTS CCC interrupt enable

HAL_I3C_IT_GRPIE

DEFGRPA CCC interrupt enable

HAL_I3C_ALL_TGT_ITS

TURNAROUND TIME TSCO

HAL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS

clock-to-data turnaround time tSCO <= 12ns

HAL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

clock-to-data turnaround time tSCO > 12ns

TX FIFO THRESHOLD

HAL_I3C_TXFIFO_THRESHOLD_1_4

Tx Fifo Threshold is 1 byte

HAL_I3C_TXFIFO_THRESHOLD_4_4

Tx Fifo Threshold is 4 bytes

I3C Utility Bus Type

I3C_PURE_I3C_BUS

Pure I3C bus, no I2C

I3C_MIXED_BUS

Mixed bus I3C and I2C

40 HAL ICACHE Generic Driver

40.1 ICACHE Firmware driver registers structures

40.1.1 ICACHE_RegionConfigTypeDef

ICACHE_RegionConfigTypeDef is defined in the stm32h5xx_hal_icache.h

Data Fields

- ***uint32_t BaseAddress***
- ***uint32_t RemapAddress***
- ***uint32_t Size***
- ***uint32_t TrafficRoute***
- ***uint32_t OutputBurstType***

Field Documentation

- ***uint32_t ICACHE_RegionConfigTypeDef::BaseAddress***
  Configures the Base address of Region i to be remapped
- ***uint32_t ICACHE_RegionConfigTypeDef::RemapAddress***
  Configures the Remap address of Region i to be remapped
- ***uint32_t ICACHE_RegionConfigTypeDef::Size***
  Configures the Region size. This parameter can be a value of ***ICACHE_Region_Size***
- ***uint32_t ICACHE_RegionConfigTypeDef::TrafficRoute***
  Selects the traffic route. This parameter can be a value of ***ICACHE_Traffic_Route***
- ***uint32_t ICACHE_RegionConfigTypeDef::OutputBurstType***
  Selects the output burst type. This parameter can be a value of ***ICACHE_Output_Burst_Type***

40.2 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

40.2.1 ICACHE main features

The Instruction Cache (ICACHE) is introduced on C-AHB code bus of Cortex-M33 processor to improve performance when fetching instruction and data from both internal and external memories. It allows close to zero wait states performance.

- The ICACHE provides two performance counters (Hit and Miss), cache invalidate maintenance operation, error management and TrustZone security support.
- The ICACHE provides additionally the possibility to remap input address falling into up to four memory regions (used to remap aliased code in external memories to the internal Code region, for execution)
  ==========

40.2.2 How to use this driver

The ICACHE HAL driver can be used as follows:

1. Optionally configure the Instruction Cache mode with HAL_ICACHE_ConfigAssociativityMode() if the default configuration does not suit the application requirements.
2. Enable and disable the Instruction Cache with respectively HAL_ICACHE_Enable() and HAL_ICACHE_Disable(). Use HAL_ICACHE_IsEnabled() to get the Instruction Cache status.
3. Initiate the cache maintenance invalidation procedure with either HAL_ICACHE_Invalidate() (blocking mode) or HAL_ICACHE_Invalidate_IT() (interrupt mode). When interrupt mode is used, the callback function HAL_ICACHE_InvalidateCompleteCallback() is called when the invalidate procedure is complete. The function HAL_ICACHE_WaitForInvalidateComplete() may be called to wait for the end of the invalidate procedure automatically initiated when disabling the Instruction Cache with HAL_ICACHE_Disable(). The cache operation is bypassed during the invalidation procedure.

4. Use the performance monitoring counters for Hit and Miss with the following functions: HAL_ICACHE_Monitor_Start(), HAL_ICACHE_Monitor_Stop(), HAL_ICACHE_Monitor_Reset(), HAL_ICACHE_Monitor_GetHitValue() and HAL_ICACHE_Monitor_GetMissValue()
5. Enable and disable up to four regions to remap input address from external memories to the internal Code region for execution with HAL_ICACHE_EnableRemapRegion() and HAL_ICACHE_DisableRemapRegion()

40.2.3 Initialization and control functions

This section provides functions allowing to initialize and control the Instruction Cache (mode, invalidate procedure, performance counters).

This section contains the following APIs:

- ***HAL_ICACHE_ConfigAssociativityMode()***
- ***HAL_ICACHE_DeInit()***
- ***HAL_ICACHE_Enable()***
- ***HAL_ICACHE_Disable()***
- ***HAL_ICACHE_IsEnabled()***
- ***HAL_ICACHE_Invalidate()***
- ***HAL_ICACHE_Invalidate_IT()***
- ***HAL_ICACHE_WaitForInvalidateComplete()***
- ***HAL_ICACHE_Monitor_Start()***
- ***HAL_ICACHE_Monitor_Stop()***
- ***HAL_ICACHE_Monitor_Reset()***
- ***HAL_ICACHE_Monitor_GetHitValue()***
- ***HAL_ICACHE_Monitor_GetMissValue()***

40.2.4 IRQ and callback functions

This section provides functions allowing to handle ICACHE global interrupt and the associated callback functions.

This section contains the following APIs:

- ***HAL_ICACHE_IRQHandler()***
- ***HAL_ICACHE_InvalidateCompleteCallback()***
- ***HAL_ICACHE_ErrorCallback()***

40.2.5 Memory remapped regions functions

This section provides functions allowing to manage the remapping of external memories to internal Code for execution.

This section contains the following APIs:

- ***HAL_ICACHE_EnableRemapRegion()***
- ***HAL_ICACHE_DisableRemapRegion()***

40.2.6 Detailed description of functions

HAL_ICACHE_Enable

Function name

HAL_StatusTypeDef HAL_ICACHE_Enable (void)

Function description

Enable the Instruction Cache.

Return values

- **HAL:** status (HAL_OK)

Notes

- This function always returns HAL_OK even if there is any ongoing cache operation. The Instruction Cache is bypassed until the cache operation completes.

HAL_ICACHE_Disable

Function name

HAL_StatusTypeDef HAL_ICACHE_Disable (void)

Function description

Disable the Instruction Cache.

Return values

- **HAL:** status (HAL_OK/HAL_TIMEOUT)

Notes

- This function waits for the cache being disabled but not for the end of the automatic cache invalidation procedure.

HAL_ICACHE_IsEnabled

Function name

uint32_t HAL_ICACHE_IsEnabled (void)

Function description

Check whether the Instruction Cache is enabled or not.

Return values

- **Status:** (0: disabled, 1: enabled)

HAL_ICACHE_ConfigAssociativityMode

Function name

HAL_StatusTypeDef HAL_ICACHE_ConfigAssociativityMode (uint32_t AssociativityMode)

Function description

Configure the Instruction Cache cache associativity mode selection.

Parameters

- **AssociativityMode:** Associativity mode selection This parameter can be one of the following values:
  - ICACHE_1WAY 1-way cache (direct mapped cache)
  - ICACHE_2WAYS 2-ways set associative cache (default)

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

HAL_ICACHE_DeInit

Function name

HAL_StatusTypeDef HAL_ICACHE_DeInit (void)

Function description

DeInitialize the Instruction Cache.

Return values

- **HAL:** status (HAL_OK/HAL_TIMEOUT)

HAL_ICACHE_Invalidate

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate (void)

Function description

Invalidate the Instruction Cache.

Return values

- **HAL:** status (HAL_OK/HAL_ERROR/HAL_TIMEOUT)

Notes

- This function waits for the end of cache invalidation procedure and clears the associated BSYENDF flag.

HAL_ICACHE_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate_IT (void)

Function description

Invalidate the Instruction Cache with interrupt.

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

Notes

- This function launches cache invalidation and returns. User application shall resort to interrupt generation to check the end of the cache invalidation with the BSYENDF flag and the HAL_ICACHE_InvalidateCompleteCallback() callback.

HAL_ICACHE_WaitForInvalidateComplete

Function name

HAL_StatusTypeDef HAL_ICACHE_WaitForInvalidateComplete (void)

Function description

Wait for the end of the Instruction Cache invalidate procedure.

Return values

- **HAL:** status (HAL_OK/HAL_TIMEOUT)

Notes

- This function checks and clears the BSYENDF flag when set.

HAL_ICACHE_Monitor_Start

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Start (uint32_t MonitorType)

Function description

Start the Instruction Cache performance monitoring.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  - ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  - ICACHE_MONITOR_HIT Hit monitoring
  - ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

HAL_ICACHE_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Stop (uint32_t MonitorType)

Function description

Stop the Instruction Cache performance monitoring.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  - ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  - ICACHE_MONITOR_HIT Hit monitoring
  - ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

Notes

- Stopping the monitoring does not reset the values.

HAL_ICACHE_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Reset (uint32_t MonitorType)

Function description

Reset the Instruction Cache performance monitoring values.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  - ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  - ICACHE_MONITOR_HIT Hit monitoring
  - ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

HAL_ICACHE_Monitor_GetHitValue

Function name

uint32_t HAL_ICACHE_Monitor_GetHitValue (void)

Function description

Get the Instruction Cache performance Hit monitoring value.

Return values

- **Hit:** monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_ICACHE_Monitor_GetMissValue

Function name

uint32_t HAL_ICACHE_Monitor_GetMissValue (void)

Function description

Get the Instruction Cache performance Miss monitoring value.

Return values

- **Miss:** monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_ICACHE_IRQHandler

Function name

void HAL_ICACHE_IRQHandler (void)

Function description

Handle the Instruction Cache interrupt request.

Return values

- **None:**

Notes

- This function should be called under the ICACHE_IRQHandler().
- This function respectively disables the interrupt and clears the flag of any pending flag before calling the associated user callback.

HAL_ICACHE_InvalidateCompleteCallback

Function name

void HAL_ICACHE_InvalidateCompleteCallback (void)

Function description

Cache invalidation complete callback.

HAL_ICACHE_ErrorCallback

Function name

void HAL_ICACHE_ErrorCallback (void)

Function description

Error callback.

HAL_ICACHE_EnableRemapRegion

Function name

HAL_StatusTypeDef HAL_ICACHE_EnableRemapRegion (uint32_t Region, const ICACHE_RegionConfigTypeDef *const pRegionConfig)

Function description

Configure and enable a region for memory remapping.

Parameters

- **Region:** Region number This parameter can be a value of
  - Remapped Region number
- **pRegionConfig:** Pointer to structure of ICACHE region configuration parameters

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

Notes

- The Instruction Cache and the region must be disabled.

HAL_ICACHE_DisableRemapRegion

Function name

HAL_StatusTypeDef HAL_ICACHE_DisableRemapRegion (uint32_t Region)

Function description

Disable the memory remapping for a predefined region.

Parameters

- **Region:** Region number This parameter can be a value of
  - Remapped Region number

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

40.3 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 ICACHE

ICACHE

Flags

ICACHE_FLAG_BUSY

Busy flag

ICACHE_FLAG_BUSYEND

Busy end flag

ICACHE_FLAG_ERROR

Cache error flag

Flags and Interrupts Management

__HAL_ICACHE_ENABLE_IT

Description:

- Enable ICACHE interrupts.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt sources to be enabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_DISABLE_IT

Description:

- Disable ICACHE interrupts.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt sources to be disabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_GET_IT_SOURCE

Description:

- Check whether the specified ICACHE interrupt source is enabled or not.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt source to check. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

Return value:

- The: state of __INTERRUPT__ (0 or 1).

__HAL_ICACHE_GET_FLAG

Description:

- Check whether the selected ICACHE flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - ICACHE_FLAG_BUSY Busy flag
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Return value:

- The: state of __FLAG__ (0 or 1).

__HAL_ICACHE_CLEAR_FLAG

Description:

- Clear the selected ICACHE flags.

Parameters:

- __FLAG__: specifies the ICACHE flags to clear. This parameter can be any combination of the following values:
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Interrupts

ICACHE_IT_BUSYEND

Busy end interrupt

ICACHE_IT_ERROR

Cache error interrupt

Monitor type

ICACHE_MONITOR_HIT_MISS

Hit & Miss monitoring

ICACHE_MONITOR_HIT

Hit monitoring

ICACHE_MONITOR_MISS

Miss monitoring

Remapped Output burst type

ICACHE_OUTPUT_BURST_WRAP

WRAP

ICACHE_OUTPUT_BURST_INCR

INCR

Remapped Region number

ICACHE_REGION_0

Region 0

ICACHE_REGION_1

Region 1

ICACHE_REGION_2

Region 2

ICACHE_REGION_3

Region 3

Remapped Region size

ICACHE_REGIONSIZE_2MB

Region size 2MB

ICACHE_REGIONSIZE_4MB

Region size 4MB

ICACHE_REGIONSIZE_8MB

Region size 8MB

ICACHE_REGIONSIZE_16MB

Region size 16MB

ICACHE_REGIONSIZE_32MB

Region size 32MB

ICACHE_REGIONSIZE_64MB

Region size 64MB

ICACHE_REGIONSIZE_128MB

Region size 128MB

Remapped Traffic route

ICACHE_MASTER1_PORT

Master1 port

ICACHE_MASTER2_PORT

Master2 port

Ways selection

ICACHE_1WAY

1-way cache (direct mapped cache)

ICACHE_2WAYS

2-ways set associative cache (default)

41 HAL IRDA Generic Driver

41.1 IRDA Firmware driver registers structures

41.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32h5xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.
  Note:
  – Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***
  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

41.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32h5xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***

- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef __IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***const uint8_t* __IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* __IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t __IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef __IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t __IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code
- ***void(* __IRDA_HandleTypeDef::TxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::TxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Complete Callback

- ***void(* __IRDA_HandleTypeDef::RxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::RxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Complete Callback
- ***void(* __IRDA_HandleTypeDef::ErrorCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Error Callback
- ***void(* __IRDA_HandleTypeDef::AbortCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortTransmitCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Transmit Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortReceiveCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Receive Complete Callback
- ***void(* __IRDA_HandleTypeDef::MspInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp Init callback
- ***void(* __IRDA_HandleTypeDef::MspDeInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp DeInit callback

41.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

41.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   - Enable the USARTx/UARTx interface clock.
   - USARTx/UARTx pins configuration:
     - Enable the clock for the USARTx/UARTx GPIOs.
     - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
     - Configure the USARTx/UARTx interrupt priority.
     - Enable the NVIC USARTx/UARTx IRQ handle.
     - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
   - DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.

4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note: The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note: You can refer to the IRDA HAL driver header file for more useful macros

41.2.2 Callback registration

The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_IRDA_RegisterCallback() to register a user callback. Function HAL_IRDA_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_IRDA_UnRegisterCallback() to reset a callback to the default weak function. HAL_IRDA_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit.

By default, after the HAL_IRDA_Init() and when the state is HAL_IRDA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_IRDA_Init() and HAL_IRDA_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_IRDA_Init() and HAL_IRDA_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_IRDA_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_IRDA_STATE_READY or HAL_IRDA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_IRDA_RegisterCallback() before calling HAL_IRDA_DeInit() or HAL_IRDA_Init() function.

When The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

41.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init()***

- *HAL_IRDA_DeInit()*
- *HAL_IRDA_MspInit()*
- *HAL_IRDA_MspDeInit()*
- *HAL_IRDA_RegisterCallback()*
- *HAL_IRDA_UnRegisterCallback()*

41.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   - Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   - HAL_IRDA_Transmit()
   - HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are :
   - HAL_IRDA_Transmit_IT()
   - HAL_IRDA_Receive_IT()
   - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_IRDA_Transmit_DMA()
   - HAL_IRDA_Receive_DMA()
   - HAL_IRDA_DMAPause()
   - HAL_IRDA_DMAResume()
   - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
   - HAL_IRDA_TxHalfCpltCallback()
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxHalfCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_IRDA_Abort()
   - HAL_IRDA_AbortTransmit()
   - HAL_IRDA_AbortReceive()
   - HAL_IRDA_Abort_IT()
   - HAL_IRDA_AbortTransmit_IT()
   - HAL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_IRDA_AbortCpltCallback()
   - HAL_IRDA_AbortTransmitCpltCallback()
   - HAL_IRDA_AbortReceiveCpltCallback()

8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   – Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.
   – Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:

- *HAL_IRDA_Transmit()*
- *HAL_IRDA_Receive()*
- *HAL_IRDA_Transmit_IT()*
- *HAL_IRDA_Receive_IT()*
- *HAL_IRDA_Transmit_DMA()*
- *HAL_IRDA_Receive_DMA()*
- *HAL_IRDA_DMAPause()*
- *HAL_IRDA_DMAResume()*
- *HAL_IRDA_DMAStop()*
- *HAL_IRDA_Abort()*
- *HAL_IRDA_AbortTransmit()*
- *HAL_IRDA_AbortReceive()*
- *HAL_IRDA_Abort_IT()*
- *HAL_IRDA_AbortTransmit_IT()*
- *HAL_IRDA_AbortReceive_IT()*
- *HAL_IRDA_IRQHandler()*
- *HAL_IRDA_TxCpltCallback()*
- *HAL_IRDA_TxHalfCpltCallback()*
- *HAL_IRDA_RxCpltCallback()*
- *HAL_IRDA_RxHalfCpltCallback()*
- *HAL_IRDA_ErrorCallback()*
- *HAL_IRDA_AbortCpltCallback()*
- *HAL_IRDA_AbortTransmitCpltCallback()*
- *HAL_IRDA_AbortReceiveCpltCallback()*

41.2.5 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_IRDA_GetState()*
- *HAL_IRDA_GetError()*

41.2.6 Detailed description of functions

HAL_IRDA_Init

Function name

HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DeInit

Function name

HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA peripheral.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_MspInit

Function name

void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_MspDeInit

Function name

void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_RegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID, pIRDA_CallbackTypeDef pCallback)

Function description

Register a User IRDA Callback To be used to override the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_IRDA_ERROR_CB_ID Error Callback ID
  - HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  - HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_IRDA_RegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_UnRegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID)

Function description

Unregister an IRDA callback IRDA callback is redirected to the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_IRDA_ERROR_CB_ID Error Callback ID
  - HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  - HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_IRDA_UnRegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to un-register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive

Function name

HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.
- When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name

HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)

Function description

Pause the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAResume

Function name

HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description

Resume the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_DMAStop

Function name

HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description

Stop the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

Handle IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Half Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Half Transfer complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA error callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Receive Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_GetState

Function name

HAL_IRDA_StateTypeDef HAL_IRDA_GetState (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle state.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** state

HAL_IRDA_GetError

Function name

uint32_t HAL_IRDA_GetError (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle error code.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **IRDA:** Error Code

41.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

41.3.1 IRDA

IRDA

IRDA Clock Prescaler

IRDA_PRESCALER_DIV1

fclk_pres = fclk

IRDA_PRESCALER_DIV2

fclk_pres = fclk/2

IRDA_PRESCALER_DIV4

fclk_pres = fclk/4

IRDA_PRESCALER_DIV6

fclk_pres = fclk/6

IRDA_PRESCALER_DIV8

fclk_pres = fclk/8

IRDA_PRESCALER_DIV10

fclk_pres = fclk/10

IRDA_PRESCALER_DIV12

fclk_pres = fclk/12

IRDA_PRESCALER_DIV16

fclk_pres = fclk/16

IRDA_PRESCALER_DIV32

fclk_pres = fclk/32

IRDA_PRESCALER_DIV64

fclk_pres = fclk/64

IRDA_PRESCALER_DIV128

fclk_pres = fclk/128

IRDA_PRESCALER_DIV256

fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE

IRDA DMA RX disabled

IRDA_DMA_RX_ENABLE

IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE

IRDA DMA TX disabled

IRDA_DMA_TX_ENABLE

IRDA DMA TX enabled

IRDA Error Code Definition

HAL_IRDA_ERROR_NONE

No error

HAL_IRDA_ERROR_PE

Parity error

HAL_IRDA_ERROR_NE

Noise error

HAL_IRDA_ERROR_FE

frame error

HAL_IRDA_ERROR_ORE

Overrun error

HAL_IRDA_ERROR_DMA

DMA transfer error

HAL_IRDA_ERROR_BUSY

Busy Error

HAL_IRDA_ERROR_INVALID_CALLBACK

Invalid Callback error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF
  - IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF Parity Error Clear Flag
  - IRDA_CLEAR_FEF Framing Error Clear Flag
  - IRDA_CLEAR_NEF Noise detected Clear Flag
  - IRDA_CLEAR_OREF OverRun Error Clear Flag
  - IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_REACK

IRDA receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA busy flag

IRDA_FLAG_ABRF

IRDA auto Baud rate flag

IRDA_FLAG_ABRE

IRDA auto Baud rate error

IRDA_FLAG_TXE

IRDA transmit data register empty

IRDA_FLAG_TC

IRDA transmission complete

IRDA_FLAG_RXNE

IRDA read data register not empty

IRDA_FLAG_ORE

IRDA overrun error

IRDA_FLAG_NE

IRDA noise error

IRDA_FLAG_FE

IRDA frame error

IRDA_FLAG_PE

IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK

IRDA Interruptions flags mask

IRDA_CR_MASK

IRDA control register mask

IRDA_CR_POS

IRDA control register position

IRDA_ISR_MASK

IRDA ISR register mask

IRDA_ISR_POS

IRDA ISR register position

IRDA Interrupts Definition

IRDA_IT_PE

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise Error detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA State Code Definition

HAL_IRDA_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_IRDA_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_IRDA_STATE_BUSY

An internal process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_IRDA_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_IRDA_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_IRDA_STATE_ERROR

Error Value is allowed for gState only

IRDA Transfer Mode

IRDA_MODE_RX

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

42 HAL IRDA Extension Driver

42.1 IRDAEx Firmware driver defines

The following section lists the various define and macros of the module.

42.1.1 IRDAEx

IRDAEx

IRDAEx Word Length

IRDA_WORDLENGTH_7B

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

43 HAL IWDG Generic Driver

43.1 IWDG Firmware driver registers structures

43.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32h5xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***
- ***uint32_t EWI***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::EWI***
  Specifies if IWDG Early Wakeup Interrupt is enable or not and the comparator value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF value 0 means that EWI is disabled

43.1.2 __IWDG_HandleTypeDef

__IWDG_HandleTypeDef is defined in the stm32h5xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***IWDG_TypeDef* __IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef __IWDG_HandleTypeDef::Init***
  IWDG required parameters
- ***void(* __IWDG_HandleTypeDef::EwiCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Early WakeUp Interrupt callback
- ***void(* __IWDG_HandleTypeDef::MspInitCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Msp Init callback

43.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

43.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by the Low-Speed Internal clock (LSI) and thus stays active even if the main clock fails.

- Once the IWDG is started, the LSI is forced ON and both cannot be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded into the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake up the CPU from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode: When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~131.04s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32H5xx devices provide the capability to measure the LSI clock frequency (LSI clock is internally connected to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

Default timeout value (necessary for IWDG_SR status register update): Constant LSI_VALUE is defined based on the nominal LSI clock frequency. This frequency being subject to variations as mentioned above, the default timeout value (defined through constant HAL_IWDG_DEFAULT_TIMEOUT below) may become too short or too long. In such cases, this default timeout value can be tuned by redefining the constant LSI_VALUE at user-application level (based, for instance, on the measured LSI clock frequency as explained above).

43.2.2 How to use this driver

1. Register callback to treat Iwdg interrupt and MspInit using HAL_IWDG_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from HAL_IWDG_CallbackIDTypeDef.
   - Provide callback function pointer.
2. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts counting down.
   - Enable write access to configuration registers: IWDG_PR, IWDG_RLR, IWDG_WINR and EWCR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
   - Depending on Early Wakeup Interrupt parameter:
     - If EWI is set to disable, comparator is set to 0, interrupt is disable & flag is clear.
     - Else modify EWCR register, setting comparator value, enable interrupt & clear flag.
   - Wait for status flags to be reset.
3. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

43.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.

- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- ***HAL_IWDG_Init()***
- ***HAL_IWDG_MspInit()***
- ***HAL_IWDG_RegisterCallback()***
- ***HAL_IWDG_UnRegisterCallback()***

43.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***
- ***HAL_IWDG_GetActiveStatus()***
- ***HAL_IWDG_IRQHandler()***
- ***HAL_IWDG_EarlyWakeupCallback()***

43.2.5 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_MspInit

Function name

void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG MSP.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_IWDG_Init function is called again to change parameters.

HAL_IWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_RegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID, pIWDG_CallbackTypeDef pCallback)

Function description

Register a User IWDG Callback To be used instead of the weak predefined callback.

Parameters

- **hiwdg:** IWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_IWDG_MSPINIT_CB_ID MspInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_IWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_UnRegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID)

Function description

Unregister a IWDG Callback IWDG Callback is redirected to the weak predefined callback.

Parameters

- **hiwdg:** IWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_IWDG_MSPINIT_CB_ID MspInit callback ID

Return values

- **status:**

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

Refresh the IWDG.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_GetActiveStatus

Function name

uint32_t HAL_IWDG_GetActiveStatus (const IWDG_HandleTypeDef * hiwdg)

Function description

Get back IWDG running status.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **can:** be one of following value :
  - IWDG_STATUS_DISABLE
  - IWDG_STATUS_ENABLE

Notes

- This API allows to know if IWDG has been started by other master, thread or by hardware.

HAL_IWDG_IRQHandler

Function name

void HAL_IWDG_IRQHandler (IWDG_HandleTypeDef * hiwdg)

Function description

Handle IWDG interrupt request.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_IWDG_Init function with EWIMode set to IWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_IWDG_EarlyWakeupCallback

Function name

void HAL_IWDG_EarlyWakeupCallback (IWDG_HandleTypeDef * hiwdg)

Function description

IWDG Early Wakeup callback.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

43.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

43.3.1 IWDG

IWDG

IWDG Active Status

IWDG_STATUS_DISABLE

IWDG_STATUS_ENABLE

IWDG Early Wakeup Interrupt Mode

IWDG_EWI_DISABLE

EWI Disable

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR, IWDG_WINR and EWCR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

IWDG_PRESCALER_512

IWDG prescaler set to 512

IWDG_PRESCALER_1024

IWDG prescaler set to 1024

IWDG Window option

IWDG_WINDOW_DISABLE

44 HAL LPTIM Generic Driver

44.1 LPTIM Firmware driver registers structures

44.1.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

44.1.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

44.1.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

44.1.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***

- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t Period***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0001 and Max_Data = 0xFFFF.
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***
- ***uint32_t LPTIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). Note: When using repetition counter the UpdateMode field must be set to LPTIM_UPDATE_ENDOFPERIOD otherwise unpredictable behavior may occur. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

44.1.5 LPTIM_OC_ConfigTypeDef

LPTIM_OC_ConfigTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OCPolarity***

Field Documentation

- ***uint32_t LPTIM_OC_ConfigTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t LPTIM_OC_ConfigTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***LPTIM_Output_Compare_Polarity***

44.1.6 LPTIM_IC_ConfigTypeDef

LPTIM_IC_ConfigTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***uint32_t ICInputSource***
- ***uint32_t ICPrescaler***
- ***uint32_t ICPolarity***

- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LPTIM_IC_ConfigTypeDef::ICInputSource***
  Specifies source selected for IC channel. This parameter can be a value of ***LPTIM_Input_Capture_Source***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***LPTIM_Input_Capture_Prescaler***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***LPTIM_Input_Capture_Polarity***
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***LPTIM_Input_Capture_Filter***

44.1.7 __LPTIM_HandleTypeDef

__LPTIM_HandleTypeDef is defined in the stm32h5xx_hal_lptim.h

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_LPTIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***
- ***__IO HAL_LPTIM_ChannelStateTypeDef ChannelState***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* CompareMatchCallback***
- ***void(* AutoReloadMatchCallback***
- ***void(* TriggerCallback***
- ***void(* CompareWriteCallback***
- ***void(* AutoReloadWriteCallback***
- ***void(* DirectionUpCallback***
- ***void(* DirectionDownCallback***
- ***void(* UpdateEventCallback***
- ***void(* RepCounterWriteCallback***
- ***void(* UpdateEventHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* IC_CaptureCallback***
- ***void(* IC_CaptureHalfCpltCallback***
- ***void(* IC_OverCaptureCallback***

Field Documentation

- ***LPTIM_TypeDef* __LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef __LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_LPTIM_ActiveChannel __LPTIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __LPTIM_HandleTypeDef::hdma[3]***
  DMA Handlers array, This array is accessed by a ***LPTIM_DMA_Handle_index***
- ***HAL_StatusTypeDef __LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status

- ***HAL_LockTypeDef __LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef __LPTIM_HandleTypeDef::State***
  LPTIM peripheral state
- ***__IO HAL_LPTIM_ChannelStateTypeDef __LPTIM_HandleTypeDef::ChannelState[2]***
  LPTIM channel operation state
- ***void(* __LPTIM_HandleTypeDef::MspInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp Init Callback
- ***void(* __LPTIM_HandleTypeDef::MspDeInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp DeInit Callback
- ***void(* __LPTIM_HandleTypeDef::CompareMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare match Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload match Callback
- ***void(* __LPTIM_HandleTypeDef::TriggerCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  External trigger event detection Callback
- ***void(* __LPTIM_HandleTypeDef::CompareWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionUpCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Up-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionDownCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Down-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event detection Callback
- ***void(* __LPTIM_HandleTypeDef::RepCounterWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Repetition counter register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventHalfCpltCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event half complete detection Callback
- ***void(* __LPTIM_HandleTypeDef::ErrorCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Error Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Input capture Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __LPTIM_HandleTypeDef *htim)***
  Input Capture half complete Callback
- ***void(* __LPTIM_HandleTypeDef::IC_OverCaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Over capture Callback

44.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

44.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().
2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 or LPTIM6.
   - Clock: the counter clock.
     - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or CSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge : only for hardware trigger.
     - SampleTime : trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity : 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
   - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
   - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

Callback registration

The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_LPTIM_RegisterCallback() to register a callback. HAL_LPTIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_LPTIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_LPTIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- MspInitCallback : LPTIM Base Msp Init Callback.
- MspDeInitCallback : LPTIM Base Msp DeInit Callback.
- CompareMatchCallback : Compare match Callback.
- AutoReloadMatchCallback : Auto-reload match Callback.
- TriggerCallback : External trigger event detection Callback.
- CompareWriteCallback : Compare register write complete Callback.
- AutoReloadWriteCallback : Auto-reload register write complete Callback.
- DirectionUpCallback : Up-counting direction change Callback.
- DirectionDownCallback : Down-counting direction change Callback.
- UpdateEventCallback : Update event detection Callback.
- RepCounterWriteCallback : Repetition counter register write complete Callback.

By default, after the Init and when the state is HAL_LPTIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_LPTIM_TriggerCallback(), HAL_LPTIM_CompareMatchCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init/DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init/DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_LPTIM_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LPTIM_STATE_READY or HAL_LPTIM_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LPTIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

44.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize the LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init()***
- ***HAL_LPTIM_DeInit()***
- ***HAL_LPTIM_MspInit()***
- ***HAL_LPTIM_MspDeInit()***

44.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- *HAL_LPTIM_PWM_Start()*
- *HAL_LPTIM_PWM_Stop()*
- *HAL_LPTIM_PWM_Start_IT()*
- *HAL_LPTIM_PWM_Stop_IT()*
- *HAL_LPTIM_PWM_Start_DMA()*
- *HAL_LPTIM_PWM_Stop_DMA()*
- *HAL_LPTIM_OnePulse_Start()*
- *HAL_LPTIM_OnePulse_Stop()*
- *HAL_LPTIM_OnePulse_Start_IT()*
- *HAL_LPTIM_OnePulse_Stop_IT()*
- *HAL_LPTIM_SetOnce_Start()*
- *HAL_LPTIM_SetOnce_Stop()*
- *HAL_LPTIM_SetOnce_Start_IT()*
- *HAL_LPTIM_SetOnce_Stop_IT()*
- *HAL_LPTIM_Encoder_Start()*
- *HAL_LPTIM_Encoder_Stop()*
- *HAL_LPTIM_Encoder_Start_IT()*
- *HAL_LPTIM_Encoder_Stop_IT()*
- *HAL_LPTIM_TimeOut_Start()*
- *HAL_LPTIM_TimeOut_Stop()*
- *HAL_LPTIM_TimeOut_Start_IT()*
- *HAL_LPTIM_TimeOut_Stop_IT()*
- *HAL_LPTIM_Counter_Start()*
- *HAL_LPTIM_Counter_Stop()*
- *HAL_LPTIM_Counter_Start_IT()*
- *HAL_LPTIM_Counter_Stop_IT()*
- *HAL_LPTIM_IC_Start()*
- *HAL_LPTIM_IC_Stop()*
- *HAL_LPTIM_IC_Start_IT()*
- *HAL_LPTIM_IC_Stop_IT()*
- *HAL_LPTIM_IC_Start_DMA()*
- *HAL_LPTIM_IC_Stop_DMA()*
- *HAL_LPTIM_OC_ConfigChannel()*
- *HAL_LPTIM_IC_ConfigChannel()*

44.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter()*
- *HAL_LPTIM_ReadAutoReload()*
- *HAL_LPTIM_ReadCapturedValue()*
- *HAL_LPTIM_IC_GetOffset()*

44.2.5 LPTIM Config function

This section provides LPTIM Config function.

- Configure channel: Output Compare mode, Period, Polarity.

This section contains the following APIs:

- *HAL_LPTIM_OC_ConfigChannel()*
- *HAL_LPTIM_IC_ConfigChannel()*

44.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_LPTIM_GetState()*

44.2.7 Detailed description of functions

HAL_LPTIM_Init

Function name

HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_OC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_OC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

Parameters

- **hlptim:** LPTIM handle
- **sConfig:** The output configuration structure
- **Channel:** LPTIM Channel to be configured This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

Notes

- Successive calls to HAL_LPTIM_OC_ConfigChannel can only be performed after a delay that must be greater or equal than the value of (PRESC x 3) kernel clock cycles, PRESC[2:0] being the clock decimal division factor (1, 2, 4, ..., 128). Any successive call violating this delay, leads to unpredictable results.

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, const uint32_t * pData, uint32_t Length)

Function description

Start the LPTIM PWM generation in DMA mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected
- **pData:** The destination Buffer address
- **Length:** The length of data to be transferred from LPTIM peripheral to memory

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation in DMA mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM in Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** LPTIM handle
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_IC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

Parameters

- **hlptim:** LPTIM handle
- **sConfig:** The input configuration structure
- **Channel:** LPTIM Channel to be configured This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

Notes

- Successive calls to HAL_LPTIM_IC_ConfigChannel can only be performed after a delay that must be greater or equal than the value of (PRESC x 3) kernel clock cycles, PRESC[2:0] being the clock decimal division factor (1, 2, 4, ..., 128). Any successive call violating this delay, leads to unpredictable results.

HAL_LPTIM_IC_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Starts the LPTIM Input Capture measurement.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Starts the LPTIM Input Capture measurement in interrupt mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement in interrupt mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, uint32_t * pData, uint32_t Length)

Function description

Starts the LPTIM Input Capture measurement in DMA mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected
- **pData:** The destination Buffer address
- **Length:** The length of data to be transferred from LPTIM peripheral to memory

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement in DMA mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current counter value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCapturedValue

Function name

uint32_t HAL_LPTIM_ReadCapturedValue (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Return the current Compare (Pulse) value.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be selected This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **Compare:** value.

HAL_LPTIM_IC_GetOffset

Function name

uint8_t HAL_LPTIM_IC_GetOffset (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

LPTimer Input Capture Get Offset(in counter step unit)

Parameters

- **hlptim:** pointer to a LPTIM_HandleTypeDef structure that contains the configuration information for LPTIM module.
- **Channel:** This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **The:** offset value

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset The Offset value is depending on the glitch filter value for the channel and the value of the prescaler for the kernel clock. Please check Errata Sheet V1_8 for more details under "variable latency on input capture channel" section.

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

Handle LPTIM interrupt request.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_ErrorCallback

Function name

void HAL_LPTIM_ErrorCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Error callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_UpdateEventCallback

Function name

void HAL_LPTIM_UpdateEventCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Repetition counter underflowed (or contains zero) and LPTIM counter overflowed callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_UpdateEventHalfCpltCallback

Function name

void HAL_LPTIM_UpdateEventHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Update event half complete callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_RepCounterWriteCallback

Function name

void HAL_LPTIM_RepCounterWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Successful APB bus write to repetition counter register callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_IC_CaptureCallback

Function name

void HAL_LPTIM_IC_CaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_IC_CaptureHalfCpltCallback

Function name

void HAL_LPTIM_IC_CaptureHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM IC handle

Return values

- **None:**

HAL_LPTIM_IC_OverCaptureCallback

Function name

void HAL_LPTIM_IC_OverCaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Over Capture callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_RegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID, pLPTIM_CallbackTypeDef pCallback)

Function description

Register a User LPTIM callback to be used instead of the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID
  - HAL_LPTIM_UPDATE_EVENT_CB_ID Update event detection Callback ID
  - HAL_LPTIM_REP_COUNTER_WRITE_CB_ID Repetition counter register write complete Callback ID
  - HAL_LPTIM_UPDATE_EVENT_HALF_CB_ID Update event Half detection Callback ID
  - HAL_LPTIM_ERROR_CB_ID Error Callback ID
  - HAL_LPTIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_LPTIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_LPTIM_OVER_CAPTURE_CB_ID Over Capture Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_LPTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_UnRegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a LPTIM callback LLPTIM callback is redirected to the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID
  - HAL_LPTIM_UPDATE_EVENT_CB_ID Update event detection Callback ID
  - HAL_LPTIM_REP_COUNTER_WRITE_CB_ID Repetition counter register write complete Callback ID
  - HAL_LPTIM_UPDATE_EVENT_HALF_CB_ID Update event Half detection Callback ID
  - HAL_LPTIM_ERROR_CB_ID Error Callback ID
  - HAL_LPTIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_LPTIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_LPTIM_OVER_CAPTURE_CB_ID Over Capture Callback ID

Return values

- **status:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the LPTIM handle state.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** state

44.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

44.3.1 LPTIM

LPTIM

LPTIM Channel

LPTIM_CHANNEL_1

Capture/compare channel 1 identifier

LPTIM_CHANNEL_2

Capture/compare channel 2 identifier

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM DMA Handle Index

LPTIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

LPTIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Update event 1 DMA request

LPTIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Update event 2 DMA request

LPTIM DMA Sources

LPTIM_DMA_UPDATE

DMA request is triggered by the update event

LPTIM_DMA_CC1

DMA request is triggered by the capture 1 event

LPTIM_DMA_CC2

DMA request is triggered by the capture 2 event

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

Description:

- Start the LPTIM peripheral in single mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER

Description:

- Reset the LPTIM Counter register in synchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER_AFTERREAD

Description:

- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Write the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

- None

Notes:

- The ARR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value
- __CHANNEL__: TIM Channel to be configured

Return value:

- None

Notes:

- The CCRx registers can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_REPETITIONCOUNTER_SET

Description:

- Write the passed parameter in the Repetition register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Repetition value

Return value:

- None

__HAL_LPTIM_REPETITIONCOUNTER_GET

Description:

- Return the current Repetition value.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- Repetition: register value

Notes:

- The RCR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_CAPTURE_COMPARE_ENABLE

Description:

- Enable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be enabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_CAPTURE_COMPARE_DISABLE

Description:

- Disable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be disabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be enabled when the LPTIM instance is enabled.

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be disabled when the LPTIM instance is enabled.

__HAL_LPTIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the LPTIM DMA request to enable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the LPTIM Handle.
- __DMA__: specifies the LPTIM DMA request to disable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- Interrupt: status.

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_CC1O

LPTIM_FLAG_CC2O

LPTIM_FLAG_CC1

LPTIM_FLAG_CC2

LPTIM_FLAG_CMP1OK

LPTIM_FLAG_CMP2OK

LPTIM_FLAG_DIEROK

LPTIM_FLAG_REPOK

LPTIM_FLAG_UPDATE

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 and LPTIM6

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 and LPTIM6

LPTIM Input Capture Filter

LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

LPTIM Input Capture Polarity

LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

LPTIM Input Capture Prescaler

LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LPTIM_ICPSC_DIV8

Capture performed once every 8 events

LPTIM Input Capture Source

LPTIM_IC1SOURCE_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 and LPTIM6

LPTIM_IC2SOURCE_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 and LPTIM6

LPTIM_IC2SOURCE_LSI

For LPTIM1

LPTIM_IC2SOURCE_LSE

For LPTIM1

LPTIM_IC2SOURCE_HSI_1024

For LPTIM2

LPTIM_IC2SOURCE_CSI_128

For LPTIM2

LPTIM_IC2SOURCE_HSI_8

For LPTIM2

LPTIM Interrupts Definition

LPTIM_IT_CC1O

LPTIM_IT_CC2O

LPTIM_IT_CC1

LPTIM_IT_CC2

LPTIM_IT_CMP1OK

LPTIM_IT_CMP2OK

LPTIM_IT_REPOK

LPTIM_IT_UPDATE

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM Output Compare Polarity

LPTIM_OCPOLARITY_HIGH

Capture/Compare output polarity

LPTIM_OCPOLARITY_LOW

Capture/Compare output polarity

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

45 HAL MMC Generic Driver

45.1 MMC Firmware driver registers structures

45.1.1 HAL_MMC_CardInfoTypeDef

HAL_MMC_CardInfoTypeDef is defined in the stm32h5xx_hal_mmc.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***

Field Documentation

- ***uint32_t HAL_MMC_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_MMC_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_MMC_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes

45.1.2 __MMC_HandleTypeDef

__MMC_HandleTypeDef is defined in the stm32h5xx_hal_mmc.h

Data Fields

- ***MMC_TypeDef * Instance***
- ***MMC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_MMC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_MMC_CardInfoTypeDef MmcCard***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***uint32_t Ext_CSD***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***

- ***void(* AbortCpltCallback***
- ***void(* Read_DMALnkLstBufCpltCallback***
- ***void(* Write_DMALnkLstBufCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MMC_TypeDef* __MMC_HandleTypeDef::Instance***
  MMC registers base address
- ***MMC_InitTypeDef __MMC_HandleTypeDef::Init***
  MMC required parameters
- ***HAL_LockTypeDef __MMC_HandleTypeDef::Lock***
  MMC locking object
- ***const uint8_t* __MMC_HandleTypeDef::pTxBuffPtr***
  Pointer to MMC Tx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::TxXferSize***
  MMC Tx Transfer size
- ***uint8_t* __MMC_HandleTypeDef::pRxBuffPtr***
  Pointer to MMC Rx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::RxXferSize***
  MMC Rx Transfer size
- ***__IO uint32_t __MMC_HandleTypeDef::Context***
  MMC transfer context
- ***__IO HAL_MMC_StateTypeDef __MMC_HandleTypeDef::State***
  MMC card State
- ***__IO uint32_t __MMC_HandleTypeDef::ErrorCode***
  MMC Card Error codes
- ***HAL_MMC_CardInfoTypeDef __MMC_HandleTypeDef::MmcCard***
  MMC Card information
- ***uint32_t __MMC_HandleTypeDef::CSD[4U]***
  MMC card specific data table
- ***uint32_t __MMC_HandleTypeDef::CID[4U]***
  MMC card identification number table
- ***uint32_t __MMC_HandleTypeDef::Ext_CSD[128]***
- ***void(* __MMC_HandleTypeDef::TxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::RxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::ErrorCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::AbortCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspInitCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspDeInitCallback)(struct __MMC_HandleTypeDef *hmmc)***

45.1.3 HAL_MMC_CardCSDTypeDef

HAL_MMC_CardCSDTypeDef is defined in the stm32h5xx_hal_mmc.h

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***
- ***__IO uint8_t TAAC***
- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***

- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_MMC_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved4***
  Always 1

45.1.4 HAL_MMC_CardCIDTypeDef

HAL_MMC_CardCIDTypeDef is defined in the stm32h5xx_hal_mmc.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved2***
  Always 1

45.2 MMC Firmware driver API description

The following section lists the various functions of the MMC library.

45.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_MMC_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the HAL_MMC_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for MMC card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_MMC_ReadBlocks_IT() and HAL_MMC_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_MMC_ENABLE_IT() and __HAL_MMC_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_MMC_GET_IT() and __HAL_MMC_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

MMC Card Initialization and configuration

To initialize the MMC Card, use the HAL_MMC_Init() function. It Initializes SDMMC Peripheral (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the MMC card. The API used is HAL_MMC_InitCard(). This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is compatible with MMC standard. This API (HAL_MMC_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.

MMC Card Read operation

- You can read from MMC card in polling mode by using function HAL_MMC_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can read from MMC card in DMA mode by using function HAL_MMC_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Rx interrupt event.
- You can read from MMC card in Interrupt mode by using function HAL_MMC_ReadBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Rx interrupt event.

MMC Card Write operation

- You can write to MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can write to MMC card in DMA mode by using function HAL_MMC_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Tx interrupt event.
- You can write to MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Tx interrupt event.

MMC card information

- To get MMC card information, you can use the function HAL_MMC_GetCardInfo(). It returns useful information about the MMC card such as block size, card type, block number ...

MMC card CSD register

- The HAL_MMC_GetCardCSD() API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

MMC card CID register

- The HAL_MMC_GetCardCID() API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

MMC HAL driver macros list

Below the list of most used macros in MMC HAL driver.

- __HAL_MMC_ENABLE_IT: Enable the MMC device interrupt
- __HAL_MMC_DISABLE_IT: Disable the MMC device interrupt
- __HAL_MMC_GET_FLAG:Check whether the specified MMC flag is set or not
- __HAL_MMC_CLEAR_FLAG: Clear the MMC's pending flags

Note: You can refer to the MMC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_MMC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_MMC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : MMC MspInit.

- MspDeInitCallback : MMC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_MMC_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : MMC MspInit.
- MspDeInitCallback : MMC MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_MMC_Init and if the state is HAL_MMC_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_MMC_Init and HAL_MMC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_MMC_Init and HAL_MMC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_MMC_RegisterCallback before calling HAL_MMC_DeInit or HAL_MMC_Init function. When The compilation define USE_HAL_MMC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

45.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the MMC card device to be ready for use.

This section contains the following APIs:

- ***HAL_MMC_Init()***
- ***HAL_MMC_InitCard()***
- ***HAL_MMC_DeInit()***
- ***HAL_MMC_MspInit()***
- ***HAL_MMC_MspDeInit()***

45.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to MMC card.

This section contains the following APIs:

- ***HAL_MMC_ReadBlocks()***
- ***HAL_MMC_WriteBlocks()***
- ***HAL_MMC_ReadBlocks_IT()***
- ***HAL_MMC_WriteBlocks_IT()***
- ***HAL_MMC_ReadBlocks_DMA()***
- ***HAL_MMC_WriteBlocks_DMA()***
- ***HAL_MMC_Erase()***
- ***HAL_MMC_IRQHandler()***
- ***HAL_MMC_GetState()***
- ***HAL_MMC_GetError()***
- ***HAL_MMC_TxCpltCallback()***
- ***HAL_MMC_RxCpltCallback()***
- ***HAL_MMC_ErrorCallback()***
- ***HAL_MMC_AbortCallback()***

- *HAL_MMC_RegisterCallback()*
- *HAL_MMC_UnRegisterCallback()*

45.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MMC card operations and get the related information

This section contains the following APIs:

- *HAL_MMC_GetCardCID()*
- *HAL_MMC_GetCardCSD()*
- *HAL_MMC_GetCardInfo()*
- *HAL_MMC_GetCardExtCSD()*
- *HAL_MMC_ConfigWideBusOperation()*
- *HAL_MMC_ConfigSpeedBusOperation()*
- *HAL_MMC_GetCardState()*
- *HAL_MMC_Abort()*
- *HAL_MMC_Abort_IT()*
- *HAL_MMC_EraseSequence()*
- *HAL_MMC_Sanitize()*
- *HAL_MMC_ConfigSecRemovalType()*
- *HAL_MMC_GetSupportedSecRemovalType()*
- *HAL_MMC_SleepDevice()*
- *HAL_MMC_AwakeDevice()*

45.2.5 Detailed description of functions

HAL_MMC_Init

Function name

HAL_StatusTypeDef HAL_MMC_Init (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC according to the specified parameters in the MMC_HandleTypeDef and create the associated handle.

Parameters

- **hmmc:** Pointer to the MMC handle

Return values

- **HAL:** status

HAL_MMC_InitCard

Function name

HAL_StatusTypeDef HAL_MMC_InitCard (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC Card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This function initializes the MMC card. It could be used when a card re-initialization is needed.

HAL_MMC_DeInit

Function name

HAL_StatusTypeDef HAL_MMC_DeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initializes the MMC card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_MspInit

Function name

void HAL_MMC_MspInit (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_MspDeInit

Function name

void HAL_MMC_MspDeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initialize MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of MMC blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of MMC blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Erase

Function name

HAL_StatusTypeDef HAL_MMC_Erase (MMC_HandleTypeDef * hmmc, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given MMC card.

Parameters

- **hmmc:** Pointer to MMC handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Tx interrupt event.

HAL_MMC_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_DMA (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Tx interrupt event.

HAL_MMC_IRQHandler

Function name

void HAL_MMC_IRQHandler (MMC_HandleTypeDef * hmmc)

Function description

This function handles MMC card interrupt request.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_TxCpltCallback

Function name

void HAL_MMC_TxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Tx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_RxCpltCallback

Function name

void HAL_MMC_RxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Rx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_ErrorCallback

Function name

void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC error callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC Abort callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_RegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId, pMMC_CallbackTypeDef pCallback)

Function description

Register a User MMC Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Rx Linked List Node buffer Callback ID
  - HAL_MMC_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Tx Linked List Node buffer Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_MMC_RegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_UnRegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId)

Function description

Unregister a User MMC Callback MMC Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Rx Linked List Node buffer Callback ID
  - HAL_MMC_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Tx Linked List Node buffer Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_MMC_UnRegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hmmc, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hmmc:** Pointer to MMC handle
- **WideMode:** Specifies the MMC card wide bus mode This parameter can be one of the following values:
  - SDMMC_BUS_WIDE_8B: 8-bit data transfer
  - SDMMC_BUS_WIDE_4B: 4-bit data transfer
  - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_MMC_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSpeedBusOperation (MMC_HandleTypeDef * hmmc, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hmmc:** Pointer to the MMC handle
- **SpeedMode:** Specifies the MMC card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed (MMC @ 26MHz)
  - SDMMC_SPEED_MODE_HIGH: High Speed (MMC @ 52 MHz)
  - SDMMC_SPEED_MODE_DDR: High Speed DDR (MMC DDR @ 52 MHz)

Return values

- **HAL:** status

HAL_MMC_GetCardState

Function name

HAL_MMC_CardStateTypeDef HAL_MMC_GetCardState (MMC_HandleTypeDef * hmmc)

Function description

Gets the current mmc card data state.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **Card:** state

HAL_MMC_GetCardCID

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCID (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCID:** Pointer to a HAL_MMC_CIDTypedef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCSD (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCSD:** Pointer to a HAL_MMC_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardInfo

Function name

HAL_StatusTypeDef HAL_MMC_GetCardInfo (MMC_HandleTypeDef * hmmc, HAL_MMC_CardInfoTypeDef * pCardInfo)

Function description

Gets the MMC card info.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCardInfo:** Pointer to the HAL_MMC_CardInfoTypeDef structure that will contain the MMC card status information

Return values

- **HAL:** status

HAL_MMC_GetCardExtCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardExtCSD (MMC_HandleTypeDef * hmmc, uint32_t * pExtCSD, uint32_t Timeout)

Function description

Returns information the information of the card which are stored on the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pExtCSD:** Pointer to a memory area (512 bytes) that contains all Extended CSD register parameters
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_GetState

Function name

HAL_MMC_StateTypeDef HAL_MMC_GetState (const MMC_HandleTypeDef * hmmc)

Function description

return the MMC state

Parameters

- **hmmc:** Pointer to mmc handle

Return values

- **HAL:** state

HAL_MMC_GetError

Function name

uint32_t HAL_MMC_GetError (const MMC_HandleTypeDef * hmmc)

Function description

Return the MMC error code.

Parameters

- **hmmc:** : Pointer to a MMC_HandleTypeDef structure that contains the configuration information.

Return values

- **MMC:** Error Code

HAL_MMC_Abort

Function name

HAL_StatusTypeDef HAL_MMC_Abort (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_Abort_IT

Function name

HAL_StatusTypeDef HAL_MMC_Abort_IT (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC (IT mode).

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_EraseSequence

Function name

HAL_StatusTypeDef HAL_MMC_EraseSequence (MMC_HandleTypeDef * hmmc, uint32_t EraseType, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Perform specific commands sequence for the different type of erase.

Parameters

- **hmmc:** Pointer to MMC handle
- **EraseType:** Specifies the type of erase to be performed This parameter can be one of the following values:
  - HAL_MMC_TRIM Erase the write blocks identified by CMD35 & 36
  - HAL_MMC_ERASE Erase the erase groups identified by CMD35 & 36
  - HAL_MMC_DISCARD Discard the write blocks identified by CMD35 & 36
  - HAL_MMC_SECURE_ERASE Perform a secure purge according SRT on the erase groups identified by CMD35 & 36
  - HAL_MMC_SECURE_TRIM_STEP1 Mark the write blocks identified by CMD35 & 36 for secure erase
  - HAL_MMC_SECURE_TRIM_STEP2 Perform a secure purge according SRT on the write blocks previously identified
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Sanitize

Function name

HAL_StatusTypeDef HAL_MMC_Sanitize (MMC_HandleTypeDef * hmmc)

Function description

Perform sanitize operation on the device.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ConfigSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t SRTMode)

Function description

Configure the Secure Removal Type (SRT) in the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **SRTMode:** Specifies the type of erase to be performed This parameter can be one of the following values:
  - HAL_MMC_SRT_ERASE Information removed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  - HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_GetSupportedSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_GetSupportedSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t * SupportedSRT)

Function description

Gets the supported values of the the Secure Removal Type (SRT).

Parameters

- **hmmc:** pointer to MMC handle
- **SupportedSRT:** pointer for supported SRT value This parameter is a bit field of the following values:
  - HAL_MMC_SRT_ERASE Information removed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  - HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

HAL_MMC_SleepDevice

Function name

HAL_StatusTypeDef HAL_MMC_SleepDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Standby State to Sleep State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_AwakeDevice

Function name

HAL_StatusTypeDef HAL_MMC_AwakeDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Sleep State to Standby State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

45.3 MMC Firmware driver defines

The following section lists the various define and macros of the module.

45.3.1 MMC

MMC

MMC Error status enumeration Structure definition

HAL_MMC_ERROR_NONE

No error

HAL_MMC_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_MMC_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_MMC_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_MMC_ERROR_DATA_TIMEOUT

Data timeout

HAL_MMC_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_MMC_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_MMC_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_MMC_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_MMC_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_MMC_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_MMC_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_MMC_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_MMC_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_MMC_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_MMC_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_MMC_ERROR_CC_ERR

Internal card controller error

HAL_MMC_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_MMC_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_MMC_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_MMC_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_MMC_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_MMC_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_MMC_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_MMC_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_MMC_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_MMC_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_MMC_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_MMC_ERROR_PARAM

the used parameter is not valid

HAL_MMC_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_MMC_ERROR_BUSY

Error when transfer process is busy

HAL_MMC_ERROR_DMA

Error while DMA transfer

HAL_MMC_ERROR_TIMEOUT

Timeout error

HAL_MMC_ERROR_INVALID_CALLBACK

Invalid callback error

MMC context enumeration

MMC_CONTEXT_NONE

None

MMC_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

MMC_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

MMC_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

MMC_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

MMC_CONTEXT_IT

Process in Interrupt mode

MMC_CONTEXT_DMA

Process in DMA mode

MMC Voltage mode

MMC_HIGH_VOLTAGE_RANGE

High voltage in byte mode

MMC_DUAL_VOLTAGE_RANGE

Dual voltage in byte mode

MMC_LOW_VOLTAGE_RANGE

Low voltage in byte mode

EMMC_HIGH_VOLTAGE_RANGE

High voltage in sector mode

EMMC_DUAL_VOLTAGE_RANGE

Dual voltage in sector mode

EMMC_LOW_VOLTAGE_RANGE

Low voltage in sector mode

MMC_INVALID_VOLTAGE_RANGE

MMC Memory Cards

MMC_LOW_CAPACITY_CARD

MMC Card Capacity <=2Gbytes

MMC_HIGH_CAPACITY_CARD

MMC Card Capacity >2Gbytes and <2Tbytes

MMC Erase Type

HAL_MMC_ERASE

Erase the erase groups identified by CMD35 & 36

HAL_MMC_TRIM

Erase the write blocks identified by CMD35 & 36

HAL_MMC_DISCARD

Discard the write blocks identified by CMD35 & 36

HAL_MMC_SECURE_ERASE

Perform a secure purge according SRT on the erase groups identified by CMD35 & 36

HAL_MMC_SECURE_TRIM_STEP1

Mark the write blocks identified by CMD35 & 36 for secure erase

HAL_MMC_SECURE_TRIM_STEP2

Perform a secure purge according SRT on the write blocks previously identified

IS_MMC_ERASE_TYPE

MMC Secure Removal Type

HAL_MMC_SRT_ERASE

Information removed by an erase

HAL_MMC_SRT_WRITE_CHAR_ERASE

Information removed by an overwriting with a character followed by an erase

HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM

Information removed by an overwriting with a character, its complement then a random character

HAL_MMC_SRT_VENDOR_DEFINED

Information removed using a vendor defined

IS_MMC_SRT_TYPE

Exported Constants

MMC_BLOCKSIZE

Block size is 512 bytes

MMC Exported Macros

__HAL_MMC_RESET_HANDLE_STATE

Description:

- Reset MMC handle state.

Parameters:

- __HANDLE__: MMC Handle.

Return value:

- None

__HAL_MMC_ENABLE_IT

Description:

- Enable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_DISABLE_IT

Description:

- Disable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_GET_FLAG

Description:

- Check whether the specified MMC flag is set or not.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of MMC FLAG (SET or RESET).

__HAL_MMC_CLEAR_FLAG

Description:

- Clear the MMC's pending flags.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_MMC_GET_IT

Description:

- Check whether the specified MMC interrupt has occurred or not.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of MMC IT (SET or RESET).

__HAL_MMC_CLEAR_IT

Description:

- Clear the MMC's interrupt pending bits.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

MMC Card State enumeration structure

HAL_MMC_CARD_IDLE

Card is in idle state (can't be checked by CMD13)

HAL_MMC_CARD_READY

Card state is ready (can't be checked by CMD13)

HAL_MMC_CARD_IDENTIFICATION

Card is in identification state (can't be checked by CMD13)

HAL_MMC_CARD_STANDBY

Card is in standby state

HAL_MMC_CARD_TRANSFER

Card is in transfer state

HAL_MMC_CARD_SENDING

Card is sending an operation

HAL_MMC_CARD_RECEIVING

Card is receiving operation information

HAL_MMC_CARD_PROGRAMMING

Card is in programming state

HAL_MMC_CARD_DISCONNECTED

Card is disconnected

HAL_MMC_CARD_BUSTEST

Card is in bus test state

HAL_MMC_CARD_SLEEP

Card is in sleep state (can't be checked by CMD13)

HAL_MMC_CARD_ERROR

Card response Error (can't be checked by CMD13)

MMC Handle Structure definition

MMC_InitTypeDef

MMC_TypeDef

46 HAL MMC Extension Driver

46.1 MMCEx Firmware driver API description

The following section lists the various functions of the MMCEx library.

46.1.1 How to use this driver

The MMC Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_MMCEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_MMCEx_ReadBlocksDMAMultiBuffer() and HAL_MMCEx_WriteBlocksDMAMultiBuffer() functions.

46.1.2 Linked List management functions

This subsection provides a set of functions allowing to manage the needed functions.

This section contains the following APIs:

- *HAL_MMCEx_DMALinkedList_BuildNode()*
- *HAL_MMCEx_DMALinkedList_InsertNode()*
- *HAL_MMCEx_DMALinkedList_RemoveNode()*
- *HAL_MMCEx_DMALinkedList_LockNode()*
- *HAL_MMCEx_DMALinkedList_UnlockNode()*
- *HAL_MMCEx_DMALinkedList_EnableCircularMode()*
- *HAL_MMCEx_DMALinkedList_DisableCircularMode()*
- *HAL_MMCEx_DMALinkedList_ReadBlocks()*
- *HAL_MMCEx_DMALinkedList_WriteBlocks()*
- *HAL_MMCEx_DMALinkedList_ReadBlocks()*
- *HAL_MMCEx_DMALinkedList_WriteBlocks()*
- *HAL_MMCEx_DMALinkedList_EnableCircularMode()*
- *HAL_MMCEx_DMALinkedList_DisableCircularMode()*
- *HAL_MMCEx_Read_DMALnkLstBufCpltCallback()*
- *HAL_MMCEx_Write_DMALnkLstBufCpltCallback()*

46.1.3 Detailed description of functions

HAL_MMCEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_ReadBlocks (MMC_HandleTypeDef * hmmc, SDMMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_WriteBlocks (MMC_HandleTypeDef * hmmc, SDMMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_BuildNode (MMC_DMALinkNodeTypeDef * pNode, MMC_DMALinkNodeConfTypeDef * pNodeConf)

Function description

Build Linked List node.

Parameters

- **pNode:** Pointer to new node to add.
- **pNodeConf:** Pointer to configuration parameters for new node to add.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_InsertNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pPrevNode, MMC_DMALinkNodeTypeDef * pNewNode)

Function description

Insert Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pPrevNode:** Pointer to previous node.
- **pNewNode:** Pointer to new node to insert.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_RemoveNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pNode)

Function description

Remove Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_LockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

Lock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_UnlockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

Unlock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_EnableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_DisableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Read_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Write_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

46.2 MMCEx Firmware driver defines

The following section lists the various define and macros of the module.

46.2.1 MMCEx

MMCEx

Linked List Wrapper

MMC_DMALinkNodeTypeDef

MMC_DMALinkNodeConfTypeDef

MMC_DMALinkedListTypeDef

HAL_MMCx_DMALinkedList_WriteCpltCallback

HAL_MMCx_DMALinkedList_ReadCpltCallback

47 HAL NAND Generic Driver

47.1 NAND Firmware driver registers structures

47.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32h5xx_hal_nand.h

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

47.1.2 NAND_AddressTypeDef

NAND_AddressTypeDef is defined in the stm32h5xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

47.1.3 NAND_DeviceConfigTypeDef

NAND_DeviceConfigTypeDef is defined in the stm32h5xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages

- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. Example: Toshiba THTH58BYG3S0HBAI6. This parameter could be ENABLE or DISABLE Please check the Read Mode sequence in the NAND device datasheet

47.1.4 __NAND_HandleTypeDef

__NAND_HandleTypeDef is defined in the stm32h5xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* ItCallback***

Field Documentation

- ***FMC_NAND_TypeDef* __NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef __NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef __NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef __NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef __NAND_HandleTypeDef::Config***
  NAND physical characteristic information structure
- ***void(* __NAND_HandleTypeDef::MspInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp Init callback
- ***void(* __NAND_HandleTypeDef::MspDeInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp DeInit callback
- ***void(* __NAND_HandleTypeDef::ItCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND IT callback

47.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

47.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.

- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/ HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/ HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

Callback registration

The compilation define USE_HAL_NAND_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NAND_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NAND_UnRegisterCallback() to reset a callback to the default weak function. It allows to reset following callbacks:
- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NAND_Init and if the state is HAL_NAND_STATE_RESET all callbacks are reset to the corresponding legacy weak functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_NAND_Init and HAL_NAND_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NAND_Init and HAL_NAND_DeInit keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NAND_RegisterCallback before calling HAL_NAND_DeInit or HAL_NAND_Init function. When The compilation define USE_HAL_NAND_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

47.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***
- ***HAL_NAND_ITCallback()***
- ***HAL_NAND_ConfigDevice()***
- ***HAL_NAND_Read_ID()***

47.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- *HAL_NAND_Read_ID()*
- *HAL_NAND_Reset()*
- *HAL_NAND_ConfigDevice()*
- *HAL_NAND_Read_Page_8b()*
- *HAL_NAND_Read_Page_16b()*
- *HAL_NAND_Write_Page_8b()*
- *HAL_NAND_Write_Page_16b()*
- *HAL_NAND_Read_SpareArea_8b()*
- *HAL_NAND_Read_SpareArea_16b()*
- *HAL_NAND_Write_SpareArea_8b()*
- *HAL_NAND_Write_SpareArea_16b()*
- *HAL_NAND_Erase_Block()*
- *HAL_NAND_Address_Inc()*
- *HAL_NAND_RegisterCallback()*
- *HAL_NAND_UnRegisterCallback()*

47.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- *HAL_NAND_ECC_Enable()*
- *HAL_NAND_ECC_Disable()*
- *HAL_NAND_GetECC()*

47.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:

- *HAL_NAND_GetState()*
- *HAL_NAND_Read_Status()*

47.2.6 Detailed description of functions

HAL_NAND_Init

Function name

HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing:** pointer to Common space timing structure
- **AttSpace_Timing:** pointer to Attribute space timing structure

Return values

- **HAL:** status

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

Read the NAND memory electronic signature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP DeInit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

NAND memory reset.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

Increment the NAND memory address.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **The:** new status of the increment address operation. It can be:
  - NAND_VALID_ADDRESS: When the new address is valid address
  - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_RegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId, pNAND_CallbackTypeDef pCallback)

Function description

Register a User NAND Callback To be used instead of the weak predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  - HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  - HAL_NAND_IT_CB_ID NAND IT callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NAND_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_UnRegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NAND Callback NAND Callback is redirected to the weak predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  - HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  - HAL_NAND_IT_CB_ID NAND IT callback ID

Return values

- **status:**

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

Enables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

Disables dynamically FMC_NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

Disables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values

- **HAL:** status

HAL_NAND_GetState

Function name

HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)

Function description

return the NAND state

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** state

HAL_NAND_Read_Status

Function name

uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)

Function description

NAND memory read status.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **NAND:** status

47.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

47.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None

48 HAL NOR Generic Driver

48.1 NOR Firmware driver registers structures

48.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32h5xx_hal_nor.h

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command

48.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32h5xx_hal_nor.h

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
  < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***

48.1.3 __NOR_HandleTypeDef

__NOR_HandleTypeDef is defined in the stm32h5xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***
- ***uint32_t CommandSet***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __NOR_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef __NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef __NOR_HandleTypeDef::State***
  NOR device access state
- ***uint32_t __NOR_HandleTypeDef::CommandSet***
  NOR algorithm command set and control
- ***void(* __NOR_HandleTypeDef::MspInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp Init callback
- ***void(* __NOR_HandleTypeDef::MspDeInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp DeInit callback

48.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

48.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

Note: This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

Callback registration

The compilation define USE_HAL_NOR_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NOR_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NOR MspInit.
- MspDeInitCallback : NOR MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NOR_UnRegisterCallback() to reset a callback to the default weak function. It allows to reset following callbacks:
- MspInitCallback : NOR MspInit.

- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_NOR_Init and HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NOR_Init and HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NOR_RegisterCallback before calling HAL_NOR_DeInit or HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

48.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Init()***
- ***HAL_NOR_DeInit()***
- ***HAL_NOR_MspInit()***
- ***HAL_NOR_MspDeInit()***
- ***HAL_NOR_MspWait()***

48.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Read_ID()***
- ***HAL_NOR_ReturnToReadMode()***
- ***HAL_NOR_Read()***
- ***HAL_NOR_Program()***
- ***HAL_NOR_ReadBuffer()***
- ***HAL_NOR_ProgramBuffer()***
- ***HAL_NOR_Erase_Block()***
- ***HAL_NOR_Erase_Chip()***
- ***HAL_NOR_Read_CFI()***
- ***HAL_NOR_RegisterCallback()***
- ***HAL_NOR_UnRegisterCallback()***

48.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- ***HAL_NOR_WriteOperation_Enable()***
- ***HAL_NOR_WriteOperation_Disable()***

48.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- ***HAL_NOR_GetState()***
- ***HAL_NOR_GetStatus()***

48.2.6 Detailed description of functions

HAL_NOR_Init

Function name

HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Perform the NOR memory Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timing:** pointer to NOR control timing structure
- **ExtTiming:** pointer to NOR extended mode timing structure

Return values

- **HAL:** status

HAL_NOR_DeInit

Function name

HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)

Function description

Perform NOR memory De-Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_MspInit

Function name

void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP Init.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspDeInit

Function name

void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP DeInit.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspWait

Function name

void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)

Function description

NOR MSP Wait for Ready/Busy signal.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timeout:** Maximum timeout value

Return values

- **None:**

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

Read NOR flash IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** pointer to NOR ID structure

Return values

- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

Returns the NOR memory to Read mode.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Read data from NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** pointer to read data

Return values

- **HAL:** status

HAL_NOR_Program

Function name

HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Program data to NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** pointer to the data to write

Return values

- **HAL:** status

HAL_NOR_ReadBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Reads a half-word buffer from the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** number of Half word to read.

Return values

- **HAL:** status

HAL_NOR_ProgramBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Writes a half-word buffer to the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal start write address
- **pData:** pointer to source data buffer.
- **uwBufferSize:** Size of the buffer to write

Return values

- **HAL:** status

HAL_NOR_Erase_Block

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** Block to erase address
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Erase_Chip

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)

Function description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Read_CFI

Function name

HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)

Function description

Read NOR flash CFI IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** pointer to NOR CFI IDs structure

Return values

- **HAL:** status

HAL_NOR_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_RegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId, pNOR_CallbackTypeDef pCallback)

Function description

Register a User NOR Callback To be used instead of the weak predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NOR_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_UnRegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NOR Callback NOR Callback is redirected to the weak predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID

Return values

- **status:**

HAL_NOR_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)

Function description

Enables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

Disables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)

Function description

return the NOR controller state

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **NOR:** controller state

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

Returns the NOR operation status.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR programming Timeout

Return values

- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

48.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

48.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: specifies the NOR handle.

Return value:

- None

49 HAL OPAMP Generic Driver

49.1 OPAMP Firmware driver registers structures

49.1.1 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32h5xx_hal_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t PgaGain***
- ***uint32_t PgaConnect***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***
- ***uint32_t TrimmingValuePHighSpeed***
- ***uint32_t TrimmingValueNHighSpeed***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::PowerMode***
  Specifies the power mode Normal or High Speed. This parameter must be a value of ***OPAMP_PowerMode***
- ***uint32_t OPAMP_InitTypeDef::Mode***
  Specifies the OPAMP mode This parameter must be a value of ***OPAMP_Mode*** mode is either Standalone, Follower or PGA
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***
  Specifies the inverting input in Standalone & PGA modes.
  - In Standalone mode i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInput***
  - In Follower mode i.e when mode is OPAMP_FOLLOWER_MODE & In PGA mode i.e when mode is OPAMP_PGA_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***
  Specifies the non inverting input of the opamp: This parameter must be a value of ***OPAMP_NonInvertingInput***
- ***uint32_t OPAMP_InitTypeDef::PgaGain***
  Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain***
- ***uint32_t OPAMP_InitTypeDef::PgaConnect***
  Specifies the inverting pin in PGA mode i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of ***OPAMP_PgaConnect*** Either: not connected, connected to VINM0, connected to VINM1 (VINM0 or VINM1 are typically used for external filtering)
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***OPAMP_UserTrimming*** UserTrimming is either factory or user trimming.
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***
  Specifies the offset trimming value (PMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***
  Specifies the offset trimming value (NMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePHighSpeed***
  Specifies the offset trimming value (PMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNHighSpeed***
  Specifies the offset trimming value (NMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

49.1.2 __OPAMP_HandleTypeDef

__OPAMP_HandleTypeDef is defined in the stm32h5xx_hal_opamp.h

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OPAMP_TypeDef* __OPAMP_HandleTypeDef::Instance***
  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef __OPAMP_HandleTypeDef::Init***
  OPAMP required parameters
- ***HAL_StatusTypeDef __OPAMP_HandleTypeDef::Status***
  OPAMP peripheral status
- ***HAL_LockTypeDef __OPAMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_OPAMP_StateTypeDef __OPAMP_HandleTypeDef::State***
  OPAMP communication state
- ***void(* __OPAMP_HandleTypeDef::MspInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***
- ***void(* __OPAMP_HandleTypeDef::MspDeInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***

49.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

49.2.1 OPAMP Peripheral Features

The device integrates one operational amplifiers OPAMP1

1. The OPAMP provides several exclusive running modes.
   – Standalone mode
   – Programmable Gain Amplifier (PGA) modes
   – Follower mode
2. Each OPAMP(s) can be configured in normal and high speed mode.

3. The OPAMP(s) provide(s) calibration capabilities.
   – Calibration aims at correcting some offset for running mode.
   – The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   – The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate
   – HAL_OPAMP_SelfCalibrate:
     ◦ Runs automatically the calibration in 2 steps. (90% of VDDA for NMOS transistors, 10% of VDDA for PMOS transistors). (As OPAMP is Rail-to-rail input/output, these 2 steps calibration is appropriate and enough in most cases).
     ◦ Runs automatically the calibration.
     ◦ Enables the user trimming mode
     ◦ Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
4. Running mode: Standalone mode
   – Gain is set externally (gain depends on external loads).
   – Follower mode also possible externally by connecting the inverting input to the output.
5. Running mode: Follower mode
   – No Inverting Input is connected.
6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
7. The OPAMP(s) output(s) can be internally connected to resistor feedback output.
8. OPAMP gain can be selected as :
   a. Gain of x2, x4, x8 or x16 for non inverting mode with:
      ◦ VREF- referenced.
      ◦ Filtering on VINM0, VREF- referenced.
      ◦ VINM0 node for bias voltage and VINP0 for input signal.
      ◦ VINM0 node for bias voltage and VINP0 for input signal, VINM1 node for filtering.
   b. Gain of x-1, x-3, x-7 or x-15 for inverting mode with:
      ◦ VINM0 node for input signal and VINP0 for bias.
      ◦ VINM0 node for input signal and VINP0 for bias voltage, VINM1 node for filtering.
9. The OPAMPs inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.
10. The OPAMPs non inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.

==========

49.2.2 How to use this driver

High speed / normal power mode

To run in high speed mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select OPAMP_POWERMODE_HIGHSPEED
   – Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the OPAMP calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
   - Enable the OPAMP Peripheral clock using macro __HAL_RCC_OPAMP_CLK_ENABLE()
   - Configure the OPAMP input AND output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Registrate Callbacks
   - The compilation define USE_HAL_OPAMP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.
   - Use Functions HAL_OPAMP_RegisterCallback() to register a user callback, it allows to register following callbacks:
     - MspInitCallback : OPAMP MspInit.
     - MspDeInitCallback : OPAMP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
   - Use function HAL_OPAMP_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
     - MspInitCallback : OPAMP MspInit.
     - MspDeInitCallback : OPAMP MspDeInit.
     - All Callbacks
3. Configure the OPAMP using HAL_OPAMP_Init() function:
   - Select the mode
   - Select the inverting input
   - Select the non-inverting input
   - If PGA mode is enabled, Select if inverting input is connected.
   - Select either factory or user defined trimming mode.
   - If the user-defined trimming mode is enabled, select PMOS & NMOS trimming values (typically values set by HAL_OPAMP_SelfCalibrate function).
4. Enable the OPAMP using HAL_OPAMP_Start() function.
5. Disable the OPAMP using HAL_OPAMP_Stop() function.
6. Lock the OPAMP in running mode using HAL_OPAMP_Lock() function. Caution: On STM32H5, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
7. If needed, unlock the OPAMP using HAL_OPAMPEx_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
   - This is the case for instance if you wish to use new OPAMP I/O
2. Configure the OPAMP using HAL_OPAMP_Init() function:
   - As in configure case, select first the parameters you wish to modify.
3. Change from high speed mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

OPAMP pinout

Table 2. OPAMPs outputs for the STM32H5 devices: +---------- | | | OPAMP1 | |----------|--------|----------| | Output | VOUT | External : | | | PA7 | | | | | | | Internal : | | | | ADC1_INM3 | | | | ADC1_INP7 | |----------|--------|----------|

49.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_OPAMP_Init()*

- *HAL_OPAMP_DeInit()*
- *HAL_OPAMP_MspInit()*
- *HAL_OPAMP_MspDeInit()*

49.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- *HAL_OPAMP_Start()*
- *HAL_OPAMP_Stop()*
- *HAL_OPAMP_SelfCalibrate()*

49.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- *HAL_OPAMP_Lock()*
- *HAL_OPAMP_GetTrimOffset()*
- *HAL_OPAMP_RegisterCallback()*
- *HAL_OPAMP_UnRegisterCallback()*

49.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OPAMP_GetState()*

49.2.7 Detailed description of functions

HAL_OPAMP_Init

Function name

HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name

HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize the OPAMP peripheral.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- Deinitialization can be performed if the OPAMP configuration is locked. (the lock is SW in H7)

HAL_OPAMP_MspInit

Function name

void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_MspDeInit

Function name

void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_Start

Function name

HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)

Function description

Start the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_Stop

Function name

HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)

Function description

Stop the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)

Function description

Run the self calibration of one OPAMP.

Parameters

- **hopamp:** handle

Return values

- **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled
- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or high-speed). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.

HAL_OPAMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_RegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackId, pOPAMP_CallbackTypeDef pCallback)

Function description

Register a User OPAMP Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hopamp:** OPAMP handle
- **CallbackId:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OPAMP_MSPINIT_CB_ID OPAMP MspInit callback ID
  - HAL_OPAMP_MSPDEINIT_CB_ID OPAMP MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_OPAMP_RegisterCallback() may be called before HAL_OPAMP_Init() in HAL_OPAMP_STATE_RESET to register callbacks for HAL_OPAMP_MSPINIT_CB_ID and HAL_OPAMP_MSPDEINIT_CB_ID

HAL_OPAMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_UnRegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackId)

Function description

Unregister a User OPAMP Callback OPAMP Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hopamp:** OPAMP handle
- **CallbackId:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_OPAMP_MSPINIT_CB_ID OPAMP MSP Init Callback ID
  - HAL_OPAMP_MSPDEINIT_CB_ID OPAMP MSP DeInit Callback ID
  - HAL_OPAMP_ALL_CB_ID OPAMP All Callbacks

Return values

- **status:**

Notes

- The HAL_OPAMP_UnRegisterCallback() may be called before HAL_OPAMP_Init() in HAL_OPAMP_STATE_RESET to un-register callbacks for HAL_OPAMP_MSPINIT_CB_ID and HAL_OPAMP_MSPDEINIT_CB_ID

HAL_OPAMP_Lock

Function name

HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- On STM32H5, HAL OPAMP lock is software lock only (in contrast of hardware lock available on some other STM32 devices)

HAL_OPAMP_GetTrimOffset

Function name

HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (const OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)

Function description

Return the OPAMP factory trimming value.

Parameters

- **hopamp:** OPAMP handle
- **trimmingoffset:** Trimming offset (P or N) This parameter must be a value of OPAMP Factory Trimming

Return values

- **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

Notes

- On STM32H5 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before. Therefore, this function must be called when OPAMP init parameter "UserTrimming" is set to trimming factory, and before OPAMP calibration (function "HAL_OPAMP_SelfCalibrate()"). Otherwise, factory trimming value cannot be retrieved and error status is returned.
- Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or high-speed). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

HAL_OPAMP_GetState

Function name

HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (const OPAMP_HandleTypeDef * hopamp)

Function description

Return the OPAMP handle state.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** state

49.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

49.3.1 OPAMP

OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- __HANDLE__: OPAMP handle.

Return value:

- None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY

Dummy value if trimming value could not be retrieved

OPAMP_FACTORYTRIMMING_N

Offset trimming N

OPAMP_FACTORYTRIMMING_P

Offset trimming P

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0

OPAMP inverting input connected to I/O VINM0 (PC5 for OPAMP1)

OPAMP_INVERTINGINPUT_IO1

OPAMP inverting input connected to I/0 VINM1 (PB1 for OPAMP1)

OPAMP_INVERTINGINPUT_CONNECT_NO

OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA with positive gain without bias). Note: On this STM32 series, this literal include cases of value 0x11 for mode follower and value 0x10 for mode PGA.

OPAMP Mode

OPAMP_STANDALONE_MODE

standalone mode

OPAMP_PGA_MODE

PGA mode

OPAMP_FOLLOWER_MODE

follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0

OPAMP non inverting input connected to I/O VINP0 (PB0 for OPAMP1)

OPAMP_NONINVERTINGINPUT_IO1

OPAMP non inverting input connected to I/O VINP0 (PA0 for OPAMP1)

OPAMP_NONINVERTINGINPUT_DAC_CH

OPAMP non-inverting input connected internally to DAC channel

OPAMP Pga Connect

OPAMP_PGA_CONNECT_INVERTINGINPUT_NO

In PGA mode, the inverting input is not connected

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0

In PGA mode, the inverting input is connected to VINM0

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias , VINM1 connected for filtering

OPAMP Pga Gain

OPAMP_PGA_GAIN_2_OR_MINUS_1

PGA gain could be 2 or -1

OPAMP_PGA_GAIN_4_OR_MINUS_3

PGA gain could be 4 or -3

OPAMP_PGA_GAIN_8_OR_MINUS_7

PGA gain could be 8 or -7

OPAMP_PGA_GAIN_16_OR_MINUS_15

PGA gain could be 16 or -15

OPAMP PowerMode

OPAMP_POWERMODE_NORMAL

OPAMP_POWERMODE_HIGHSPEED

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY

Factory trimming

OPAMP_TRIMMING_USER

User trimming

OPAMP VREF

OPAMP_VREF_3VDDA

OPAMP Vref = 3.3% VDDA

OPAMP_VREF_10VDDA

OPAMP Vref = 10% VDDA

OPAMP_VREF_50VDDA

OPAMP Vref = 50% VDDA

OPAMP_VREF_90VDDA

OPAMP Vref = 90% VDDA

50 HAL OPAMP Extension Driver

50.1 OPAMPEx Firmware driver API description

The following section lists the various functions of the OPAMPEx library.

50.1.1 Peripheral Control functions

- OPAMP unlock.

50.1.2 Detailed description of functions

HAL_OPAMPEx_Unlock

Function name

HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)

Function description

Unlock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- This function must be called only when OPAMP is in state "locked".

51 HAL OTFDEC Generic Driver

51.1 OTFDEC Firmware driver registers structures

51.1.1 OTFDEC_RegionConfigTypeDef

OTFDEC_RegionConfigTypeDef is defined in the stm32h5xx_hal_otfdec.h

Data Fields

- ***uint32_t Nonce***
- ***uint32_t StartAddress***
- ***uint32_t EndAddress***
- ***uint16_t Version***

Field Documentation

- ***uint32_t OTFDEC_RegionConfigTypeDef::Nonce[2]***
  OTFDEC region nonce
- ***uint32_t OTFDEC_RegionConfigTypeDef::StartAddress***
  OTFDEC region start address
- ***uint32_t OTFDEC_RegionConfigTypeDef::EndAddress***
  OTFDEC region end address
- ***uint16_t OTFDEC_RegionConfigTypeDef::Version***
  OTFDEC region firmware version

51.1.2 __OTFDEC_HandleTypeDef

__OTFDEC_HandleTypeDef is defined in the stm32h5xx_hal_otfdec.h

Data Fields

- ***OTFDEC_TypeDef * Instance***
- ***HAL_OTFDEC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OTFDEC_TypeDef* __OTFDEC_HandleTypeDef::Instance***
  OTFDEC registers base address
- ***HAL_OTFDEC_StateTypeDef __OTFDEC_HandleTypeDef::State***
  OTFDEC state
- ***HAL_LockTypeDef __OTFDEC_HandleTypeDef::Lock***
  OTFDEC locking object
- ***__IO uint32_t __OTFDEC_HandleTypeDef::ErrorCode***
  OTFDEC error code
- ***void(* __OTFDEC_HandleTypeDef::ErrorCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC error callback
- ***void(* __OTFDEC_HandleTypeDef::MspInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp Init callback
- ***void(* __OTFDEC_HandleTypeDef::MspDeInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp DeInit callback

51.2 OTFDEC Firmware driver API description

The following section lists the various functions of the OTFDEC library.

51.2.1 How to use this driver

The OTFDEC HAL driver can be used as follows:

1. Declare an OTFDEC_HandleTypeDef handle structure (eg. OTFDEC_HandleTypeDef hotfdec).
2. Initialize the OTFDEC low level resources by implementing the HAL_OTFDEC_MspInit() API:
   - Enable the OTFDEC interface clock.
   - NVIC configuration if interrupts are used
     - Configure the OTFDEC interrupt priority.
     - Enable the NVIC OTFDEC IRQ handle.
3. Initialize the OTFDEC peripheral by calling the HAL_OTFDEC_Init() API.
4. In the case of encryption, enable ciphering mode for the peripheral
5. For each region,
   - Configure the region deciphering mode by calling the HAL_OTFDEC_RegionSetMode() API.
   - Write the region Key by calling the HAL_OTFDEC_RegionSetKey() API. If desired, read the key CRC by calling HAL_OTFDEC_RegionGetKeyCRC() API and compare the result with the theoretically expected CRC.
   - Initialize the OTFDEC region config structure with the Nonce, protected region start and end addresses and firmware version, and wrap-up the region configuration by calling HAL_OTFDEC_RegionConfig() API.
6. At this point, the OTFDEC region configuration is done and the deciphering or enciphering enabled. The region can be deciphered on the fly after having made sure the OctoSPI is configured in memory-mapped mode or data can be enciphered by calling HAL_OTFDEC_Cipher() API.

Note: Warning: the OTFDEC en/deciphering is based on a different endianness compared to the AES-CTR as implemented in the AES peripheral. E.g., if the OTFEC resorts to the Key (B0, B1, B2, B3) where Bi are 32-bit longwords and B0 is the Least Significant Word, the AES has to be configured with the Key (B3, B2, B1, B0) to report the same result (with the same swapping applied to the Initialization Vector).

Callback registration

The compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_OTFDEC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_OTFDEC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_OTFDEC_UnRegisterCallback to reset a callback to the default weak function.

@ref HAL_OTFDEC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback

By default, after the @ref HAL_OTFDEC_Init() and when the state is @ref HAL_OTFDEC_STATE_RESET all callbacks are set to the corresponding weak functions: example @ref HAL_OTFDEC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_OTFDEC_Init()/ @ref HAL_OTFDEC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the @ref HAL_OTFDEC_Init()/ @ref HAL_OTFDEC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_OTFDEC_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in @ref HAL_OTFDEC_STATE_READY or @ref HAL_OTFDEC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_OTFDEC_RegisterCallback() before calling @ref HAL_OTFDEC_DeInit() or @ref HAL_OTFDEC_Init() function.

When the compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

51.2.2 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_OTFDEC_Init()*
- *HAL_OTFDEC_DeInit()*
- *HAL_OTFDEC_MspInit()*
- *HAL_OTFDEC_MspDeInit()*
- *HAL_OTFDEC_RegisterCallback()*
- *HAL_OTFDEC_UnRegisterCallback()*

51.2.3 OTFDEC IRQ handler management

This section provides OTFDEC IRQ handler function.

This section contains the following APIs:

- *HAL_OTFDEC_IRQHandler()*
- *HAL_OTFDEC_ErrorCallback()*

51.2.4 Peripheral Control functions

This subsection permits to configure the OTFDEC peripheral

This section contains the following APIs:

- *HAL_OTFDEC_RegionKeyLock()*
- *HAL_OTFDEC_RegionSetKey()*
- *HAL_OTFDEC_RegionSetMode()*
- *HAL_OTFDEC_RegionConfig()*
- *HAL_OTFDEC_ConfigAttributes()*
- *HAL_OTFDEC_KeyCRCComputation()*
- *HAL_OTFDEC_EnableEnciphering()*
- *HAL_OTFDEC_DisableEnciphering()*
- *HAL_OTFDEC_Cipher()*
- *HAL_OTFDEC_RegionEnable()*
- *HAL_OTFDEC_RegionDisable()*

51.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OTFDEC_GetState()*
- *HAL_OTFDEC_GetConfigAttributes()*
- *HAL_OTFDEC_RegionGetKeyCRC()*
- *HAL_OTFDEC_RegionGetConfig()*

51.2.6 Detailed description of functions

HAL_OTFDEC_Init

Function name

HAL_StatusTypeDef HAL_OTFDEC_Init (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC peripheral and create the associated handle.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_DeInit

Function name

HAL_StatusTypeDef HAL_OTFDEC_DeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize the OTFDEC peripheral.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_MspInit

Function name

void HAL_OTFDEC_MspInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_MspDeInit

Function name

void HAL_OTFDEC_MspDeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID, pOTFDEC_CallbackTypeDef pCallback)

Function description

Register a User OTFDEC Callback To be used instead of the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  - HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_OTFDEC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_UnRegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID)

Function description

Unregister a OTFDEC Callback OTFDEC callback is redirected to the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  - HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_OTFDEC_IRQHandler

Function name

void HAL_OTFDEC_IRQHandler (OTFDEC_HandleTypeDef * hotfdec)

Function description

Handle OTFDEC interrupt request.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_ErrorCallback

Function name

void HAL_OTFDEC_ErrorCallback (OTFDEC_HandleTypeDef * hotfdec)

Function description

OTFDEC error callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegionKeyLock

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionKeyLock (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Lock region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are locked

Return values

- **HAL:** state

Notes

- Writes to this region KEYRx registers are ignored until next OTFDEC reset.

HAL_OTFDEC_RegionSetKey

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetKey (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t * pKey)

Function description

Set region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are set
- **pKey:** pointer at set of keys

Return values

- **HAL:** state

Notes

- The API reads the key CRC computed by the peripheral and compares it with that theoretically expected. An error is reported if they are different.

HAL_OTFDEC_RegionSetMode

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetMode (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t mode)

Function description

Set region mode.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the mode of which is set
- **mode:** This parameter can be only:
  - OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES All read accesses are decrypted (instruction or data)
  - OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER Only instruction accesses are decrypted with proprietary cipher activated

Return values

- **HAL:** state

HAL_OTFDEC_RegionConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, const OTFDEC_RegionConfigTypeDef * Config, uint32_t lock)

Function description

Set region configuration.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region that is configured
- **Config:** pointer on structure containing the region configuration parameters
- **lock:** configuration lock enable or disable parameter This parameter can be one of the following values:
  - OTFDEC_REG_CONFIGR_LOCK_DISABLE OTFDEC region configuration is not locked
  - OTFDEC_REG_CONFIGR_LOCK_ENABLE OTFDEC region configuration is locked

Return values

- **HAL:** state

Notes

- Region enciphering/deciphering is enabled at the end of this function

HAL_OTFDEC_KeyCRCComputation

Function name

uint32_t HAL_OTFDEC_KeyCRCComputation (const uint32_t * pKey)

Function description

Compute Key CRC.

Parameters

- **pKey:** pointer at set of keys

Return values

- **CRC:** value

HAL_OTFDEC_RegionEnable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionEnable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Enable region processing (enciphering or deciphering).

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the enciphering or deciphering is enabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_RegionDisable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionDisable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Disable region processing (enciphering or deciphering).

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the enciphering or deciphering is disabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_ConfigAttributes

Function name

HAL_StatusTypeDef HAL_OTFDEC_ConfigAttributes (OTFDEC_HandleTypeDef * hotfdec, uint32_t Attributes)

Function description

Configure OTFDEC attributes.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **Attributes:** This parameter can be only:
  - OTFDEC_ATTRIBUTE_PRIV Set privileged access protection
  - OTFDEC_ATTRIBUTE_NPRIV Reset privileged access protection

Return values

- **HAL:** state

Notes

- This function sets or resets regions privileged access protection.

HAL_OTFDEC_EnableEnciphering

Function name

HAL_StatusTypeDef HAL_OTFDEC_EnableEnciphering (OTFDEC_HandleTypeDef * hotfdec)

Function description

Enable peripheral enciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

Notes

- By default, deciphering mode is enabled at reset

HAL_OTFDEC_DisableEnciphering

Function name

HAL_StatusTypeDef HAL_OTFDEC_DisableEnciphering (OTFDEC_HandleTypeDef * hotfdec)

Function description

Disable peripheral enciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

HAL_OTFDEC_Cipher

Function name

HAL_StatusTypeDef HAL_OTFDEC_Cipher (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, const uint32_t * input, uint32_t * output, uint32_t size, uint32_t start_address)

Function description

Cipher data.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the configuration of which is used to encipher
- **input:** plain data
- **output:** ciphered data
- **size:** plain data size in words
- **start_address:** starting address in the external memory area where the enciphered data will be eventually stored

Return values

- **HAL:** state

Notes

- Region configuration parameters and OTFDEC_CR ENC bit must be set.
- output pointer points at a temporary area in RAM to store the ciphered data. It is up to the user code to copy the ciphered data in external RAM once the enciphering process is over.

HAL_OTFDEC_GetState

Function name

HAL_OTFDEC_StateTypeDef HAL_OTFDEC_GetState (const OTFDEC_HandleTypeDef * hotfdec)

Function description

Return the OTFDEC state.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

HAL_OTFDEC_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_OTFDEC_GetConfigAttributes (OTFDEC_HandleTypeDef * hotfdec, uint32_t * Attributes)

Function description

Get OTFDEC configuration attributes.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **Attributes:** pointer to attributes variable. This parameter can be only:
  - OTFDEC_ATTRIBUTE_PRIV Set privileged access protection
  - OTFDEC_ATTRIBUTE_NPRIV Reset privileged access protection

Return values

- **HAL:** state

Notes

- This function returns whether or not the regions access protection is in privileged mode.

HAL_OTFDEC_RegionGetKeyCRC

Function name

uint32_t HAL_OTFDEC_RegionGetKeyCRC (const OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Return region keys CRC.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys CRC of which is read

Return values

- **Key:** CRC

HAL_OTFDEC_RegionGetConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionGetConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, OTFDEC_RegionConfigTypeDef * Config)

Function description

Return region configuration parameters.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the configuration of which is read
- **Config:** pointer on structure that will be filled up with the region configuration parameters

Return values

- **HAL:** state

51.3 OTFDEC Firmware driver defines

The following section lists the various define and macros of the module.

51.3.1 OTFDEC

OTFDEC

OTFDEC Configuration Attributes

OTFDEC_ATTRIBUTE_NPRIV

Non-privileged access protection

OTFDEC_ATTRIBUTE_PRIV

Privileged access protection

OTFDEC Error Definition

HAL_OTFDEC_ERROR_NONE

No error

HAL_OTFDEC_SECURITY_ERROR

Security error

HAL_OTFDEC_EXECUTE_ERROR

Execute-only Execute-Never error

HAL_OTFDEC_KEY_ERROR

Key error

HAL_OTFDEC_ERROR_INVALID_CALLBACK

Invalid Callback error

OTFDEC Exported Macros

__HAL_OTFDEC_RESET_HANDLE_STATE

Description:

- Reset OTFDEC handle state.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return value:

- None

__HAL_OTFDEC_ENABLE_IT

Description:

- Enable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on enabled interrupts This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  - OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_DISABLE_IT

Description:

- Disable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on disabled interrupts This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  - OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_GET_FLAG

Description:

- Check whether the specified combination of OTFDEC interrupt flags is set or not.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  - OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_OTFDEC_CLEAR_FLAG

Description:

- Clear the specified combination of OTFDEC interrupt flags.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  - OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- None

OTFDEC Interrupts

OTFDEC_SEC_ERROR_INT

OTFDEC security error interrupt

OTFDEC_EXE_ERROR_INT

OTFDEC execution error interrupt

OTFDEC_KEY_ERROR_INT

OTFDEC key error interrupt

OTFDEC_SEC_EXE_ERROR_INT

OTFDEC security and execution errors interrupts

OTFDEC_SEC_KEY_ERROR_INT

OTFDEC security and key errors interrupts

OTFDEC_EXE_KEY_ERROR_INT

OTFDEC execution and key errors interrupts

OTFDEC_ALL_INT

OTFDEC all interrupts

OTFDEC Regions Index

OTFDEC_REGION1

OTFDEC region 1

OTFDEC_REGION2

OTFDEC region 2

OTFDEC_REGION3

OTFDEC region 3

OTFDEC_REGION4

OTFDEC region 4

OTFDEC Region Configuration Lock

OTFDEC_REG_CONFIGR_LOCK_DISABLE

OTFDEC region configuration lock disable

OTFDEC_REG_CONFIGR_LOCK_ENABLE

OTFDEC region configuration lock enable

OTFDEC Region Enable

OTFDEC_REG_CONFIGR_REG_DISABLE

OTFDEC region encryption or on-the-fly decryption disable

OTFDEC_REG_CONFIGR_REG_ENABLE

OTFDEC region encryption or on-the-fly decryption enable

OTFDEC Region Operating Mode

OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES

All read accesses are decrypted

OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER

Only instruction accesses are decrypted with proprietary cipher activated

52 HAL PCD Generic Driver

52.1 PCD Firmware driver registers structures

52.1.1 __PCD_HandleTypeDef

__PCD_HandleTypeDef is defined in the stm32h5xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* SetupStageCallback***
- ***void(* ResetCallback***
- ***void(* SuspendCallback***
- ***void(* ResumeCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* DataOutStageCallback***
- ***void(* DataInStageCallback***
- ***void(* ISOOUTIncompleteCallback***
- ***void(* ISOINIncompleteCallback***
- ***void(* BCDCallback***
- ***void(* LPMCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PCD_TypeDef* __PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef __PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t __PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef __PCD_HandleTypeDef::IN_ep[8]***
  IN endpoint parameters
- ***PCD_EPTypeDef __PCD_HandleTypeDef::OUT_ep[8]***
  OUT endpoint parameters
- ***HAL_LockTypeDef __PCD_HandleTypeDef::Lock***
  PCD peripheral status

- ***__IO PCD_StateTypeDef __PCD_HandleTypeDef::State***
  PCD communication state
- ***__IO uint32_t __PCD_HandleTypeDef::ErrorCode***
  PCD Error code
- ***uint32_t __PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef __PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t __PCD_HandleTypeDef::BESL***
- ***uint32_t __PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t __PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* __PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler
- ***void(* __PCD_HandleTypeDef::SOFCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD SOF callback
- ***void(* __PCD_HandleTypeDef::SetupStageCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Setup Stage callback
- ***void(* __PCD_HandleTypeDef::ResetCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Reset callback
- ***void(* __PCD_HandleTypeDef::SuspendCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Suspend callback
- ***void(* __PCD_HandleTypeDef::ResumeCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Resume callback
- ***void(* __PCD_HandleTypeDef::ConnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Connect callback
- ***void(* __PCD_HandleTypeDef::DisconnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Disconnect callback
- ***void(* __PCD_HandleTypeDef::DataOutStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data OUT Stage callback
- ***void(* __PCD_HandleTypeDef::DataInStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data IN Stage callback
- ***void(* __PCD_HandleTypeDef::ISOOUTIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO OUT Incomplete callback
- ***void(* __PCD_HandleTypeDef::ISOINIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO IN Incomplete callback
- ***void(* __PCD_HandleTypeDef::BCDCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_BCD_MsgTypeDef msg)***
  USB OTG PCD BCD callback
- ***void(* __PCD_HandleTypeDef::LPMCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_LPM_MsgTypeDef msg)***
  USB OTG PCD LPM callback
- ***void(* __PCD_HandleTypeDef::MspInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp Init callback
- ***void(* __PCD_HandleTypeDef::MspDeInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp DeInit callback

52.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

52.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      ◦ __HAL_RCC_USB_CLK_ENABLE(); For USB Device FS peripheral
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

52.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***
- ***HAL_PCD_RegisterCallback()***
- ***HAL_PCD_UnRegisterCallback()***
- ***HAL_PCD_RegisterDataOutStageCallback()***
- ***HAL_PCD_UnRegisterDataOutStageCallback()***
- ***HAL_PCD_RegisterDataInStageCallback()***
- ***HAL_PCD_UnRegisterDataInStageCallback()***
- ***HAL_PCD_RegisterIsoOutIncpltCallback()***
- ***HAL_PCD_UnRegisterIsoOutIncpltCallback()***
- ***HAL_PCD_RegisterIsoInIncpltCallback()***
- ***HAL_PCD_UnRegisterIsoInIncpltCallback()***
- ***HAL_PCD_RegisterBcdCallback()***
- ***HAL_PCD_UnRegisterBcdCallback()***
- ***HAL_PCD_RegisterLpmCallback()***
- ***HAL_PCD_UnRegisterLpmCallback()***

52.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- ***HAL_PCD_Start()***
- ***HAL_PCD_Stop()***
- ***HAL_PCD_IRQHandler()***
- ***HAL_PCD_DataOutStageCallback()***
- ***HAL_PCD_DataInStageCallback()***
- ***HAL_PCD_SetupStageCallback()***
- ***HAL_PCD_SOFCallback()***
- ***HAL_PCD_ResetCallback()***

- *HAL_PCD_SuspendCallback()*
- *HAL_PCD_ResumeCallback()*
- *HAL_PCD_ISOOUTIncompleteCallback()*
- *HAL_PCD_ISOINIncompleteCallback()*
- *HAL_PCD_ConnectCallback()*
- *HAL_PCD_DisconnectCallback()*

52.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- *HAL_PCD_DevConnect()*
- *HAL_PCD_DevDisconnect()*
- *HAL_PCD_SetAddress()*
- *HAL_PCD_EP_Open()*
- *HAL_PCD_EP_Close()*
- *HAL_PCD_EP_Receive()*
- *HAL_PCD_EP_GetRxCount()*
- *HAL_PCD_EP_Transmit()*
- *HAL_PCD_EP_SetStall()*
- *HAL_PCD_EP_ClrStall()*
- *HAL_PCD_EP_Abort()*
- *HAL_PCD_EP_Flush()*
- *HAL_PCD_ActivateRemoteWakeup()*
- *HAL_PCD_DeActivateRemoteWakeup()*

52.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PCD_GetState()*

52.2.6 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize the associated handle.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes the PCD peripheral.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID, pPCD_CallbackTypeDef pCallback)

Function description

Register a User USB PCD Callback To be used instead of the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB PCD Callback USB PCD callback is redirected to the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PCD_RegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataOutStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data OUT Stage Callback To be used instead of the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data OUT Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data OUT Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataInStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataInStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data IN Stage Callback To be used instead of the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data IN Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataInStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data IN Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoOutIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso OUT incomplete Callback To be used instead of the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso OUT incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso OUT incomplete Callback USB PCD Iso OUT incomplete Callback is redirected to the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoInIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso IN incomplete Callback To be used instead of the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso IN incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso IN incomplete Callback USB PCD Iso IN incomplete Callback is redirected to the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterBcdCallback (PCD_HandleTypeDef * hpcd, pPCD_BcdCallbackTypeDef pCallback)

Function description

Register USB PCD BCD Callback To be used instead of the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD BCD Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterBcdCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD BCD Callback USB BCD Callback is redirected to the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterLpmCallback (PCD_HandleTypeDef * hpcd, pPCD_LpmCallbackTypeDef pCallback)

Function description

Register USB PCD LPM Callback To be used instead of the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD LPM Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterLpmCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD LPM Callback USB LPM Callback is redirected to the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

Start the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

Stop the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

This function handles PCD interrupt request.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Start Of Frame callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SetupStageCallback

Function name

void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Reset callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

Suspend event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

Resume event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Connection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Disconnection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DataOutStageCallback

Function name

void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data OUT stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback

Function name

void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data IN stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO OUT callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO IN callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DevConnect

Function name

HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function description

Connect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DevDisconnect

Function name

HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function description

Disconnect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetAddress

Function name

HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description

Set the USB Device address.

Parameters

- **hpcd:** PCD handle
- **address:** new device address

Return values

- **HAL:** status

HAL_PCD_EP_Open

Function name

HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description

Open and configure an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values

- **HAL:** status

HAL_PCD_EP_Close

Function name

HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Deactivate an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Receive

Function name

HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Receive an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values

- **HAL:** status

HAL_PCD_EP_Transmit

Function name

HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Send an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the transmission buffer
- **len:** amount of data to be sent

Return values

- **HAL:** status

HAL_PCD_EP_SetStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Set a STALL condition over an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_ClrStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Clear a STALL condition over in an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Flush

Function name

HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef const * hpcd, uint8_t ep_addr)

Function description

Flush an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Abort

Function name

HAL_StatusTypeDef HAL_PCD_EP_Abort (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Abort an USB EP transaction.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

Activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

De-activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_EP_GetRxCount

Function name

uint32_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef const * hpcd, uint8_t ep_addr)

Function description

Get Received Data Size.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **Data:** Size

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef const * hpcd)

Function description

Return the PCD handle state.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** state

PCD_GET_EP_RX_CNT

Function name

__STATIC_INLINE uint16_t PCD_GET_EP_RX_CNT (const PCD_TypeDef * Instance, uint16_t bEpNum)

Function description

gets counter of the rx buffer.

Parameters

- **Instance:** USB peripheral instance register address.
- **bEpNum:** channel Number.

Return values

- **Counter:** value

PCD_GET_EP_DBUF0_CNT

Function name

__STATIC_INLINE uint16_t PCD_GET_EP_DBUF0_CNT (const PCD_TypeDef * Instance, uint16_t bEpNum)

Function description

gets counter of the rx buffer0.

Parameters

- **Instance:** USB peripheral instance register address.
- **bEpNum:** channel Number.

Return values

- **Counter:** value

PCD_GET_EP_DBUF1_CNT

Function name

__STATIC_INLINE uint16_t PCD_GET_EP_DBUF1_CNT (const PCD_TypeDef * Instance, uint16_t bEpNum)

Function description

gets counter of the rx buffer1.

Parameters

- **Instance:** USB peripheral instance register address.
- **bEpNum:** channel Number.

Return values

- **Counter:** value

52.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

52.3.1 PCD

PCD

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD Endpoint Kind

PCD_SNG_BUF

PCD_DBL_BUF

PCD EP0 MPS

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD Error Code definition

HAL_PCD_ERROR_INVALID_CALLBACK

Invalid Callback error

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

PCD PHY Module

PCD_PHY_ULPI

PCD_PHY_EMBEDDED

PCD_PHY_UTMI

PCD Speed

PCD_SPEED_FULL

53 HAL PCD Extension Driver

53.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

53.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL_PCDEx_PMAConfig()*
- *HAL_PCDEx_ActivateBCD()*
- *HAL_PCDEx_DeActivateBCD()*
- *HAL_PCDEx_BCD_VBUSDetect()*
- *HAL_PCDEx_ActivateLPM()*
- *HAL_PCDEx_DeActivateLPM()*
- *HAL_PCDEx_LPM_Callback()*
- *HAL_PCDEx_BCD_Callback()*

53.1.2 Detailed description of functions

HAL_PCDEx_PMAConfig

Function name

HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaadress)

Function description

Configure PMA for EP.

Parameters

- **hpcd:** Device instance
- **ep_addr:** endpoint address
- **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used
- **pmaadress:** EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.

Return values

- **HAL:** status

HAL_PCDEx_ActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Activate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Deactivate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_ActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Activate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Deactivate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

Handle BatteryCharging Process.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_LPM_Callback

Function name

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

Send LPM message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

HAL_PCDEx_BCD_Callback

Function name

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

Send BatteryCharging message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

54 HAL PKA Generic Driver

54.1 PKA Firmware driver registers structures

54.1.1 __PKA_HandleTypeDef

__PKA_HandleTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***PKA_TypeDef * Instance***
- ***__IO HAL_PKA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* OperationCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PKA_TypeDef* __PKA_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_PKA_StateTypeDef __PKA_HandleTypeDef::State***
  PKA state
- ***__IO uint32_t __PKA_HandleTypeDef::ErrorCode***
  PKA Error code
- ***void(* __PKA_HandleTypeDef::OperationCpltCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA End of operation callback
- ***void(* __PKA_HandleTypeDef::ErrorCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Error callback
- ***void(* __PKA_HandleTypeDef::MspInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp Init callback
- ***void(* __PKA_HandleTypeDef::MspDeInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp DeInit callback

54.1.2 PKA_ECCMulInTypeDef

PKA_ECCMulInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t scalarMulSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint8_t * scalarMul***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECCMulInTypeDef::scalarMulSize***
  Number of element in scalarMul array
- ***uint32_t PKA_ECCMulInTypeDef::modulusSize***
  Number of element in modulus, coefA, pointX and pointY arrays
- ***uint32_t PKA_ECCMulInTypeDef::coefSign***
  Curve coefficient a sign

- ***const uint8_t* PKA_ECCMulInTypeDef::coefA***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::coefB***
  pointer to curve coefficient b
- ***const uint8_t* PKA_ECCMulInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::scalarMul***
  Pointer to scalar multiplier k (Array of scalarMulSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::primeOrder***
  pointer to order of the curve

54.1.3 PKA_PointCheckInTypeDef

PKA_PointCheckInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_PointCheckInTypeDef::modulusSize***
  Number of element in coefA, coefB, modulus, pointX and pointY arrays
- ***uint32_t PKA_PointCheckInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_PointCheckInTypeDef::coefA***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::coefB***
  Pointer to curve coefficient b (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint32_t* PKA_PointCheckInTypeDef::pMontgomeryParam***
  pointer to montgomery param R2 (modulus N)

54.1.4 PKA_RSACRTExpInTypeDef

PKA_RSACRTExpInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint8_t * pOpDp***
- ***const uint8_t * pOpDq***
- ***const uint8_t * pOpQinv***
- ***const uint8_t * pPrimeP***
- ***const uint8_t * pPrimeQ***

- ***const uint8_t * popA***

Field Documentation

- ***uint32_t PKA_RSACRTExpInTypeDef::size***
  Number of element in popA array
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpDp***
  Pointer to operand dP (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpDq***
  Pointer to operand dQ (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpQinv***
  Pointer to operand qinv (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pPrimeP***
  Pointer to prime p (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pPrimeQ***
  Pointer to prime Q (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::popA***
  Pointer to operand A (Array of size elements)

54.1.5 PKA_ECDSAVerifInTypeDef

PKA_ECDSAVerifInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coef***
- ***const uint8_t * modulus***
- ***const uint8_t * basePointX***
- ***const uint8_t * basePointY***
- ***const uint8_t * pPubKeyCurvePtX***
- ***const uint8_t * pPubKeyCurvePtY***
- ***const uint8_t * RSign***
- ***const uint8_t * SSign***
- ***const uint8_t * hash***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECDSAVerifInTypeDef::primeOrderSize***
  Number of element in primeOrder array
- ***uint32_t PKA_ECDSAVerifInTypeDef::modulusSize***
  Number of element in modulus array
- ***uint32_t PKA_ECDSAVerifInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::coef***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::basePointX***
  Pointer to curve base point xG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::basePointY***
  Pointer to curve base point yG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::pPubKeyCurvePtX***
  Pointer to public-key curve point xQ (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::pPubKeyCurvePtY***
  Pointer to public-key curve point yQ (Array of modulusSize elements)

- ***const uint8_t* PKA_ECDSAVerifInTypeDef::RSign***
  Pointer to signature part r (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::SSign***
  Pointer to signature part s (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::hash***
  Pointer to hash of the message e (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::primeOrder***
  Pointer to order of the curve n (Array of primeOrderSize elements)

54.1.6 PKA_ECDSASignInTypeDef

PKA_ECDSASignInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coef***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * integer***
- ***const uint8_t * basePointX***
- ***const uint8_t * basePointY***
- ***const uint8_t * hash***
- ***const uint8_t * privateKey***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECDSASignInTypeDef::primeOrderSize***
  Number of element in primeOrder array
- ***uint32_t PKA_ECDSASignInTypeDef::modulusSize***
  Number of element in modulus array
- ***uint32_t PKA_ECDSASignInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_ECDSASignInTypeDef::coef***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::coefB***
  Pointer to B coefficient (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::integer***
  Pointer to random integer k (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::basePointX***
  Pointer to curve base point xG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::basePointY***
  Pointer to curve base point yG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::hash***
  Pointer to hash of the message (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::privateKey***
  Pointer to private key d (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::primeOrder***
  Pointer to order of the curve n (Array of primeOrderSize elements)

54.1.7 PKA_ECDSASignOutTypeDef

PKA_ECDSASignOutTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint8_t * RSign***
- ***uint8_t * SSign***

Field Documentation

- ***uint8_t* PKA_ECDSASignOutTypeDef::RSign***
  Pointer to signature part r (Array of modulusSize elements)
- ***uint8_t* PKA_ECDSASignOutTypeDef::SSign***
  Pointer to signature part s (Array of modulusSize elements)

54.1.8 PKA_ECDSASignOutExtParamTypeDef

PKA_ECDSASignOutExtParamTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint8_t * ptX***
- ***uint8_t * ptY***

Field Documentation

- ***uint8_t* PKA_ECDSASignOutExtParamTypeDef::ptX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***uint8_t* PKA_ECDSASignOutExtParamTypeDef::ptY***
  Pointer to point P coordinate yP (Array of modulusSize elements)

54.1.9 PKA_ECCCompleteAdditionOutTypeDef

PKA_ECCCompleteAdditionOutTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint8_t * ptX***
- ***uint8_t * ptY***
- ***uint8_t * ptZ***

Field Documentation

- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptX***
  pointer to point P coordinate xP
- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptY***
  pointer to point P coordinate yP
- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptZ***
  pointer to point P coordinate zP

54.1.10 PKA_ModExpInTypeDef

PKA_ModExpInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pExp***
- ***const uint8_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModExpInTypeDef::expSize***
  Number of element in pExp array
- ***uint32_t PKA_ModExpInTypeDef::OpSize***
  Number of element in pOp1 and pMod arrays
- ***const uint8_t* PKA_ModExpInTypeDef::pExp***
  Pointer to Exponent (Array of expSize elements)
- ***const uint8_t* PKA_ModExpInTypeDef::pOp1***
  Pointer to Operand (Array of OpSize elements)
- ***const uint8_t* PKA_ModExpInTypeDef::pMod***
  Pointer to modulus (Array of OpSize elements)

54.1.11 PKA_ModExpProtectModeInTypeDef

PKA_ModExpProtectModeInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pOp1***
- ***const uint8_t * pExp***
- ***const uint8_t * pMod***
- ***const uint8_t * pPhi***

Field Documentation

- ***uint32_t PKA_ModExpProtectModeInTypeDef::expSize***
  Size of the operand in bytes
- ***uint32_t PKA_ModExpProtectModeInTypeDef::OpSize***
  Size of the operand in bytes
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pOp1***
  Pointer to Operand 1
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pExp***
  Pointer to Exponent
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pMod***
  Pointer to Operand 1
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pPhi***
  Pointer to Phi value

54.1.12 PKA_ModExpFastModeInTypeDef

PKA_ModExpFastModeInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pExp***
- ***const uint8_t * pOp1***
- ***const uint8_t * pMod***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_ModExpFastModeInTypeDef::expSize***
  Number of element in pExp and pMontgomeryParam arrays
- ***uint32_t PKA_ModExpFastModeInTypeDef::OpSize***
  Number of element in pOp1 and pMod arrays
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pExp***
  Pointer to Exponent (Array of expSize elements)
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pOp1***
  Pointer to Operand (Array of OpSize elements)
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pMod***
  Pointer to modulus (Array of OpSize elements)
- ***const uint32_t* PKA_ModExpFastModeInTypeDef::pMontgomeryParam***
  Pointer to Montgomery parameter (Array of expSize/4 elements)

54.1.13 PKA_MontgomeryParamInTypeDef

PKA_MontgomeryParamInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint8_t * pOp1***

Field Documentation

- ***uint32_t PKA_MontgomeryParamInTypeDef::size***
  Number of element in pOp1 array
- ***const uint8_t* PKA_MontgomeryParamInTypeDef::pOp1***
  Pointer to Operand (Array of size elements)

54.1.14 PKA_AddInTypeDef

PKA_AddInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint32_t * pOp2***

Field Documentation

- ***uint32_t PKA_AddInTypeDef::size***
  Number of element in pOp1 and pOp2 arrays
- ***const uint32_t* PKA_AddInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint32_t* PKA_AddInTypeDef::pOp2***
  Pointer to Operand 2 (Array of size elements)

54.1.15 PKA_ModInvInTypeDef

PKA_ModInvInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModInvInTypeDef::size***
  Number of element in pOp1 array
- ***const uint32_t* PKA_ModInvInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint8_t* PKA_ModInvInTypeDef::pMod***
  Pointer to modulus value n (Array of size*4 elements)

54.1.16 PKA_ModRedInTypeDef

PKA_ModRedInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t OpSize***
- ***uint32_t modSize***
- ***const uint32_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModRedInTypeDef::OpSize***
  Number of element in pOp1 array
- ***uint32_t PKA_ModRedInTypeDef::modSize***
  Number of element in pMod array
- ***const uint32_t* PKA_ModRedInTypeDef::pOp1***
  Pointer to Operand 1 (Array of OpSize elements)
- ***const uint8_t* PKA_ModRedInTypeDef::pMod***
  Pointer to modulus value n (Array of modSize elements)

54.1.17 PKA_ModAddInTypeDef

PKA_ModAddInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint32_t * pOp2***
- ***const uint8_t * pOp3***

Field Documentation

- ***uint32_t PKA_ModAddInTypeDef::size***
  Number of element in pOp1 and pOp2 arrays
- ***const uint32_t* PKA_ModAddInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint32_t* PKA_ModAddInTypeDef::pOp2***
  Pointer to Operand 2 (Array of size elements)
- ***const uint8_t* PKA_ModAddInTypeDef::pOp3***
  Pointer to Operand 3 (Array of size*4 elements)

54.1.18 PKA_ECCDoubleBaseLadderInTypeDef

PKA_ECCDoubleBaseLadderInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * modulus***
- ***const uint8_t * integerK***
- ***const uint8_t * integerM***
- ***const uint8_t * basePointX1***
- ***const uint8_t * basePointY1***
- ***const uint8_t * basePointZ1***
- ***const uint8_t * basePointX2***
- ***const uint8_t * basePointY2***
- ***const uint8_t * basePointZ2***

Field Documentation

- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::primeOrderSize***
  curve prime order n length
- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::modulusSize***
  curve modulus p length
- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::coefSign***
  curve coefficient a sign
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::coefA***
  pointer to curve coefficient |a|
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::integerK***
  pointer to cryptographically secure random integer k
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::integerM***
  pointer to cryptographically secure random integer m
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointX1***
  pointer to curve base first point coordinate x
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointY1***
  pointer to curve base first point coordinate y
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointZ1***
  pointer to curve base first point coordinate z

- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointX2***
  pointer to curve base second point coordinate x
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointY2***
  pointer to curve base second point coordinate y
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointZ2***
  pointer to curve base second point coordinate z

54.1.19 PKA_ECCProjective2AffineInTypeDef

PKA_ECCProjective2AffineInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***const uint8_t * modulus***
- ***const uint8_t * basePointX***
- ***const uint8_t * basePointY***
- ***const uint8_t * basePointZ***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_ECCProjective2AffineInTypeDef::modulusSize***
  curve modulus p length
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointX***
  pointer to curve base point coordinate x
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointY***
  pointer to curve base point coordinate y
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointZ***
  pointer to curve base point coordinate z
- ***const uint32_t* PKA_ECCProjective2AffineInTypeDef::pMontgomeryParam***
  pointer to montgomery parameter R2 modulus n

54.1.20 PKA_ECCCompleteAdditionInTypeDef

PKA_ECCCompleteAdditionInTypeDef is defined in the stm32h5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * modulus***
- ***const uint8_t * coefA***
- ***const uint8_t * basePointX1***
- ***const uint8_t * basePointY1***
- ***const uint8_t * basePointZ1***
- ***const uint8_t * basePointX2***
- ***const uint8_t * basePointY2***
- ***const uint8_t * basePointZ2***

Field Documentation

- ***uint32_t PKA_ECCCompleteAdditionInTypeDef::modulusSize***
  curve modulus p length
- ***uint32_t PKA_ECCCompleteAdditionInTypeDef::coefSign***
  curve coefficient a sign
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::coefA***
  pointer to curve coefficient |a|

- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointX1***
  pointer to curve base first point coordinate x
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointY1***
  pointer to curve base first point coordinate y
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointZ1***
  pointer to curve base first point coordinate z
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointX2***
  pointer to curve base second point coordinate x
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointY2***
  pointer to curve base second point coordinate y
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointZ2***
  pointer to curve base second point coordinate z

54.2 PKA Firmware driver API description

The following section lists the various functions of the PKA library.

54.2.1 How to use this driver

The PKA HAL driver can be used as follows:

1. Declare a PKA_HandleTypeDef handle structure, for example: PKA_HandleTypeDef hpka;
2. Initialize the PKA low level resources by implementing the HAL_PKA_MspInit() API:
   a. Enable the PKA interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the PKA interrupt priority
      - Enable the NVIC PKA IRQ Channel
3. Initialize the PKA registers by calling the HAL_PKA_Init() API which trig HAL_PKA_MspInit().
4. Fill entirely the input structure corresponding to your operation: For instance: PKA_ModExpInTypeDef for HAL_PKA_ModExp().
5. Execute the operation (in polling or interrupt) and check the returned value.
6. Retrieve the result of the operation (For instance, HAL_PKA_ModExp_GetResult for HAL_PKA_ModExp operation). The function to gather the result is different for each kind of operation. The correspondence can be found in the following section.
7. Call the function HAL_PKA_DeInit() to restore the default configuration which trig HAL_PKA_MspDeInit().

High level operation

- Input structure requires buffers as uint8_t array.
- Output structure requires buffers as uint8_t array.
- Modular exponentiation using:
  – HAL_PKA_ModExp().
  – HAL_PKA_ModExp_IT().
  – HAL_PKA_ModExpFastMode().
  – HAL_PKA_ModExpFastMode_IT().
  – HAL_PKA_ModExpProtectMode().
  – HAL_PKA_ModExpProtectMode_IT().
  – HAL_PKA_ModExp_GetResult() to retrieve the result of the operation.
- RSA Chinese Remainder Theorem (CRT) using:
  – HAL_PKA_RSACRTExp().
  – HAL_PKA_RSACRTExp_IT().
  – HAL_PKA_RSACRTExp_GetResult() to retrieve the result of the operation.

- ECC Point Check using:
  - HAL_PKA_PointCheck().
  - HAL_PKA_PointCheck_IT().
  - HAL_PKA_PointCheck_IsOnCurve() to retrieve the result of the operation.
- ECDSA Sign
  - HAL_PKA_ECDSASign().
  - HAL_PKA_ECDSASign_IT().
  - HAL_PKA_ECDSASign_GetResult() to retrieve the result of the operation.
- ECDSA Verify
  - HAL_PKA_ECDSAVerif().
  - HAL_PKA_ECDSAVerif_IT().
  - HAL_PKA_ECDSAVerif_IsValidSignature() to retrieve the result of the operation.
- ECC Scalar Multiplication using:
  - HAL_PKA_ECCMul().
  - HAL_PKA_ECCMul_IT().
  - HAL_PKA_ECCMul_GetResult() to retrieve the result of the operation.
- ECC double base ladder using:
  - HAL_PKA_ECCDoubleBaseLadder().
  - HAL_PKA_ECCDoubleBaseLadder_IT().
  - HAL_PKA_ECCDoubleBaseLadder_GetResult() to retrieve the result of the operation.
- ECC projective to affine using:
  - HAL_PKA_ECCProjective2Affine().
  - HAL_PKA_ECCProjective2Affine_IT().
  - HAL_PKA_ECCProjective2Affine_GetResult() to retrieve the result of the operation.
- ECC complete addition using:
  - HAL_PKA_ECCCompleteAddition().
  - HAL_PKA_ECCCompleteAddition_IT().
  - HAL_PKA_ECCCompleteAddition_GetResult() to retrieve the result of the operation.

Low level operation

- Input structure requires buffers as uint32_t array.
- Output structure requires buffers as uint32_t array.
- Arithmetic addition using:
  - HAL_PKA_Add().
  - HAL_PKA_Add_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation. The resulting size can be the input parameter or the input parameter size + 1 (overflow).
- Arithmetic subtraction using:
  - HAL_PKA_Sub().
  - HAL_PKA_Sub_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Arithmetic multiplication using:
  - HAL_PKA_Mul().
  - HAL_PKA_Mul_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Comparison using:
  - HAL_PKA_Cmp().
  - HAL_PKA_Cmp_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

- Modular addition using:
  - HAL_PKA_ModAdd().
  - HAL_PKA_ModAdd_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular subtraction using:
  - HAL_PKA_ModSub().
  - HAL_PKA_ModSub_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular inversion using:
  - HAL_PKA_ModInv().
  - HAL_PKA_ModInv_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular reduction using:
  - HAL_PKA_ModRed().
  - HAL_PKA_ModRed_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Montgomery multiplication using:
  - HAL_PKA_MontgomeryMul().
  - HAL_PKA_MontgomeryMul_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

Montgomery parameter

Polling mode operation

- When an operation is started in polling mode, the function returns when:
  - A timeout is encounter.
  - The operation is completed.

Interrupt mode operation

- Add HAL_PKA_IRQHandler to the IRQHandler of PKA.
- Enable the IRQ using HAL_NVIC_EnableIRQ().
- When an operation is started in interrupt mode, the function returns immediately.
- When the operation is completed, the callback HAL_PKA_OperationCpltCallback is called.
- When an error is encountered, the callback HAL_PKA_ErrorCallback is called.
- To stop any operation in interrupt mode, use HAL_PKA_Abort().

Utilities

- To clear the PKA RAM, use HAL_PKA_RAMReset().
- To get current state, use HAL_PKA_GetState().
- To get current error, use HAL_PKA_GetError().

Callback registration

The compilation flag USE_HAL_PKA_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_PKA_RegisterCallback() to register an interrupt callback.

Function HAL_PKA_RegisterCallback() allows to register following callbacks:

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_PKA_UnRegisterCallback to reset a callback to the default weak function.

HAL_PKA_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_PKA_Init() and when the state is HAL_PKA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_PKA_OperationCpltCallback(), HAL_PKA_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_PKA_Init()/ HAL_PKA_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_PKA_Init()/ HAL_PKA_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_PKA_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_PKA_STATE_READY or HAL_PKA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_PKA_RegisterCallback() before calling HAL_PKA_DeInit() or HAL_PKA_Init() function.

When the compilation flag USE_HAL_PKA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

54.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PKAx peripheral:

- User must implement HAL_PKA_MspInit() function in which he configures all related peripherals resources (CLOCK, IT and NVIC).
- Call the function HAL_PKA_Init() to configure the device.
- Call the function HAL_PKA_DeInit() to restore the default configuration of the selected PKAx peripheral.

This section contains the following APIs:

- ***HAL_PKA_Init()***
- ***HAL_PKA_DeInit()***
- ***HAL_PKA_MspInit()***
- ***HAL_PKA_MspDeInit()***
- ***HAL_PKA_RegisterCallback()***
- ***HAL_PKA_UnRegisterCallback()***

54.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PKA operations.

1. There are two modes of operation:
   - Blocking mode : The operation is performed in the polling mode. These functions return when data operation is completed.
   - No-Blocking mode : The operation is performed using Interrupts. These functions return immediately. The end of the operation is indicated by HAL_PKA_ErrorCallback in case of error. The end of the operation is indicated by HAL_PKA_OperationCpltCallback in case of success. To stop any operation in interrupt mode, use HAL_PKA_Abort().

2. Blocking mode functions are :
   - HAL_PKA_ModExp()
   - HAL_PKA_ModExpFastMode()
   - HAL_PKA_ModExpProtectMode()
   - HAL_PKA_ModExp_GetResult();
   - HAL_PKA_ECDSASign()
   - HAL_PKA_ECDSASign_GetResult();
   - HAL_PKA_ECDSAVerif()
   - HAL_PKA_ECDSAVerif_IsValidSignature();
   - HAL_PKA_RSACRTExp()
   - HAL_PKA_RSACRTExp_GetResult();
   - HAL_PKA_PointCheck()
   - HAL_PKA_PointCheck_IsOnCurve();
   - HAL_PKA_ECCMul()
   - HAL_PKA_ECCMulFastMode()
   - HAL_PKA_ECCMul_GetResult();
   - HAL_PKA_ECCDoubleBaseLadder()
   - HAL_PKA_ECCDoubleBaseLadder_GetResult();
   - HAL_PKA_ECCProjective2Affine()
   - HAL_PKA_ECCProjective2Affine_GetResult();
   - HAL_PKA_ECCCompleteAddition()
   - HAL_PKA_ECCCompleteAddition_GetResult();
   - HAL_PKA_Add()
   - HAL_PKA_Sub()
   - HAL_PKA_Cmp()
   - HAL_PKA_Mul()
   - HAL_PKA_ModAdd()
   - HAL_PKA_ModSub()
   - HAL_PKA_ModInv()
   - HAL_PKA_ModRed()
   - HAL_PKA_MontgomeryMul()
   - HAL_PKA_Arithmetic_GetResult(P);
   - HAL_PKA_MontgomeryParam()
   - HAL_PKA_MontgomeryParam_GetResult();

3. No-Blocking mode functions with Interrupt are :
   - HAL_PKA_ModExp_IT();
   - HAL_PKA_ModExpFastMode_IT();
   - HAL_PKA_ModExpProtectMode_IT()
   - HAL_PKA_ModExp_GetResult();
   - HAL_PKA_ECDSASign_IT();
   - HAL_PKA_ECDSASign_GetResult();
   - HAL_PKA_ECDSAVerif_IT();
   - HAL_PKA_ECDSAVerif_IsValidSignature();
   - HAL_PKA_RSACRTExp_IT();
   - HAL_PKA_RSACRTExp_GetResult();
   - HAL_PKA_PointCheck_IT();
   - HAL_PKA_PointCheck_IsOnCurve();
   - HAL_PKA_ECCMul_IT();
   - HAL_PKA_ECCMulFastMode_IT();
   - HAL_PKA_ECCMul_GetResult();
   - HAL_PKA_ECCDoubleBaseLadder_IT()
   - HAL_PKA_ECCDoubleBaseLadder_GetResult();
   - HAL_PKA_ECCProjective2Affine_IT()
   - HAL_PKA_ECCProjective2Affine_GetResult();
   - HAL_PKA_ECCCompleteAddition_IT()
   - HAL_PKA_ECCCompleteAddition_GetResult();
   - HAL_PKA_Add_IT();
   - HAL_PKA_Sub_IT();
   - HAL_PKA_Cmp_IT();
   - HAL_PKA_Mul_IT();
   - HAL_PKA_ModAdd_IT();
   - HAL_PKA_ModSub_IT();
   - HAL_PKA_ModInv_IT();
   - HAL_PKA_ModRed_IT();
   - HAL_PKA_MontgomeryMul_IT();
   - HAL_PKA_Arithmetic_GetResult();
   - HAL_PKA_MontgomeryParam_IT();
   - HAL_PKA_MontgomeryParam_GetResult();
   - HAL_PKA_Abort();

This section contains the following APIs:

- ***HAL_PKA_ModExp()***
- ***HAL_PKA_ModExp_IT()***
- ***HAL_PKA_ModExpFastMode()***
- ***HAL_PKA_ModExpFastMode_IT()***
- ***HAL_PKA_ModExpProtectMode()***
- ***HAL_PKA_ModExpProtectMode_IT()***
- ***HAL_PKA_ModExp_GetResult()***
- ***HAL_PKA_ECDSASign()***
- ***HAL_PKA_ECDSASign_IT()***
- ***HAL_PKA_ECDSASign_GetResult()***
- ***HAL_PKA_ECDSAVerif()***
- ***HAL_PKA_ECDSAVerif_IT()***
- ***HAL_PKA_ECDSAVerif_IsValidSignature()***
- ***HAL_PKA_RSACRTExp()***

- *HAL_PKA_RSACRTExp_IT()*
- *HAL_PKA_RSACRTExp_GetResult()*
- *HAL_PKA_PointCheck()*
- *HAL_PKA_PointCheck_IT()*
- *HAL_PKA_PointCheck_IsOnCurve()*
- *HAL_PKA_ECCMul()*
- *HAL_PKA_ECCMul_IT()*
- *HAL_PKA_ECCMul_GetResult()*
- *HAL_PKA_Add()*
- *HAL_PKA_Add_IT()*
- *HAL_PKA_Sub()*
- *HAL_PKA_Sub_IT()*
- *HAL_PKA_Mul()*
- *HAL_PKA_Mul_IT()*
- *HAL_PKA_Cmp()*
- *HAL_PKA_Cmp_IT()*
- *HAL_PKA_ModAdd()*
- *HAL_PKA_ModAdd_IT()*
- *HAL_PKA_ModInv()*
- *HAL_PKA_ModInv_IT()*
- *HAL_PKA_ModSub()*
- *HAL_PKA_ModSub_IT()*
- *HAL_PKA_ModRed()*
- *HAL_PKA_ModRed_IT()*
- *HAL_PKA_MontgomeryMul()*
- *HAL_PKA_MontgomeryMul_IT()*
- *HAL_PKA_Arithmetic_GetResult()*
- *HAL_PKA_MontgomeryParam()*
- *HAL_PKA_MontgomeryParam_IT()*
- *HAL_PKA_ECCDoubleBaseLadder()*
- *HAL_PKA_ECCDoubleBaseLadder_IT()*
- *HAL_PKA_ECCProjective2Affine()*
- *HAL_PKA_ECCProjective2Affine_IT()*
- *HAL_PKA_ECCCompleteAddition()*
- *HAL_PKA_ECCCompleteAddition_IT()*
- *HAL_PKA_MontgomeryParam_GetResult()*
- *HAL_PKA_Abort()*
- *HAL_PKA_RAMReset()*
- *HAL_PKA_IRQHandler()*
- *HAL_PKA_OperationCpltCallback()*
- *HAL_PKA_ErrorCallback()*
- *HAL_PKA_ECCDoubleBaseLadder_GetResult()*
- *HAL_PKA_ECCProjective2Affine_GetResult()*
- *HAL_PKA_ECCCompleteAddition_GetResult()*

54.2.4 Peripheral State and Error functions

This subsection permit to get in run-time the status of the peripheral.
This section contains the following APIs:

- *HAL_PKA_GetState()*
- *HAL_PKA_GetError()*

54.2.5 Detailed description of functions

HAL_PKA_Init

Function name

HAL_StatusTypeDef HAL_PKA_Init (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA according to the specified parameters in the PKA_InitTypeDef and initialize the associated handle.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_DeInit

Function name

HAL_StatusTypeDef HAL_PKA_DeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA peripheral.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_MspInit

Function name

void HAL_PKA_MspInit (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_MspDeInit

Function name

void HAL_PKA_MspDeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_RegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID, pPKA_CallbackTypeDef pCallback)

Function description

Register a User PKA Callback To be used instead of the weak predefined callback.

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  - HAL_PKA_ERROR_CB_ID Error callback ID
  - HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  - HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PKA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_UnRegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID)

Function description

Unregister a PKA Callback PKA callback is redirected to the weak predefined callback.

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  - HAL_PKA_ERROR_CB_ID Error callback ID
  - HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  - HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PKA_ModExp

Function name

HAL_StatusTypeDef HAL_PKA_ModExp (PKA_HandleTypeDef * hpka, PKA_ModExpInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExp_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExp_IT (PKA_HandleTypeDef * hpka, PKA_ModExpInTypeDef * in)

Function description

Modular exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExpFastMode

Function name

HAL_StatusTypeDef HAL_PKA_ModExpFastMode (PKA_HandleTypeDef * hpka, PKA_ModExpFastModeInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExpFastMode_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExpFastMode_IT (PKA_HandleTypeDef * hpka, PKA_ModExpFastModeInTypeDef * in)

Function description

Modular exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExpProtectMode

Function name

HAL_StatusTypeDef HAL_PKA_ModExpProtectMode (PKA_HandleTypeDef * hpka, PKA_ModExpProtectModeInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation (protected) in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExpProtectMode_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExpProtectMode_IT (PKA_HandleTypeDef * hpka, PKA_ModExpProtectModeInTypeDef * in)

Function description

Modular exponentiation (protected) in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExp_GetResult

Function name

void HAL_PKA_ModExp_GetResult (PKA_HandleTypeDef * hpka, uint8_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Output buffer

Return values

- **HAL:** status

HAL_PKA_ECDSASign

Function name

HAL_StatusTypeDef HAL_PKA_ECDSASign (PKA_HandleTypeDef * hpka, PKA_ECDSASignInTypeDef * in, uint32_t Timeout)

Function description

Sign a message using elliptic curves over prime fields in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECDSASign_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECDSASign_IT (PKA_HandleTypeDef * hpka, PKA_ECDSASignInTypeDef * in)

Function description

Sign a message using elliptic curves over prime fields in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECDSASign_GetResult

Function name

void HAL_PKA_ECDSASign_GetResult (PKA_HandleTypeDef * hpka, PKA_ECDSASignOutTypeDef * out, PKA_ECDSASignOutExtParamTypeDef * outExt)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information
- **outExt:** Additional Output information (facultative)

HAL_PKA_ECDSAVerif

Function name

HAL_StatusTypeDef HAL_PKA_ECDSAVerif (PKA_HandleTypeDef * hpka, PKA_ECDSAVerifInTypeDef * in, uint32_t Timeout)

Function description

Verify the validity of a signature using elliptic curves over prime fields in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECDSAVerif_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECDSAVerif_IT (PKA_HandleTypeDef * hpka, PKA_ECDSAVerifInTypeDef * in)

Function description

Verify the validity of a signature using elliptic curves over prime fields in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECDSAVerif_IsValidSignature

Function name

uint32_t HAL_PKA_ECDSAVerif_IsValidSignature (PKA_HandleTypeDef const *const hpka)

Function description

Return the result of the ECDSA verification operation.

Parameters

- **hpka:** PKA handle

Return values

- **1:** if signature is verified, 0 in other case

HAL_PKA_RSACRTExp

Function name

HAL_StatusTypeDef HAL_PKA_RSACRTExp (PKA_HandleTypeDef * hpka, PKA_RSACRTExpInTypeDef * in, uint32_t Timeout)

Function description

RSA CRT exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_RSACRTExp_IT

Function name

HAL_StatusTypeDef HAL_PKA_RSACRTExp_IT (PKA_HandleTypeDef * hpka, PKA_RSACRTExpInTypeDef * in)

Function description

RSA CRT exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_RSACRTExp_GetResult

Function name

void HAL_PKA_RSACRTExp_GetResult (PKA_HandleTypeDef * hpka, uint8_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Pointer to memory location to receive the result of the operation

Return values

- **HAL:** status

HAL_PKA_PointCheck

Function name

HAL_StatusTypeDef HAL_PKA_PointCheck (PKA_HandleTypeDef * hpka, PKA_PointCheckInTypeDef * in, uint32_t Timeout)

Function description

Point on elliptic curve check in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_PointCheck_IT

Function name

HAL_StatusTypeDef HAL_PKA_PointCheck_IT (PKA_HandleTypeDef * hpka, PKA_PointCheckInTypeDef * in)

Function description

Point on elliptic curve check in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_PointCheck_IsOnCurve

Function name

uint32_t HAL_PKA_PointCheck_IsOnCurve (PKA_HandleTypeDef const *const hpka)

Function description

Return the result of the point check operation.

Parameters

- **hpka:** PKA handle

Return values

- **1:** if point is on curve, 0 in other case

HAL_PKA_ECCMul

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in, uint32_t Timeout)

Function description

ECC scalar multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCMul_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul_IT (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in)

Function description

ECC scalar multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCMul_GetResult

Function name

void HAL_PKA_ECCMul_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCMulOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_Add

Function name

HAL_StatusTypeDef HAL_PKA_Add (PKA_HandleTypeDef * hpka, PKA_AddInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Add_IT

Function name

HAL_StatusTypeDef HAL_PKA_Add_IT (PKA_HandleTypeDef * hpka, PKA_AddInTypeDef * in)

Function description

Arithmetic addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Sub

Function name

HAL_StatusTypeDef HAL_PKA_Sub (PKA_HandleTypeDef * hpka, PKA_SubInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic subtraction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Sub_IT

Function name

HAL_StatusTypeDef HAL_PKA_Sub_IT (PKA_HandleTypeDef * hpka, PKA_SubInTypeDef * in)

Function description

Arithmetic subtraction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Cmp

Function name

HAL_StatusTypeDef HAL_PKA_Cmp (PKA_HandleTypeDef * hpka, PKA_CmpInTypeDef * in, uint32_t Timeout)

Function description

Comparison in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Cmp_IT

Function name

HAL_StatusTypeDef HAL_PKA_Cmp_IT (PKA_HandleTypeDef * hpka, PKA_CmpInTypeDef * in)

Function description

Comparison in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Mul

Function name

HAL_StatusTypeDef HAL_PKA_Mul (PKA_HandleTypeDef * hpka, PKA_MulInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Mul_IT

Function name

HAL_StatusTypeDef HAL_PKA_Mul_IT (PKA_HandleTypeDef * hpka, PKA_MulInTypeDef * in)

Function description

Arithmetic multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModAdd

Function name

HAL_StatusTypeDef HAL_PKA_ModAdd (PKA_HandleTypeDef * hpka, PKA_ModAddInTypeDef * in, uint32_t Timeout)

Function description

Modular addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModAdd_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModAdd_IT (PKA_HandleTypeDef * hpka, PKA_ModAddInTypeDef * in)

Function description

Modular addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModSub

Function name

HAL_StatusTypeDef HAL_PKA_ModSub (PKA_HandleTypeDef * hpka, PKA_ModSubInTypeDef * in, uint32_t Timeout)

Function description

Modular subtraction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModSub_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModSub_IT (PKA_HandleTypeDef * hpka, PKA_ModSubInTypeDef * in)

Function description

Modular subtraction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModInv

Function name

HAL_StatusTypeDef HAL_PKA_ModInv (PKA_HandleTypeDef * hpka, PKA_ModInvInTypeDef * in, uint32_t Timeout)

Function description

Modular inversion in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModInv_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModInv_IT (PKA_HandleTypeDef * hpka, PKA_ModInvInTypeDef * in)

Function description

Modular inversion in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModRed

Function name

HAL_StatusTypeDef HAL_PKA_ModRed (PKA_HandleTypeDef * hpka, PKA_ModRedInTypeDef * in, uint32_t Timeout)

Function description

Modular reduction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModRed_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModRed_IT (PKA_HandleTypeDef * hpka, PKA_ModRedInTypeDef * in)

Function description

Modular reduction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_MontgomeryMul

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryMul (PKA_HandleTypeDef * hpka, PKA_MontgomeryMulInTypeDef * in, uint32_t Timeout)

Function description

Montgomery multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_MontgomeryMul_IT

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryMul_IT (PKA_HandleTypeDef * hpka, PKA_MontgomeryMulInTypeDef * in)

Function description

Montgomery multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Arithmetic_GetResult

Function name

void HAL_PKA_Arithmetic_GetResult (PKA_HandleTypeDef * hpka, uint32_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Pointer to memory location to receive the result of the operation

HAL_PKA_MontgomeryParam

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryParam (PKA_HandleTypeDef * hpka, PKA_MontgomeryParamInTypeDef * in, uint32_t Timeout)

Function description

Montgomery parameter computation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_MontgomeryParam_IT

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryParam_IT (PKA_HandleTypeDef * hpka, PKA_MontgomeryParamInTypeDef * in)

Function description

Montgomery parameter computation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_MontgomeryParam_GetResult

Function name

void HAL_PKA_MontgomeryParam_GetResult (PKA_HandleTypeDef * hpka, uint32_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** pointer to buffer where the result will be copied

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder

Function name

HAL_StatusTypeDef HAL_PKA_ECCDoubleBaseLadder (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderInTypeDef * in, uint32_t Timeout)

Function description

ECC double base ladder in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCDoubleBaseLadder_IT (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderInTypeDef * in)

Function description

ECC double base ladder in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder_GetResult

Function name

void HAL_PKA_ECCDoubleBaseLadder_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine

Function name

HAL_StatusTypeDef HAL_PKA_ECCProjective2Affine (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineInTypeDef * in, uint32_t Timeout)

Function description

ECC projective to affine in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCProjective2Affine_IT (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineInTypeDef * in)

Function description

ECC projective to affine in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine_GetResult

Function name

void HAL_PKA_ECCProjective2Affine_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition

Function name

HAL_StatusTypeDef HAL_PKA_ECCCompleteAddition (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionInTypeDef * in, uint32_t Timeout)

Function description

ECC complete addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCCompleteAddition_IT (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionInTypeDef * in)

Function description

ECC complete addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition_GetResult

Function name

void HAL_PKA_ECCCompleteAddition_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_Abort

Function name

HAL_StatusTypeDef HAL_PKA_Abort (PKA_HandleTypeDef * hpka)

Function description

Abort any ongoing operation.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_RAMReset

Function name

void HAL_PKA_RAMReset (PKA_HandleTypeDef * hpka)

Function description

Reset the PKA RAM.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_OperationCpltCallback

Function name

void HAL_PKA_OperationCpltCallback (PKA_HandleTypeDef * hpka)

Function description

Process completed callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_ErrorCallback

Function name

void HAL_PKA_ErrorCallback (PKA_HandleTypeDef * hpka)

Function description

Error callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_IRQHandler

Function name

void HAL_PKA_IRQHandler (PKA_HandleTypeDef * hpka)

Function description

This function handles PKA event interrupt request.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_GetState

Function name

HAL_PKA_StateTypeDef HAL_PKA_GetState (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA handle state.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_GetError

Function name

uint32_t HAL_PKA_GetError (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA error code.

Parameters

- **hpka:** PKA handle

Return values

- **PKA:** error code

54.3 PKA Firmware driver defines

The following section lists the various define and macros of the module.

54.3.1 PKA

PKA

PKA Error Code definition

HAL_PKA_ERROR_NONE

HAL_PKA_ERROR_ADDRERR

HAL_PKA_ERROR_RAMERR

HAL_PKA_ERROR_TIMEOUT

HAL_PKA_ERROR_OPERATION

HAL_PKA_ERROR_INVALID_CALLBACK

Invalid Callback error

PKA Exported Macros

__HAL_PKA_RESET_HANDLE_STATE

Description:

- Reset PKA handle state.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_ENABLE_IT

Description:

- Enable the specified PKA interrupt.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - PKA_IT_PROCEND End Of Operation interrupt enable
  - PKA_IT_ADDRERR Address error interrupt enable
  - PKA_IT_RAMERR RAM error interrupt enable
  - PKA_IT_OPERR Operation error interrupt enable

Return value:

- None

__HAL_PKA_DISABLE_IT

Description:

- Disable the specified PKA interrupt.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - PKA_IT_PROCEND End Of Operation interrupt enable
  - PKA_IT_ADDRERR Address error interrupt enable
  - PKA_IT_RAMERR RAM error interrupt enable
  - PKA_IT_OPERR Operation error interrupt enable

Return value:

- None

__HAL_PKA_GET_IT_SOURCE

Description:

- Check whether the specified PKA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the PKA interrupt source to check. This parameter can be one of the following values:
  - PKA_IT_PROCEND End Of Operation interrupt enable
  - PKA_IT_ADDRERR Address error interrupt enable
  - PKA_IT_RAMERR RAM error interrupt enable
  - PKA_IT_OPERR Operation error interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET)

__HAL_PKA_GET_FLAG

Description:

- Check whether the specified PKA flag is set or not.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - PKA_FLAG_PROCEND End Of Operation
  - PKA_FLAG_ADDRERR Address error
  - PKA_FLAG_RAMERR RAM error
  - PKA_FLAG_OPERR Operation error

Return value:

- The: new state of __FLAG__ (SET or RESET)

__HAL_PKA_CLEAR_FLAG

Description:

- Clear the PKA pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PKA_FLAG_PROCEND End Of Operation
  - PKA_FLAG_ADDRERR Address error
  - PKA_FLAG_RAMERR RAM error
  - PKA_FLAG_OPERR Operation error

Return value:

- None

__HAL_PKA_ENABLE

Description:

- Enable the specified PKA peripheral.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_DISABLE

Description:

- Disable the specified PKA peripheral.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_START

Description:

- Start a PKA operation.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

PKA Flag definition

PKA_FLAG_PROCEND

PKA_FLAG_ADDRERR

PKA_FLAG_RAMERR

PKA_FLAG_OPERR

PKA Interrupt configuration definition

PKA_IT_PROCEND

PKA_IT_ADDRERR

PKA_IT_RAMERR

PKA_IT_OPERR

PKA mode

PKA_MODE_MONTGOMERY_PARAM

PKA_MODE_MODULAR_EXP

PKA_MODE_MODULAR_EXP_FAST_MODE

PKA_MODE_ECC_MUL

PKA_MODE_ECDSA_SIGNATURE

PKA_MODE_ECDSA_VERIFICATION

PKA_MODE_POINT_CHECK

PKA_MODE_RSA_CRT_EXP

PKA_MODE_MODULAR_INV

PKA_MODE_ARITHMETIC_ADD

PKA_MODE_ARITHMETIC_SUB

PKA_MODE_ARITHMETIC_MUL

PKA_MODE_COMPARISON

PKA_MODE_MODULAR_RED

PKA_MODE_MODULAR_ADD

PKA_MODE_MODULAR_SUB

PKA_MODE_MONTGOMERY_MUL

PKA_MODE_ECC_PROJECTIVE_AFF

PKA_MODE_DOUBLE_BASE_LADDER

PKA_MODE_ECC_COMPLETE_ADD

PKA_MODE_MODULAR_EXP_PROTECT

PKA Operation Status

PKA_NO_ERROR

PKA_FAILED_COMPUTATION

PKA_RPART_SIGNATURE_NULL

PKA_SPART_SIGNATURE_NULL

55 HAL PSSI Generic Driver

55.1 PSSI Firmware driver registers structures

55.1.1 PSSI_InitTypeDef

PSSI_InitTypeDef is defined in the stm32h5xx_hal_pssi.h

Data Fields

- ***uint32_t DataWidth***
- ***uint32_t BusWidth***
- ***uint32_t ControlSignal***
- ***uint32_t ClockPolarity***
- ***uint32_t DataEnablePolarity***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t PSSI_InitTypeDef::DataWidth***
- ***uint32_t PSSI_InitTypeDef::BusWidth***
- ***uint32_t PSSI_InitTypeDef::ControlSignal***
- ***uint32_t PSSI_InitTypeDef::ClockPolarity***
- ***uint32_t PSSI_InitTypeDef::DataEnablePolarity***
- ***uint32_t PSSI_InitTypeDef::ReadyPolarity***

55.1.2 PSSI_HandleTypeDef

PSSI_HandleTypeDef is defined in the stm32h5xx_hal_pssi.h

Data Fields

- ***PSSI_TypeDef * Instance***
- ***PSSI_InitTypeDef Init***
- ***uint32_t * pBuffPtr***
- ***uint32_t XferCount***
- ***uint32_t XferSize***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_PSSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***PSSI_TypeDef* PSSI_HandleTypeDef::Instance***
  PSSI register base address.
- ***PSSI_InitTypeDef PSSI_HandleTypeDef::Init***
  PSSI Initialization Structure.
- ***uint32_t* PSSI_HandleTypeDef::pBuffPtr***
  PSSI Data buffer.
- ***uint32_t PSSI_HandleTypeDef::XferCount***
  PSSI transfer count
- ***uint32_t PSSI_HandleTypeDef::XferSize***
  PSSI transfer size
- ***DMA_HandleTypeDef* PSSI_HandleTypeDef::hdmatx***
  PSSI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* PSSI_HandleTypeDef::hdmarx***
  PSSI Rx DMA Handle parameters

- ***HAL_LockTypeDef PSSI_HandleTypeDef::Lock***
  PSSI lock.
- ***__IO HAL_PSSI_StateTypeDef PSSI_HandleTypeDef::State***
  PSSI transfer state.
- ***__IO uint32_t PSSI_HandleTypeDef::ErrorCode***
  PSSI error code.

55.2 PSSI Firmware driver API description

The following section lists the various functions of the PSSI library.

55.2.1 How to use this driver

The PSSI HAL driver can be used as follows:

1. Declare a PSSI_HandleTypeDef handle structure, for example: PSSI_HandleTypeDef hpssi;
2. Initialize the PSSI low level resources by implementing the @ref HAL_PSSI_MspInit() API:
   a. Enable the PSSIx interface clock
   b. PSSI pins configuration
      - Enable the clock for the PSSI GPIOs
      - Configure PSSI pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the PSSIx interrupt priority
      - Enable the NVIC PSSI IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare DMA_HandleTypeDef handles structure for the transmit and receive
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx and Rx
      - Associate the initialized DMA handle to the hpssi DMA Tx and Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx and Rx
3. Configure the Communication Bus Width, Control Signals, Input Polarity and Output Polarity in the hpssi Init structure.
4. Initialize the PSSI registers by calling the @ref HAL_PSSI_Init(), configure also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized @ref HAL_PSSI_MspInit(&hpssi) API.
5. For PSSI IO operations, two operation modes are available within this driver :

Polling mode IO operation

- Transmit an amount of data by byte in blocking mode using @ref HAL_PSSI_Transmit()
- Receive an amount of data by byte in blocking mode using @ref HAL_PSSI_Receive()

DMA mode IO operation

- Transmit an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Transmit_DMA()
- At transmission end of transfer, @ref HAL_PSSI_TxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Receive_DMA()
- At reception end of transfer, @ref HAL_PSSI_RxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_RxCpltCallback()
- In case of transfer Error, @ref HAL_PSSI_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_ErrorCallback()
- Abort a PSSI process communication with Interrupt using @ref HAL_PSSI_Abort_IT()
- End of abort process, @ref HAL_PSSI_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_AbortCpltCallback()

PSSI HAL driver macros list

Below the list of most used macros in PSSI HAL driver.

- @ref HAL_PSSI_ENABLE : Enable the PSSI peripheral
- @ref HAL_PSSI_DISABLE : Disable the PSSI peripheral
- @ref HAL_PSSI_GET_FLAG : Check whether the specified PSSI flag is set or not
- @ref HAL_PSSI_CLEAR_FLAG : Clear the specified PSSI pending flag
- @ref HAL_PSSI_ENABLE_IT : Enable the specified PSSI interrupt
- @ref HAL_PSSI_DISABLE_IT : Disable the specified PSSI interrupt

Callback registration

Note: You can refer to the PSSI HAL driver header file for more useful macros

55.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PSSIx peripheral:

- User must implement HAL_PSSI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_PSSI_Init() to configure the selected device with the selected configuration:
  - Data Width
  - Control Signals
  - Input Clock polarity
  - Output Clock polarity
- Call the function HAL_PSSI_DeInit() to restore the default configuration of the selected PSSIx peripheral.

This section contains the following APIs:

- ***HAL_PSSI_Init()***
- ***HAL_PSSI_DeInit()***
- ***HAL_PSSI_MspInit()***
- ***HAL_PSSI_MspDeInit()***

55.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PSSI data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated the DMA IRQ .
2. Blocking mode functions are :
   - HAL_PSSI_Transmit()
   - HAL_PSSI_Receive()
3. No-Blocking mode functions with DMA are :
   - HAL_PSSI_Transmit_DMA()
   - HAL_PSSI_Receive_DMA()
4. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_PSSI_TxCpltCallback()
   - HAL_PSSI_RxCpltCallback()
   - HAL_PSSI_ErrorCallback()
   - HAL_PSSI_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_PSSI_Transmit()***

- *HAL_PSSI_Receive()*
- *HAL_PSSI_Transmit_DMA()*
- *HAL_PSSI_Receive_DMA()*
- *HAL_PSSI_Abort_DMA()*

55.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PSSI_GetState()*
- *HAL_PSSI_GetError()*

55.2.5 Detailed description of functions

HAL_PSSI_Init

Function name

HAL_StatusTypeDef HAL_PSSI_Init (PSSI_HandleTypeDef * hpssi)

Function description

Initializes the PSSI according to the specified parameters in the PSSI_InitTypeDef and initialize the associated handle.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_DeInit

Function name

HAL_StatusTypeDef HAL_PSSI_DeInit (PSSI_HandleTypeDef * hpssi)

Function description

DeInitialize the PSSI peripheral.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_MspInit

Function name

void HAL_PSSI_MspInit (PSSI_HandleTypeDef * hpssi)

Function description

Initialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_MspDeInit

Function name

void HAL_PSSI_MspDeInit (PSSI_HandleTypeDef * hpssi)

Function description

De-Initialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_Transmit

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Receive

Function name

HAL_StatusTypeDef HAL_PSSI_Receive (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Receives an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Receive_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Abort_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Abort_DMA (PSSI_HandleTypeDef * hpssi)

Function description

Abort a DMA process communication with Interrupt.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_GetState

Function name

HAL_PSSI_StateTypeDef HAL_PSSI_GetState (const PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI handle state.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** state

HAL_PSSI_GetError

Function name

uint32_t HAL_PSSI_GetError (const PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI error code.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **PSSI:** Error Code

HAL_PSSI_IRQHandler

Function name

void HAL_PSSI_IRQHandler (PSSI_HandleTypeDef * hpssi)

Function description

This function handles PSSI event interrupt request.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_TxCpltCallback

Function name

void HAL_PSSI_TxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Tx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_RxCpltCallback

Function name

void HAL_PSSI_RxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Rx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_ErrorCallback

Function name

void HAL_PSSI_ErrorCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI error callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_AbortCpltCallback

Function name

void HAL_PSSI_AbortCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI abort callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

55.3 PSSI Firmware driver defines

The following section lists the various define and macros of the module.

55.3.1 PSSI

PSSI

PSSI Bus Width

HAL_PSSI_8LINES

8 data lines

HAL_PSSI_16LINES

16 data lines

PSSI Data Width

HAL_PSSI_8BITS

8 Bits

HAL_PSSI_16BITS

16 Bits

HAL_PSSI_32BITS

32 Bits

PSSI definitions

PSSI_MAX_NBYTE_SIZE

PSSI_TIMEOUT_TRANSMIT

Timeout Value

PSSI_CR_OUTEN_INPUT

Input Mode

PSSI_CR_OUTEN_OUTPUT

Output Mode

PSSI_CR_DMA_ENABLE

DMA Mode Enable

PSSI_CR_DMA_DISABLE

DMA Mode Disable

PSSI_CR_16BITS

16 Lines Mode

PSSI_CR_8BITS

8 Lines Mode

PSSI_FLAG_RTT1B

1 Byte Fifo Flag

PSSI_FLAG_RTT4B

4 Bytes Fifo Flag

PSSI Error Code

HAL_PSSI_ERROR_NONE

No error

HAL_PSSI_ERROR_NOT_SUPPORTED

Not supported operation

HAL_PSSI_ERROR_UNDER_RUN

FIFO Under-run error

HAL_PSSI_ERROR_OVER_RUN

FIFO Over-run error

HAL_PSSI_ERROR_DMA

Dma error

HAL_PSSI_ERROR_TIMEOUT

Timeout error

PSSI Exported Macros

HAL_PSSI_RESET_HANDLE_STATE

Description:

- Reset PSSI handle state.

Parameters:

- __HANDLE__: specifies the PSSI handle.

Return value:

- None

HAL_PSSI_ENABLE

Description:

- Enable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_DISABLE

Description:

- Disable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_GET_STATUS

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_RTT1B: FIFO is ready to transfer one byte
  - PSSI_FLAG_RTT4B: FIFO is ready to transfer four bytes

Return value:

- The: state of FLAG.

HAL_PSSI_GET_FLAG

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- The: state of FLAG.

HAL_PSSI_CLEAR_FLAG

Description:

- Clear the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- None

HAL_PSSI_ENABLE_IT

Description:

- Enable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Configuration error mask

Return value:

- None

HAL_PSSI_DISABLE_IT

Description:

- Disable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - PSSI_IT_OVR_IE: Configuration error mask

Return value:

- None

HAL_PSSI_GET_IT_SOURCE

Description:

- Check whether the specified PSSI interrupt source is enabled or not.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt source to check. This parameter can be one of the following values:
  - PSSI_IT_OVR_IE: Data Buffer overrun/underrun error interrupt mask

Return value:

- The: state of INTERRUPT source.

IS_PSSI_CONTROL_SIGNAL

Description:

- Check whether the PSSI Control signal is valid.

Parameters:

- __CONTROL__: Control signals configuration

Return value:

- Valid: or not.

IS_PSSI_BUSWIDTH

Description:

- Check whether the PSSI Bus Width is valid.

Parameters:

- __BUSWIDTH__: PSSI Bush width

Return value:

- Valid: or not.

IS_PSSI_CLOCK_POLARITY

Description:

- Check whether the PSSI Clock Polarity is valid.

Parameters:

- __CLOCKPOL__: PSSI Clock Polarity

Return value:

- Valid: or not.

IS_PSSI_DE_POLARITY

Description:

- Check whether the PSSI Data Enable Polarity is valid.

Parameters:

- __DEPOL__: PSSI DE Polarity

Return value:

- Valid: or not.

IS_PSSI_RDY_POLARITY

Description:

- Check whether the PSSI Ready Polarity is valid.

Parameters:

- __RDYPOL__: PSSI RDY Polarity

Return value:

- Valid: or not.

PSSI Interrupts

PSSI_FLAG_OVR_RIS

Overrun, Underrun errors flag

PSSI_FLAG_MASK

Overrun, Underrun errors Mask

PSSI_FLAG_OVR_MIS

Overrun, Underrun masked errors flag

PSSI mode

HAL_PSSI_UNIDIRECTIONAL

Uni-directional mode

HAL_PSSI_BIDIRECTIONAL

Bi-directional mode

56 HAL PWR Generic Driver

56.1 PWR Firmware driver registers structures

56.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32h5xx_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***
  Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_Detection_Level***.
- ***uint32_t PWR_PVDTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

56.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

56.2.1 Initialization and De-Initialization Functions

This section contains the following APIs:

- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

56.2.2 Peripheral Control functions

This section contains the following APIs:

- ***HAL_PWR_ConfigPVD()***
- ***HAL_PWR_EnablePVD()***
- ***HAL_PWR_DisablePVD()***
- ***HAL_PWR_EnableWakeUpPin()***
- ***HAL_PWR_DisableWakeUpPin()***
- ***HAL_PWR_EnterSLEEPMode()***
- ***HAL_PWR_EnterSTOPMode()***
- ***HAL_PWR_EnterSTANDBYMode()***
- ***HAL_PWR_EnableSleepOnExit()***
- ***HAL_PWR_DisableSleepOnExit()***
- ***HAL_PWR_EnableSEVOnPend()***
- ***HAL_PWR_DisableSEVOnPend()***
- ***HAL_PWR_PVD_IRQHandler()***
- ***HAL_PWR_PVDCallback()***

56.2.3 PWR Attributes Functions

This section contains the following APIs:

- ***HAL_PWR_ConfigAttributes()***
- ***HAL_PWR_GetConfigAttributes()***

56.2.4 Detailed description of functions

HAL_PWR_DeInit

Function name

void HAL_PWR_DeInit (void)

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None.:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_EnableBkUpAccess

Function name

void HAL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None.:**

Notes

- After a system reset, the backup domain is protected against possible unwanted write accesses.

HAL_PWR_DisableBkUpAccess

Function name

void HAL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None:**

HAL_PWR_ConfigPVD

Function name

HAL_StatusTypeDef HAL_PWR_ConfigPVD (const PWR_PVDTypeDef * sConfigPVD)

Function description

Configure the voltage threshold detected by the Programmed Voltage Detector (PVD).

Parameters

- **sConfigPVD:** : Pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information (PVDLevel and EventMode).

Return values

- **None.:**

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** : Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values, which sets the default (rising edge):
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3,PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5, PWR_WAKEUP_PIN6, PWR_WAKEUP_PIN7.PWR_WAKEUP_PIN8. or one of the following values where the user can explicitly states the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW, PWR_WAKEUP_PIN5_HIGH, PWR_WAKEUP_PIN5_LOW, PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW, PWR_WAKEUP_PIN7_HIGH, PWR_WAKEUP_PIN7_LOW, PWR_WAKEUP_PIN8_HIGH, PWR_WAKEUP_PIN8_LOW.

Return values

- **None.:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.
- The PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW, PWR_WAKEUP_PIN7_HIGH, PWR_WAKEUP_PIN7_LOW, PWR_WAKEUP_PIN8_HIGH and PWR_WAKEUP_PIN8_LOW are not available for STM32H503xx devices.

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** : Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3,PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5, PWR_WAKEUP_PIN6, PWR_WAKEUP_PIN7.PWR_WAKEUP_PIN8. or one of the following values where the user can explicitly states the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW, PWR_WAKEUP_PIN5_HIGH, PWR_WAKEUP_PIN5_LOW, PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW, PWR_WAKEUP_PIN7_HIGH, PWR_WAKEUP_PIN7_LOW, PWR_WAKEUP_PIN8_HIGH, PWR_WAKEUP_PIN8_LOW.

Return values

- **None.:**

Notes

- The PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW, PWR_WAKEUP_PIN7_HIGH, PWR_WAKEUP_PIN7_LOW, PWR_WAKEUP_PIN8_HIGH and PWR_WAKEUP_PIN8_LOW are not available for STM32H503xx devices.

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enter the CPU in SLEEP mode.

Parameters

- **Regulator:** : Specifies the regulator state in Sleep mode. This parameter can be one of the following values :
  - PWR_MAINREGULATOR_ON
  - PWR_LOWPOWERREGULATOR_ON
- **SLEEPEntry:** : Specifies if SLEEP mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  - PWR_SLEEPENTRY_WFI enter SLEEP mode with Wait For Interrupt request.
  - PWR_SLEEPENTRY_WFE enter SLEEP mode with Wait For Event request.

Return values

- **None.:**

Notes

- In SLEEP mode, all I/O pins keep the same state as in Run mode.
- CPU clock is off and all peripherals including Cortex-M33 core such as NVIC and SysTick can run and wake up the CPU when an interrupt or an event occurs.
- This parameter is not available in this product. The parameter is kept just to maintain compatibility with other products.
- When WFI entry is used, ticks interrupt must be disabled to avoid unexpected CPU wake up.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enter the whole system to STOP mode.

Parameters

- **Regulator:** : Specifies the regulator state in Sleep mode. This parameter can be one of the following values :
  - PWR_MAINREGULATOR_ON
  - PWR_LOWPOWERREGULATOR_ON
- **STOPEntry:** : Specifies if STOP mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  - PWR_STOPENTRY_WFI enter STOP mode with Wait For Interrupt request.
  - PWR_STOPENTRY_WFE enter STOP mode with Wait For Event request.

Return values

- **None.:**

Notes

- In STOP mode, the regulator remains in main regulator mode, allowing a very fast wakeup time but with much higher consumption comparing to other STOP modes.
- STOP offers the largest number of active peripherals and wakeup sources, a smaller wakeup time but a higher consumption. STOP mode achieves the lowest power consumption while retaining the content of SRAM and registers. All clocks in the VCORE domain are stopped. The PLL, the HSI, the CSI and the HSE crystal oscillators are disabled. The LSE or LSI is still running.
- The system clock when exiting from Stop mode can be either HSI or CSI, depending on software configuration.
- This parameter is not available in this product. The parameter is kept just to maintain compatibility with other products.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter the whole system to STANDBY mode.

Return values

- **None.:**

Notes

- The STANDBY mode is used to achieve the lowest power consumption with BOR. The internal regulator is switched off so that the VCORE domain is powered off. The PLL, the HSI, the CSI and the HSE crystal oscillators are also switched off.
- After entering STANDBY mode, SRAMs and register contents are lost except for registers and backup SRAM in the Backup domain and STANDBY circuitry.

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable CORTEX SEV-ON-PEND feature.

Return values

- **None.:**

Notes

- Sets SEVONPEND bit of SCR register. When this bit is set, any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry wakes up the Cortex-M33.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable CORTEX SEVONPEND feature.

Return values

- **None.:**

Notes

- Resets SEVONPEND bit of SCR register. When this bit is reset, only enabled pending causes exception entry wakes up the Cortex-M33.

HAL_PWR_PVD_IRQHandler

Function name

void HAL_PWR_PVD_IRQHandler (void)

Function description

This function handles the PWR PVD interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().
- The use of this API is only when we activate the PVD.
- When the PVD and AVD are activated at the same time you must use this API: HAL_PWREx_PVD_AVD_IRQHandler.

HAL_PWR_PVDCallback

Function name

void HAL_PWR_PVDCallback (void)

Function description

PWR PVD interrupt callback.

Return values

- **None.:**

HAL_PWR_ConfigAttributes

Function name

void HAL_PWR_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the PWR item attributes.

Parameters

- **Item:** : Specifies the item(s) to set attributes on. This parameter can be a combination of PWR Items.
- **Attributes:** : Specifies the available attribute(s). This parameter can be one of PWR Attributes.

Return values

- **None.:**

Notes

- Available attributes are security and privilege protection.
- Security attribute can only be set only by secure access.
- Privilege attribute for secure items can be managed only by a secure privileged access.
- Privilege attribute for nsecure items can be managed by a secure privileged access or by a nsecure privileged access.

HAL_PWR_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_PWR_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get attribute(s) of a PWR item.

Parameters

- **Item:** : Specifies the item(s) to set attributes on. This parameter can be one of PWR Items.
- **pAttributes:** : Pointer to return attribute(s). Returned value could be on of PWR Attributes.

Return values

- **HAL:** Status.

56.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

56.3.1 PWR

PWR

PWR Attributes

PWR_NSEC_PRIV

NSecure and Privileged attribute

PWR_NSEC_NPRIV

NSecure and NPrivileged attribute

PWR_SEC_PRIV

Secure and Privileged attribute

PWR_SEC_NPRIV

Secure and NPrivileged attribute

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check PWR flags are set or not.

Parameters:

- __FLAG__: : Specifies the flag to check. This parameter can be one of the following values :
  - PWR_FLAG_STOPF : Stop flag. Indicates that the device was resumed from Stop mode.
  - PWR_FLAG_SBF : Standby flag. Indicates that the device was resumed from Standby mode.
  - PWR_FLAG_VOSRDY : Voltage scaling ready flag. Indicates that the Vcore level at or above VOS selected level.
  - PWR_FLAG_ACTVOSRDY : Currently applied VOS ready flag. Indicates that Vcore is equal to the current voltage scaling provided by ACTVOS.
  - PWR_FLAG_BRR : Backup regulator ready flag. This bit is not reset when the device wakes up from STANDBY mode or by a system reset or power-on reset.
  - PWR_FLAG_VBATL : Backup domain voltage level flag (versus low threshold). Indicates the backup domain voltage level is equal or above low threshold.
  - PWR_FLAG_VBATH : Backup domain voltage level flag (versus high threshold). Indicates the backup domain voltage level is equal or above high threshold.
  - PWR_FLAG_TEMPL : Temperature level flag (versus low threshold). Indicates the temperature is equal or above low threshold.
  - PWR_FLAG_TEMPH : Temperature level flag (versus high threshold). Indicates the temperature is equal or above high threshold.
  - PWR_FLAG_AVDO : Regulator selection flag. Indicates the regulator selected.
  - PWR_FLAG_VDDIO2RDY : VDDIO2 ready flag (versus 0.9 V threshold). Indicates that VDDIO2 is equal or above the threshold of the VDDIO2 voltage monitor (around 0.9 V).
  - PWR_FLAG_PVDO : Voltage detector output flag. Indicates that Vdd is equal or above the PVD threshold selected by PVDLS.
  - PWR_FLAG_USB33RDY : VDDUSB ready flag (versus 1.2 V threshold). Indicates that VDDUSB is equal or above the threshold of the VDDUSB voltage monitor (around 1.2 V).
  - PWR_WAKEUP_FLAG1 : Wakeup flag 1. Indicates that a wakeup event was received from the WKUP line 1.
  - PWR_WAKEUP_FLAG2 : Wakeup flag 2. Indicates that a wakeup event was received from the WKUP line 2.
  - PWR_WAKEUP_FLAG3 : Wakeup flag 3. Indicates that a wakeup event was received from the WKUP line 3.
  - PWR_WAKEUP_FLAG4 : Wakeup flag 4. Indicates that a wakeup event was received from the WKUP line 4.
  - PWR_WAKEUP_FLAG5 : Wakeup flag 5. Indicates that a wakeup event was received from the WKUP line 5.
  - PWR_WAKEUP_FLAG6 : Wakeup flag 6. Indicates that a wakeup event was received from the WKUP line 6.
  - PWR_WAKEUP_FLAG7 : Wakeup flag 7. Indicates that a wakeup event was received from the WKUP line 7.
  - PWR_WAKEUP_FLAG8 : Wakeup flag 8. Indicates that a wakeup event was received from the WKUP line 8.

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

Notes:

- The PWR_WAKEUP_FLAG6, PWR_WAKEUP_FLAG7 AND PWR_WAKEUP_FLAG8 are not available for STM32H503xx devices.

__HAL_PWR_CLEAR_FLAG

Description:

- Clear PWR flags.

Parameters:

- __FLAG__: : Specifies the flag to clear. This parameter can be one of the following values :
  - PWR_FLAG_STOPF : STOP flag. Indicates that the device was resumed from STOP mode.
  - PWR_FLAG_SBF : STANDBY flag. Indicates that the device was resumed from STANDBY mode.
  - PWR_WAKEUP_FLAG1 : Wakeup flag 1. Indicates that a wakeup event was received from the WKUP line 1.
  - PWR_WAKEUP_FLAG2 : Wakeup flag 2. Indicates that a wakeup event was received from the WKUP line 2.
  - PWR_WAKEUP_FLAG3 : Wakeup flag 3. Indicates that a wakeup event was received from the WKUP line 3.
  - PWR_WAKEUP_FLAG4 : Wakeup flag 4. Indicates that a wakeup event was received from the WKUP line 4.
  - PWR_WAKEUP_FLAG5 : Wakeup flag 5. Indicates that a wakeup event was received from the WKUP line 5.
  - PWR_WAKEUP_FLAG6 : Wakeup flag 6. Indicates that a wakeup event was received from the WKUP line 6.
  - PWR_WAKEUP_FLAG7 : Wakeup flag 7. Indicates that a wakeup event was received from the WKUP line 7.
  - PWR_WAKEUP_FLAG8 : Wakeup flag 8. Indicates that a wakeup event was received from the WKUP line 8.

Return value:

- None.

Notes:

- The PWR_WAKEUP_FLAG6, PWR_WAKEUP_FLAG7 AND PWR_WAKEUP_FLAG8 are not available for STM32H503xx devices.

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable the PVD Event Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable the PVD Event Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:

- Generate a Software Interrupt on selected EXTI line.

Return value:

- None

__HAL_PWR_PVD_EXTI_GET_RISING_FLAG

Description:

- Check whether the specified PVD EXTI Rising interrupt flag is set or not.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_GET_FALLING_FLAG

Description:

- Check whether the specified PVD EXTI Falling interrupt flag is set or not.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_CLEAR_RISING_FLAG

Description:

- Clear the PVD EXTI Interrupt Rising flag.

Return value:

- None.

__HAL_PWR_PVD_EXTI_CLEAR_FALLING_FLAG

Description:

- Clear the PVD EXTI Interrupt Falling flag.

Return value:

- None.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

Description:

- Clear the PVD EXTI Interrupt flag.

Return value:

- None.

PWR Flags

PWR_FLAG_STOPF

STOP flag

PWR_FLAG_SBF

STANDBY flag

PWR_FLAG_VOSRDY

Voltage scaling ready flag

PWR_FLAG_ACTVOSRDY

Currently applied VOS ready flag

PWR_FLAG_BRR

Backup regulator ready flag

PWR_FLAG_VBATL

Backup domain voltage level flag (versus low threshold)

PWR_FLAG_VBATH

Backup domain voltage level flag (versus high threshold)

PWR_FLAG_TEMPL

Temperature level flag (versus low threshold)

PWR_FLAG_TEMPH

Temperature level flag (versus high threshold)

PWR_FLAG_AVDO

VDDA voltage detector output flag

PWR_FLAG_VDDIO2RDY

VDDIO2 voltage detector output flag

PWR_FLAG_PVDO

VDD voltage detector output flag

PWR_FLAG_USB33RDY

VDDUSB33 ready flag

PWR_WAKEUP_FLAG1

Wake up line 1 flag

PWR_WAKEUP_FLAG2

Wake up line 2 flag

PWR_WAKEUP_FLAG3

Wake up line 3 flag

PWR_WAKEUP_FLAG4

Wake up line 4 flag

PWR_WAKEUP_FLAG5

Wake up line 5 flag

PWR_WAKEUP_FLAG6

Wake up line 6 flag

PWR_WAKEUP_FLAG7

Wake up line 7 flag

PWR_WAKEUP_FLAG8

Wake up line 8 flag

PWR_WAKEUP_ALL_FLAG

Wakeup flag all

PWR Items

PWR_WKUP1

WUP1 secure protection

PWR_WKUP2

WUP2 secure protection

PWR_WKUP3

WUP3 secure protection

PWR_WKUP4

WUP4 secure protection

PWR_WKUP5

WUP5 secure protection

PWR_WKUP6

WUP6 secure protection

PWR_WKUP7

WUP7 secure protection

PWR_WKUP8

WUP8 secure protection

PWR_RET

IO Retention secure protection

PWR_LPM

Low power modes secure protection

PWR_SCM

Voltage detection and monitoring secure protection

PWR_VB

Backup domain secure protection

PWR_VUSB

Voltage USB secure protection

PWR_ALL

Programmable Voltage Detection Level

PWR_PVDLEVEL_0

PVD threshold around 1.95 V

PWR_PVDLEVEL_1

PVD threshold around 2.1 V

PWR_PVDLEVEL_2

PVD threshold around 2.25 V

PWR_PVDLEVEL_3

PVD threshold around 2.4 V

PWR_PVDLEVEL_4

PVD threshold around 2.55 V

PWR_PVDLEVEL_5

PVD threshold around 2.7 V

PWR_PVDLEVEL_6

PVD threshold around 2.85 V

PWR_PVDLEVEL_7

External input analog voltage (compared internally to VREFINT)

PWR PVD Mode

PWR_PVD_MODE_NORMAL

Basic Mode is used

PWR_PVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_RISING_EDGE

Mask for rising edge set as PVD trigger

PVD_FALLING_EDGE

Mask for falling edge set as PVD trigger

PVD_MODE_IT

Mask for interruption yielded by PVD threshold crossing

PVD_MODE_EVT

Mask for event yielded by PVD threshold crossing

PWR PVM Mode Mask

PVM_RISING_EDGE

Mask for rising edge set as PVM trigger

PVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

PVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

PVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PWR Regulator State in SLEEP/STOP Mode

PWR_MAINREGULATOR_ON

Main Regulator ON in Run Mode

PWR_LOWPOWERREGULATOR_ON

Main Regulator ON in Low Power Mode

PWR SLEEP Mode Entry

PWR_SLEEPENTRY_WFI

Wait For Interruption instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE

Wait For Event instruction to enter Sleep mode

PWR STOP Mode Entry

PWR_STOPENTRY_WFI

Wait For Interruption instruction to enter Stop mode

PWR_STOPENTRY_WFE

Wait For Event instruction to enter Stop mode

57 HAL PWR Extension Driver

57.1 PWREx Firmware driver registers structures

57.1.1 PWREx_AVDTypeDef

PWREx_AVDTypeDef is defined in the stm32h5xx_hal_pwr_ex.h

Data Fields

- ***uint32_t AVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWREx_AVDTypeDef::AVDLevel***
  AVDLevel: Specifies the AVD detection level. This parameter can be a value of ***PWREx_AVD_detection_level***
- ***uint32_t PWREx_AVDTypeDef::Mode***
  Mode: Specifies the EXTI operating mode for the AVD event. This parameter can be a value of ***PWREx_AVD_Mode***.

57.1.2 PWREx_WakeupPinTypeDef

PWREx_WakeupPinTypeDef is defined in the stm32h5xx_hal_pwr_ex.h

Data Fields

- ***uint32_t WakeUpPin***
- ***uint32_t PinPolarity***
- ***uint32_t PinPull***

Field Documentation

- ***uint32_t PWREx_WakeupPinTypeDef::WakeUpPin***
  WakeUpPin: Specifies the Wake-Up pin to be enabled. This parameter can be a value of ***PWREx_WakeUp_Pins***
- ***uint32_t PWREx_WakeupPinTypeDef::PinPolarity***
  PinPolarity: Specifies the Wake-Up pin polarity. This parameter can be a value of ***PWREx_PIN_Polarity***
- ***uint32_t PWREx_WakeupPinTypeDef::PinPull***
  PinPull: Specifies the Wake-Up pin pull. This parameter can be a value of ***PWREx_PIN_Pull***

57.2 PWREx Firmware driver API description

The following section lists the various functions of the PWREx library.

57.2.1 Power supply control functions

This section contains the following APIs:

- ***HAL_PWREx_ConfigSupply()***
- ***HAL_PWREx_GetSupplyConfig()***
- ***HAL_PWREx_ControlVoltageScaling()***
- ***HAL_PWREx_GetVoltageRange()***
- ***HAL_PWREx_ControlStopModeVoltageScaling()***
- ***HAL_PWREx_GetStopModeVoltageRange()***

57.2.2 Voltage Monitoring Functions

This section contains the following APIs:

- ***HAL_PWREx_ConfigAVD()***
- ***HAL_PWREx_EnableAVD()***
- ***HAL_PWREx_DisableAVD()***

- *HAL_PWREx_EnableUSBVoltageDetector()*
- *HAL_PWREx_DisableUSBVoltageDetector()*
- *HAL_PWREx_EnableVddUSB()*
- *HAL_PWREx_DisableVddUSB()*
- *HAL_PWREx_EnableMonitoring()*
- *HAL_PWREx_DisableMonitoring()*
- *HAL_PWREx_EnableUCPDStandbyMode()*
- *HAL_PWREx_DisableUCPDStandbyMode()*
- *HAL_PWREx_EnableUCPDDeadBattery()*
- *HAL_PWREx_DisableUCPDDeadBattery()*
- *HAL_PWREx_EnableBatteryCharging()*
- *HAL_PWREx_DisableBatteryCharging()*
- *HAL_PWREx_EnableAnalogBooster()*
- *HAL_PWREx_DisableAnalogBooster()*
- *HAL_PWREx_PVD_AVD_IRQHandler()*
- *HAL_PWREx_PVD_AVD_Rising_Callback()*
- *HAL_PWREx_PVD_AVD_Falling_Callback()*

57.2.3 Wakeup Pins configuration Functions

This section contains the following APIs:

- *HAL_PWREx_EnableWakeUpPin()*
- *HAL_PWREx_DisableWakeUpPin()*

57.2.4 Memories Retention Functions

This section contains the following APIs:

- *HAL_PWREx_EnableFlashPowerDown()*
- *HAL_PWREx_DisableFlashPowerDown()*
- *HAL_PWREx_EnableMemoryShutOff()*
- *HAL_PWREx_DisableMemoryShutOff()*
- *HAL_PWREx_EnableBkupRAMRetention()*
- *HAL_PWREx_DisableBkupRAMRetention()*

57.2.5 IO and JTAG Retention Functions

In the Standby mode, the I/Os are by default in floating state. If the IORETEN bit in the PWR_IORETR register is set, the I/Os output state is retained. IO Retention mode is enabled for all IO except the IO support the standby functionality and JTAG IOs (PA13, PA14, PA15 and PB4). When entering into Standby mode, the state of the output is sampled, and pull-up or pull-down resistor are set to maintain the IO output during Standby mode. If the JTAGIORETEN bit in the PWR_IORETR register is set, the I/Os output state is retained. IO Retention mode is enabled for PA13, PA14, PA15 and PB4 (default JTAG pullup/ pull-down after wakeup are not enabled).

This section contains the following APIs:

- *HAL_PWREx_EnableStandbyIORetention()*
- *HAL_PWREx_DisableStandbyIORetention()*
- *HAL_PWREx_EnableStandbyJTAGIORetention()*
- *HAL_PWREx_DisableStandbyJTAGIORetention()*

57.2.6 Detailed description of functions

HAL_PWREx_ConfigSupply

Function name

HAL_StatusTypeDef HAL_PWREx_ConfigSupply (uint32_t SupplySource)

Function description

Configure the system Power Supply.

Parameters

- **SupplySource:** : Specifies the Power Supply source to set after a system startup. This parameter can be one of the following values :
  - PWR_EXTERNAL_SOURCE_SUPPLY : The SMPS and the LDO are Bypassed. The Vcore Power Domains are supplied from external source.

Return values

- **HAL:** status.

HAL_PWREx_GetSupplyConfig

Function name

uint32_t HAL_PWREx_GetSupplyConfig (void)

Function description

Get the power supply configuration.

Return values

- **The:** supply configuration.

HAL_PWREx_ControlVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage.

Parameters

- **VoltageScaling:** : Specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values :
  - PWR_REGULATOR_VOLTAGE_SCALE0 : Regulator voltage output Scale 0 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output range 1 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output range 2 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output range 3 mode.

Return values

- **HAL:** Status

HAL_PWREx_GetVoltageRange

Function name

uint32_t HAL_PWREx_GetVoltageRange (void)

Function description

Get the main internal regulator output voltage.

Return values

- **The:** current applied VOS selection.

HAL_PWREx_ControlStopModeVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlStopModeVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage in STOP mode.

Parameters

- **VoltageScaling:** : Specifies the regulator output voltage when the system enters Stop mode to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
  - PWR_REGULATOR_SVOS_SCALE3 : Regulator voltage output range 3 mode.
  - PWR_REGULATOR_SVOS_SCALE4 : Regulator voltage output range 4 mode.
  - PWR_REGULATOR_SVOS_SCALE5 : Regulator voltage output range 5 mode.

Return values

- **HAL:** Status.

Notes

- The Stop mode voltage scaling for SVOS4 and SVOS5 sets the voltage regulator in Low-power (LP) mode to further reduce power consumption. When preselecting SVOS3, the use of the voltage regulator low-power mode (LP) can be selected by LPDS register bit.
- The selected SVOS4 and SVOS5 levels add an additional startup delay when exiting from system Stop mode.

HAL_PWREx_GetStopModeVoltageRange

Function name

uint32_t HAL_PWREx_GetStopModeVoltageRange (void)

Function description

Get the main internal regulator output voltage in STOP mode.

Return values

- **The:** actual applied VOS selection.

HAL_PWREx_ConfigAVD

Function name

void HAL_PWREx_ConfigAVD (const PWREx_AVDTypeDef * sConfigAVD)

Function description

Configure the event mode and the voltage threshold detected by the Analog Voltage Detector (AVD).

Parameters

- **sConfigAVD:** : Pointer to an PWREx_AVDTypeDef structure that contains the configuration information for the AVD.

Return values

- **None.:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

HAL_PWREx_EnableAVD

Function name

void HAL_PWREx_EnableAVD (void)

Function description

Enable the Analog Voltage Detector (AVD).

Return values

- **None.:**

HAL_PWREx_DisableAVD

Function name

void HAL_PWREx_DisableAVD (void)

Function description

Disable the Analog Voltage Detector(AVD).

Return values

- **None.:**

HAL_PWREx_EnableUSBVoltageDetector

Function name

void HAL_PWREx_EnableUSBVoltageDetector (void)

Function description

Enable the USB voltage level detector.

Return values

- **None.:**

HAL_PWREx_DisableUSBVoltageDetector

Function name

void HAL_PWREx_DisableUSBVoltageDetector (void)

Function description

Disable the USB voltage level detector.

Return values

- **None.:**

HAL_PWREx_EnableVddUSB

Function name

void HAL_PWREx_EnableVddUSB (void)

Function description

Enable VDDUSB supply.

Return values

- **None.:**

Notes

- Remove VDDUSB electrical and logical isolation, once VDDUSB supply is present for consumption saving.

HAL_PWREx_DisableVddUSB

Function name

void HAL_PWREx_DisableVddUSB (void)

Function description

Disable VDDUSB supply.

Return values

- **None.:**

HAL_PWREx_EnableMonitoring

Function name

void HAL_PWREx_EnableMonitoring (void)

Function description

Enable the VBAT and temperature monitoring.

Return values

- **None.:**

HAL_PWREx_DisableMonitoring

Function name

void HAL_PWREx_DisableMonitoring (void)

Function description

Disable the VBAT and temperature monitoring.

Return values

- **None.:**

HAL_PWREx_EnableUCPDStandbyMode

Function name

void HAL_PWREx_EnableUCPDStandbyMode (void)

Function description

Enable UCPD configuration memorization in Standby mode.

Return values

- **None.:**

HAL_PWREx_DisableUCPDStandbyMode

Function name

void HAL_PWREx_DisableUCPDStandbyMode (void)

Function description

Disable UCPD configuration memorization in Standby mode.

Return values

- **None.:**

Notes

- This function must be called on exiting the Standby mode and before any UCPD configuration update.

HAL_PWREx_EnableUCPDDeadBattery

Function name

void HAL_PWREx_EnableUCPDDeadBattery (void)

Function description

Enable dead battery behavior.

Return values

- **None.:**

Notes

- After exiting reset, the USB Type-C (dead battery) behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to handover control to the UCPD (the UCPD must be initialized before doing the disable).

HAL_PWREx_DisableUCPDDeadBattery

Function name

void HAL_PWREx_DisableUCPDDeadBattery (void)

Function description

Disable dead battery behavior.

Return values

- **None.:**

Notes

- After exiting reset, the USB Type-C (dead battery) behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to handover control to the UCPD (the UCPD must be initialized before doing the disable).

HAL_PWREx_EnableBatteryCharging

Function name

void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorValue)

Function description

Enable the Battery charging.

Parameters

- **ResistorValue:** : Specifies the charging resistor. This parameter can be one of the following values :
  - PWR_BATTERY_CHARGING_RESISTOR_5 : 5 KOhm resistor.
  - PWR_BATTERY_CHARGING_RESISTOR_1_5 : 1.5 KOhm resistor.

Return values

- **None.:**

Notes

- When VDD is present, charge the external battery through an internal resistor.

HAL_PWREx_DisableBatteryCharging

Function name

void HAL_PWREx_DisableBatteryCharging (void)

Function description

Disable the Battery charging.

Return values

- **None.:**

HAL_PWREx_EnableAnalogBooster

Function name

void HAL_PWREx_EnableAnalogBooster (void)

Function description

Enable the booster to guarantee the analog switch AC performance when the VDD supply voltage is below 2V7.

Return values

- **None.:**

Notes

- The VDD supply voltage can be monitored through the PVD and the PLS field bits.

HAL_PWREx_DisableAnalogBooster

Function name

void HAL_PWREx_DisableAnalogBooster (void)

Function description

Disable the analog booster.

Return values

- **None.:**

HAL_PWREx_PVD_AVD_IRQHandler

Function name

void HAL_PWREx_PVD_AVD_IRQHandler (void)

Function description

This function handles the PWR PVD/AVD interrupt request.

Return values

- **None:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().
- The use of this API is when the PVD and AVD are activated at the same time.

HAL_PWREx_PVD_AVD_Rising_Callback

Function name

void HAL_PWREx_PVD_AVD_Rising_Callback (void)

Function description

PWR PVD AVD Rising interrupt callback.

Return values

- **None.:**

HAL_PWREx_PVD_AVD_Falling_Callback

Function name

void HAL_PWREx_PVD_AVD_Falling_Callback (void)

Function description

PWR PVD AVD Falling interrupt callback.

Return values

- **None.:**

HAL_PWREx_EnableWakeUpPin

Function name

void HAL_PWREx_EnableWakeUpPin (const PWREx_WakeupPinTypeDef * sPinParams)

Function description

Enable the Wake-up PINx functionality.

Parameters

- **sPinParams:** : Pointer to a PWREx_WakeupPinTypeDef structure that contains the configuration information for the wake-up Pin.

Return values

- **None.:**

HAL_PWREx_DisableWakeUpPin

Function name

void HAL_PWREx_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the Wake-up PINx functionality.

Parameters

- **WakeUpPinx:** : Specifies the Wake-Up pin to be disabled. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1
  - PWR_WAKEUP_PIN2
  - PWR_WAKEUP_PIN3
  - PWR_WAKEUP_PIN4
  - PWR_WAKEUP_PIN5
  - PWR_WAKEUP_PIN6
  - PWR_WAKEUP_PIN7
  - PWR_WAKEUP_PIN8

Return values

- **None:**

Notes

- The PWR_WAKEUP_PIN6, PWR_WAKEUP_PIN7 and PWR_WAKEUP_PIN8 are not available for STM32H503xx devices.

HAL_PWREx_EnableFlashPowerDown

Function name

void HAL_PWREx_EnableFlashPowerDown (void)

Function description

Enable the Flash Power Down in Stop mode.

Return values

- **None.:**

Notes

- When Flash Power Down is enabled the Flash memory enters low-power mode. This feature allows to obtain the best trade-off between low-power consumption and restart time when exiting from Stop mode.

HAL_PWREx_DisableFlashPowerDown

Function name

void HAL_PWREx_DisableFlashPowerDown (void)

Function description

Disable the Flash Power Down in Stop mode.

Return values

- **None.:**

Notes

- When Flash Power Down is disabled the Flash memory is kept on normal mode. This feature allows to obtain the best trade-off between low-power consumption and restart time when exiting from Stop mode.

HAL_PWREx_EnableMemoryShutOff

Function name

void HAL_PWREx_EnableMemoryShutOff (uint32_t MemoryBlock)

Function description

Enable memory block shut-off in Stop mode.

Parameters

- **MemoryBlock:** : Specifies the memory block to shut-off during Stop mode. This parameter can be one of the following values:
  - PWR_ETHERNET_MEMORY_BLOCK PWR_PMCR_ETHERNETSO : Ethernet shut-off control in Stop mode
  - PWR_RAM3_MEMORY_BLOCK PWR_PMCR_SRAM3SO : RAM3 shut-off control in Stop mode
  - PWR_RAM2_16_MEMORY_BLOCK PWR_PMCR_SRAM2_16SO : RAM2 16k byte shut-off control in Stop mode
  - PWR_RAM2_48_MEMORY_BLOCK PWR_PMCR_SRAM2_48SO : RAM2 48k byte shut-off control in Stop mode
  - PWR_RAM1_MEMORY_BLOCK PWR_PMCR_SRAM1SO : RAM1 shut-off control in Stop mode

Return values

- **None.:**

Notes

- In Stop mode, the content of the memory blocks is maintained. Further power optimization can be obtained by switching off some memory blocks. This optimization implies loss of the memory content. The user can select which memory is discarded during STOP mode by means of xxSO bits.
- The PWR_ETHERNET_MEMORY_BLOCK is not available for STM32H503xx devices.

HAL_PWREx_DisableMemoryShutOff

Function name

void HAL_PWREx_DisableMemoryShutOff (uint32_t MemoryBlock)

Function description

Disable memory block shut-off in Stop mode.

Parameters

- **MemoryBlock:** : Specifies the memory block to keep content during Stop mode. This parameter can be one of the following values:
  - PWR_ETHERNET_MEMORY_BLOCK PWR_PMCR_ETHERNETSO : Ethernet shut-off control in Stop mode
  - PWR_RAM3_MEMORY_BLOCK PWR_PMCR_SRAM3SO : RAM3 shut-off control in Stop mode
  - PWR_RAM2_16_MEMORY_BLOCK PWR_PMCR_SRAM2_16SO : RAM2 16k byte shut-off control in Stop mode
  - PWR_RAM2_48_MEMORY_BLOCK PWR_PMCR_SRAM2_48SO : RAM2 48k byte shut-off control in Stop mode
  - PWR_RAM1_MEMORY_BLOCK PWR_PMCR_SRAM1SO : RAM1 shut-off control in Stop mode

Return values

- **None.:**

Notes

- The PWR_ETHERNET_MEMORY_BLOCK is not available for STM32H503xx devices.

HAL_PWREx_EnableBkupRAMRetention

Function name

HAL_StatusTypeDef HAL_PWREx_EnableBkupRAMRetention (void)

Function description

Enable the Backup RAM retention in Standby and VBAT modes.

Return values

- **None.:**

Notes

- If BREN is reset, the backup RAM can still be used in Run, Sleep and Stop modes. However, its content is lost in Standby, Shutdown and VBAT modes. This bit can be writte

HAL_PWREx_DisableBkupRAMRetention

Function name

void HAL_PWREx_DisableBkupRAMRetention (void)

Function description

Disable the Backup RAM retention in Standby and VBAT modes.

Return values

- **None.:**

Notes

- If BREN is reset, the backup RAM can still be used in Run, Sleep and Stop modes. However, its content is lost in Standby, Shutdown and VBAT modes. This bit can be write

HAL_PWREx_EnableStandbyIORetention

Function name

void HAL_PWREx_EnableStandbyIORetention (void)

Function description

Enable GPIO state retention in Standby mode.

Return values

- **None.:**

Notes

- When entering into standby mode, the output is sampled, and applied to the output IO during the standby power mode

HAL_PWREx_DisableStandbyIORetention

Function name

void HAL_PWREx_DisableStandbyIORetention (void)

Function description

Disable GPIO state retention in Standby mode.

Return values

- **None.:**

HAL_PWREx_EnableStandbyJTAGIORetention

Function name

void HAL_PWREx_EnableStandbyJTAGIORetention (void)

Function description

Enable JTAG IOs state retention in Standby mode.

Return values

- **None.:**

Notes

- when entering into standby mode, the output is sampled, and applied to the output IO during the standby power mode

HAL_PWREx_DisableStandbyJTAGIORetention

Function name

void HAL_PWREx_DisableStandbyJTAGIORetention (void)

Function description

Disable JTAG IOs state retention in Standby mode.

Return values

- **None.:**

57.3 PWREx Firmware driver defines

The following section lists the various define and macros of the module.

57.3.1 PWREx

PWREx

PWREx AVD detection level

PWR_AVDLEVEL_0

Analog voltage detector level 0 selection : 1V7

PWR_AVDLEVEL_1

Analog voltage detector level 1 selection : 2V1

PWR_AVDLEVEL_2

Analog voltage detector level 2 selection : 2V5

PWR_AVDLEVEL_3

Analog voltage detector level 3 selection : 2V8

PWREx AVD EXTI Line 16

PWR_EXTI_LINE_AVD

External interrupt line 16 Connected to the AVD EXTI Line

PWREx AVD Mode

PWR_AVD_MODE_NORMAL

Basic mode is used

PWR_AVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_AVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_AVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_AVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR Extended AVD Mode Mask

AVD_MODE_IT

AVD_MODE_EVT

AVD_RISING_EDGE

AVD_FALLING_EDGE

AVD_RISING_FALLING_EDGE

PWR Extended Exported Macros

__HAL_PWR_AVD_EXTI_ENABLE_IT

Description:

- Enable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_IT

Description:

- Disable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_EVENT

Description:

- Enable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_EVENT

Description:

- Disable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the AVD Extended Interrupt Rising and Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the AVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_PVD_AVD_EXTI_GET_RISING_FLAG

Description:

- Check whether the specified AVD EXTI Rising interrupt flag is set or not.

Return value:

- EXTI: AVD Line Status.

__HAL_PWR_PVD_AVD_EXTI_GET_FALLING_FLAG

Description:

- Check whether the specified AVD EXTI Falling interrupt flag is set or not.

Return value:

- EXTI: AVD Line Status.

__HAL_PWR_PVD_AVD_EXTI_CLEAR_FLAG

Description:

- Clear the AVD EXTI flag.

Return value:

- None.

__HAL_PWR_AVD_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on AVD EXTI line.

Return value:

- None.

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- Configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: : Specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values :
  - PWR_REGULATOR_VOLTAGE_SCALE0 : Regulator voltage output scale 0. Provides a typical output voltage at 1.2 V. Used when system clock frequency is up to 160 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output scale 1. Provides a typical output voltage at 1.1 V. Used when system clock frequency is up to 100 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output scale 2. Provides a typical output voltage at 1.0 V. Used when system clock frequency is up to 50 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output scale 3. Provides a typical output voltage at 0.9 V. Used when system clock frequency is up to 24 MHz.

Return value:

- None.

Notes:

- This macro is similar to HAL_PWREx_ControlVoltageScaling() API but doesn't check whether or not VOSREADY flag is set. User may resort to __HAL_PWR_GET_FLAG() macro to check VOSF bit state.

Memory shut-off block selection

PWR_ETHERNET_MEMORY_BLOCK

Ethernet shut-off control in Stop mode

PWR_RAM3_MEMORY_BLOCK

RAM3 shut-off control in Stop mode

PWR_RAM2_16_MEMORY_BLOCK

RAM2 16k byte shut-off control in Stop mode

PWR_RAM2_48_MEMORY_BLOCK

RAM2 48k byte shut-off control in Stop mode

PWR_RAM1_MEMORY_BLOCK

RAM1 shut-off control in Stop mode

PWREx Pin Polarity configuration

PWR_PIN_POLARITY_HIGH

PWR_PIN_POLARITY_LOW

PWREx Pin Pull configuration

PWR_PIN_NO_PULL

PWR_PIN_PULL_UP

PWR_PIN_PULL_DOWN

PWREx Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE0

Voltage scaling range 0

PWR_REGULATOR_VOLTAGE_SCALE1

Voltage scaling range 1

PWR_REGULATOR_VOLTAGE_SCALE2

Voltage scaling range 2

PWR_REGULATOR_VOLTAGE_SCALE3

Voltage scaling range 3

PWREx Supply configuration

PWR_EXTERNAL_SOURCE_SUPPLY

The SMPS disabled and the LDO Bypass. The Core domains are supplied from an external source

PWR_SUPPLY_CONFIG_MASK

PWREx System Stop Mode Voltage Scale

PWR_REGULATOR_SVOS_SCALE5

PWR_REGULATOR_SVOS_SCALE4

PWR_REGULATOR_SVOS_SCALE3

PWR Extended Battery Charging Resistor Selection

PWR_BATTERY_CHARGING_RESISTOR_5

VBAT charging through a 5 kOhms resistor

PWR_BATTERY_CHARGING_RESISTOR_1_5

VBAT charging through a 1.5 kOhms resistor

PWREx Wake-Up Pins

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN3

PWR_WAKEUP_PIN4

PWR_WAKEUP_PIN5

PWR_WAKEUP_PIN6

PWR_WAKEUP_PIN7

PWR_WAKEUP_PIN8

PWR_WAKEUP_PIN1_HIGH

PWR_WAKEUP_PIN2_HIGH

PWR_WAKEUP_PIN3_HIGH

PWR_WAKEUP_PIN4_HIGH

PWR_WAKEUP_PIN5_HIGH

PWR_WAKEUP_PIN6_HIGH

PWR_WAKEUP_PIN7_HIGH

PWR_WAKEUP_PIN8_HIGH

PWR_WAKEUP_PIN1_LOW

PWR_WAKEUP_PIN2_LOW

PWR_WAKEUP_PIN3_LOW

PWR_WAKEUP_PIN4_LOW

PWR_WAKEUP_PIN5_LOW

PWR_WAKEUP_PIN6_LOW

PWR_WAKEUP_PIN7_LOW

PWR_WAKEUP_PIN8_LOW

PWREx Wake-Up Pins offsets

PWR_WAKEUP_PINS_PULL_SHIFT_OFFSET

58 HAL RAMCFG Generic Driver

58.1 RAMCFG Firmware driver registers structures

58.1.1 __RAMCFG_HandleTypeDef

__RAMCFG_HandleTypeDef is defined in the stm32h5xx_hal_ramcfg.h

Data Fields

- ***RAMCFG_TypeDef * Instance***
- ***__IO HAL_RAMCFG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DetectSingleErrorCallback***
- ***void(* DetectDoubleErrorCallback***

Field Documentation

- ***RAMCFG_TypeDef* __RAMCFG_HandleTypeDef::Instance***
  RAMCFG Register Base Address
- ***__IO HAL_RAMCFG_StateTypeDef __RAMCFG_HandleTypeDef::State***
  RAMCFG State
- ***__IO uint32_t __RAMCFG_HandleTypeDef::ErrorCode***
  RAMCFG Error Code
- ***void(* __RAMCFG_HandleTypeDef::MspInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP Init Callback
- ***void(* __RAMCFG_HandleTypeDef::MspDeInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP DeInit Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectSingleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Single Error Detect Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectDoubleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Double Error Detect Callback

58.2 RAMCFG Firmware driver API description

The following section lists the various functions of the RAMCFG library.

58.2.1 RAMCFG Peripheral features

- Each SRAM is managed by a RAMCFG instance.
- Each SRAM can be erased independently through its RAMCFG instance.
- The wait state value for each SRAM can be configured independently through its RAMCFG instance.
- SRAM2 is divided to 64 pages with 1 kB granularity. Each page can be write protected independently through its RAMCFG instance.
- SRAM2, SRAM3 and BKPRAM support ECC correction feature. This mechanism adopts the Single Error Correction Double Error Detection (SECDED) algorithm. This feature provides the following information:
  - Single error address.
  - Double error address.

  ==========

58.2.2 How to use this driver

1. Call HAL_RAMCFG_Init() to initialize the RAMCFG peripheral before using any feature. Call HAL_RAMCFG_DeInit() to de-initialize the RAMCFG when using this peripheral is no more needed or a hardware issue has occurred. (+) HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

(#) Call HAL_RAMCFG_Init() to initialize the RAMCFG peripheral before using any feature. Call HAL_RAMCFG_DeInit() to de-initialize the RAMCFG when using this peripheral is no more needed or a hardware issue has occurred.

- HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

ECC feature

- Call HAL_RAMCFG_StartECC() and HAL_RAMCFG_StopECC() to enable and disable ECC hardware mechanism.
  - When ECC feature is previously enabled (case of option byte activation), calling HAL_RAMCFG_StartECC() is recommended to enable the ECC address latching feature.
- Call HAL_RAMCFG_EnableNotification() and HAL_RAMCFG_DisableNotification() to enable and disable ECC interrupts. Interrupts can be:
  - Single error interrupt.
  - Double error interrupt.
  - Double error interrupt redirected to Non maskable interrupt (NMI).
- Call HAL_RAMCFG_GetSingleErrorAddress() to get the address of the last fail RAM word detected (only for single error) and call HAL_RAMCFG_GetDoubleErrorAddress() to get the address of the last fail RAM word detected (only for double error).
- Call HAL_RAMCFG_IsECCErrorDetected() to check if an ECC single/double error was detected. This API is used in silent mode (No ECC interrupt is enabled).

Write protection feature

- Call HAL_RAMCFG_EnableWriteProtection() to enable the write protection for the given SRAM2 page(s).
- There is no API to disable write protection as this feature can be disabled only by a global peripheral reset or system reset.
- Any write access to a write protected area of SRAM2 causes a HardFault interrupt.

Erase feature

- Call HAL_RAMCFG_Erase() to launch a hardware erase for the given SRAM.
- The erase value is equal to 0 when launching erase hardware through RAMCFG.
- SRAM2 write protected pages are erased when performing an erase through RAMCFG.

RAMCFG HAL driver macros list

Below the list of used macros in RAMCFG HAL driver.

- __HAL_RAMCFG_ENABLE_IT : Enable the specified RAMCFG interrupts.
- __HAL_RAMCFG_DISABLE_IT : Disable the specified RAMCFG interrupts.
- __HAL_RAMCFG_GET_FLAG : Get the RAMCFG pending flags.
- __HAL_RAMCFG_CLEAR_FLAG : Clear the RAMCFG pending flags.
- __HAL_RAMCFG_GET_IT_SOURCE : Check whether the specified RAMCFG interrupt source is enabled or not.

58.2.3 Initialization and de-initialization Functions

This section provides functions allowing to initialize and de-initialize the RAMCFG instance.

The HAL_RAMCFG_Init() function follows the RAMCFG instance configuration procedures as described in the reference manual. The HAL_RAMCFG_DeInit() function allows to deinitialize the RAMCFG instance. HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

This section contains the following APIs:

- *HAL_RAMCFG_Init()*
- *HAL_RAMCFG_DeInit()*
- *HAL_RAMCFG_MspInit()*
- *HAL_RAMCFG_MspDeInit()*

58.2.4 ECC Operations Functions

This section provides functions allowing to manage ECC feature provided by the RAMCFG peripheral.

The HAL_RAMCFG_StartECC() function allows starting the ECC mechanism and enabling ECC address latching for the selected RAMCFG instance. The HAL_RAMCFG_StopECC() function allows stopping the ECC mechanism and disabling ECC address latching for the selected RAMCFG instance. The HAL_RAMCFG_EnableNotification() function allows enabling interrupts for single ECC error, double ECC error and NMI error. The HAL_RAMCFG_DisableNotification() function allows disabling interrupts for single ECC error, double ECC error. When NMI interrupt is enabled it can only be disabled by a global peripheral reset or by a system reset. The HAL_RAMCFG_IsECCErrorDetected() function allows to check if an ECC error has occurred. The HAL_RAMCFG_GetSingleErrorAddress() function allows to get the address of the last single ECC error detected. The HAL_RAMCFG_GetDoubleErrorAddress() function allows to get the address of the last double ECC error detected.

This section contains the following APIs:

- *HAL_RAMCFG_StartECC()*
- *HAL_RAMCFG_StopECC()*
- *HAL_RAMCFG_EnableNotification()*
- *HAL_RAMCFG_DisableNotification()*
- *HAL_RAMCFG_IsECCSingleErrorDetected()*
- *HAL_RAMCFG_IsECCDoubleErrorDetected()*
- *HAL_RAMCFG_GetSingleErrorAddress()*
- *HAL_RAMCFG_GetDoubleErrorAddress()*

58.2.5 Write Protection Functions

This section provides functions to enable write protection feature for the page(s) of SRAM2.

The HAL_RAMCFG_EnableWriteProtection() function allows the user to enable the write protection for the page(s) of SRAM2. Disabling SRAM2 page(s) protection is performed only by a global peripheral reset or a by a system reset.

This section contains the following APIs:

- *HAL_RAMCFG_EnableWriteProtection()*

58.2.6 Erase Operation Functions

This section provides functions allowing a hardware erase for the given SRAM.

The HAL_RAMCFG_Erase() function allows a hardware mass erase for the given SRAM. The erase value for all SRAMs is 0.

This section contains the following APIs:

- *HAL_RAMCFG_Erase()*

58.2.7 Handle Interrupt and Callbacks Functions

This section provides functions to handle RAMCFG interrupts and Register / UnRegister the different callbacks.

The HAL_RAMCFG_IRQHandler() function allows the user to handle the active RAMCFG interrupt request. The HAL_RAMCFG_RegisterCallback() function allows the user to register the selected RAMCFG callbacks. The HAL_RAMCFG_UnRegisterCallback() function allows the user to unregister the selected RAMCFG callbacks.

This section contains the following APIs:

- *HAL_RAMCFG_IRQHandler()*
- *HAL_RAMCFG_DetectSingleErrorCallback()*
- *HAL_RAMCFG_DetectDoubleErrorCallback()*
- *HAL_RAMCFG_RegisterCallback()*
- *HAL_RAMCFG_UnRegisterCallback()*

58.2.8 State and Error Functions

This section provides functions to check and get the RAMCFG state and the error code.

The HAL_RAMCFG_GetState() function allows the user to get the RAMCFG peripheral state. The HAL_RAMCFG_GetError() function allows the user to get the RAMCFG peripheral error code.

This section contains the following APIs:

- *HAL_RAMCFG_GetState()*
- *HAL_RAMCFG_GetError()*

58.2.9 Detailed description of functions

HAL_RAMCFG_Init

Function name

HAL_StatusTypeDef HAL_RAMCFG_Init (RAMCFG_HandleTypeDef * hramcfg)

Function description

Initialize the RAMCFG by clearing flags and disabling interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_DeInit

Function name

HAL_StatusTypeDef HAL_RAMCFG_DeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

DeInitialize the RAMCFG peripheral.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_MspInit

Function name

void HAL_RAMCFG_MspInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

Initialize the RAMCFG MSP.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_MspDeInit

Function name

void HAL_RAMCFG_MspDeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

DeInitialize the RAMCFG MSP.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_StartECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StartECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

Start ECC mechanism for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_StopECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StopECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

Stop ECC mechanism for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_EnableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_EnableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

Enable the RAMCFG error interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **Notifications:** : Select the notification to be enabled. This parameter can be any value of RAMCFG Interrupts group.

Return values

- **HAL:** status.

HAL_RAMCFG_DisableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_DisableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

Disable the RAMCFG error interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **Notifications:** : Select the notification to be disabled. This parameter can be : RAMCFG_IT_SINGLEERR : Single Error Interrupt. RAMCFG_IT_DOUBLEERR : Double Error Interrupt.

Return values

- **HAL:** status.

HAL_RAMCFG_IsECCSingleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCSingleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Check if an ECC single error has occurred.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **State:** of bit (1 or 0).

HAL_RAMCFG_IsECCDoubleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCDoubleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Check if an ECC double error was occurred.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **State:** of bit (1 or 0).

HAL_RAMCFG_GetSingleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetSingleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG single error address.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **Single:** error address offset.

HAL_RAMCFG_GetDoubleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetDoubleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG double error address.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **Double:** error address offset.

HAL_RAMCFG_EnableWriteProtection

Function name

HAL_StatusTypeDef HAL_RAMCFG_EnableWriteProtection (RAMCFG_HandleTypeDef * hramcfg, uint32_t StartPage, uint32_t NbPage)

Function description

Enable write protection for the given page(s).

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **StartPage:** : Select the start page number.
- **NbPage:** : Number of page to be protected.

Return values

- **HAL:** status.

HAL_RAMCFG_Erase

Function name

HAL_StatusTypeDef HAL_RAMCFG_Erase (RAMCFG_HandleTypeDef * hramcfg)

Function description

Launch a Mass Erase for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_IRQHandler

Function name

void HAL_RAMCFG_IRQHandler (RAMCFG_HandleTypeDef * hramcfg)

Function description

Handles RAMCFG interrupt request.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **None.:**

HAL_RAMCFG_DetectSingleErrorCallback

Function name

void HAL_RAMCFG_DetectSingleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

RAMCFG single error detection callback.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_DetectDoubleErrorCallback

Function name

void HAL_RAMCFG_DetectDoubleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

RAMCFG double error detection callback.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_RegisterCallback (RAMCFG_HandleTypeDef * hramcfg, HAL_RAMCFG_CallbackIDTypeDef CallbackID, void(*)(RAMCFG_HandleTypeDef *_hramcfg) pCallback)

Function description

Register RAMCFG callbacks.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **CallbackID:** : User Callback identifier a HAL_RAMCFG_CallbackIDTypeDef ENUM as parameter.
- **pCallback:** : Pointer to private callback function.

Return values

- **HAL:** status.

HAL_RAMCFG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_UnRegisterCallback (RAMCFG_HandleTypeDef * hramcfg, HAL_RAMCFG_CallbackIDTypeDef CallbackID)

Function description

UnRegister RAMCFG callbacks.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **CallbackID:** : User Callback identifier a HAL_RAMCFG_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status.

HAL_RAMCFG_GetError

Function name

uint32_t HAL_RAMCFG_GetError (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG peripheral error code.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **RAMCFG:** error code.

HAL_RAMCFG_GetState

Function name

HAL_RAMCFG_StateTypeDef HAL_RAMCFG_GetState (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG peripheral state.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **RAMCFG:** state.

58.3 RAMCFG Firmware driver defines

The following section lists the various define and macros of the module.

58.3.1 RAMCFG

RAMCFG

RAMCFG Error Codes

HAL_RAMCFG_ERROR_NONE

RAMCFG No Error

HAL_RAMCFG_ERROR_TIMEOUT

RAMCFG Timeout Error

HAL_RAMCFG_ERROR_BUSY

RAMCFG Busy Error

HAL_RAMCFG_ERROR_INVALID_CALLBACK

Invalid Callback error

RAMCFG Exported Macros

__HAL_RAMCFG_ENABLE_IT

Description:

- Enable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be enabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt redirection to NMI Mask.
  - RAMCFG_IT_ALL : All Interrupt Mask.

Return value:

- None

__HAL_RAMCFG_DISABLE_IT

Description:

- Disable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be disabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.

Return value:

- None

Notes:

- This macros is used only to disable RAMCFG_IT_SINGLEERR and RAMCFG_IT_DOUBLEERR interrupts. RAMCFG_IT_NMIERR interrupt can only be disabled by global peripheral reset or system reset.

__HAL_RAMCFG_GET_IT_SOURCE

Description:

- Check whether the specified RAMCFG interrupt source is enabled or not.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.
- __INTERRUPT__: : Specifies the RAMCFG interrupt source to check. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt Redirection to NMI Mask.

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RAMCFG_GET_FLAG

Description:

- Get the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be checked. This parameter can be one of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.
  - RAMCFG_FLAG_SRAMBUSY : SRAM Busy Flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_RAMCFG_CLEAR_FLAG

Description:

- Clear the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be cleared. This parameter can be any combination of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.

Return value:

- None.

__HAL_RAMCFG_RESET_HANDLE_STATE

Description:

- Reset the RAMCFG handle state.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.

Return value:

- None.

RAMCFG Monitor Flags

RAMCFG_FLAG_SINGLEERR

RAMCFG Single Error Detected and Corrected Flag

RAMCFG_FLAG_DOUBLEERR

RAMCFG Double Error Detected Flag

RAMCFG_FLAG_SRAMBUSY

RAMCFG SRAM busy Flag

RAMCFG_FLAGS_ALL

RAMCFG All Flags

RAMCFG Interrupts

RAMCFG_IT_SINGLEERR

RAMCFG Single Error Interrupt

RAMCFG_IT_DOUBLEERR

RAMCFG Double Error Interrupt

RAMCFG_IT_NMIERR

RAMCFG Double Error redirected to NMI Interrupt

RAMCFG_IT_ALL

RAMCFG All RAMCFG interrupt

RAMCFG Keys

RAMCFG_ERASE_KEY1

RAMCFG launch Erase Key 1

RAMCFG_ERASE_KEY2

RAMCFG launch Erase Key 2

RAMCFG_ECC_KEY1

RAMCFG launch ECC Key 1

RAMCFG_ECC_KEY2

RAMCFG launch ECC Key 2

59 HAL RCC Generic Driver

59.1 RCC Firmware driver registers structures

59.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32h5xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***
- ***uint32_t PLLR***
- ***uint32_t PLLRGE***
- ***uint32_t PLLVCOSEL***
- ***uint32_t PLLFRACN***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  PLLState: The new state of the PLL1. This parameter can be a value of ***RCC_PLL1_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL1_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***
  PLLM: Division factor for PLL1 VCO input clock. This parameter must be a number between Min_Data = 0 and Max_Data = 63
- ***uint32_t RCC_PLLInitTypeDef::PLLN***
  PLLN: Multiplication factor for PLL1 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLLInitTypeDef::PLLP***
  PLLP: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLLInitTypeDef::PLLQ***
  PLLQ: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLR***
  PLLR: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLRGE***
  PLLRGE: PLL1 clock Input range This parameter must be a value of ***RCC_PLL1_VCI_Range***
- ***uint32_t RCC_PLLInitTypeDef::PLLVCOSEL***
  PLLVCOSEL: PLL1 clock Output range This parameter must be a value of ***RCC_PLL1_VCO_Range***
- ***uint32_t RCC_PLLInitTypeDef::PLLFRACN***
  PLLFRACN: Specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

59.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32h5xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***

- ***uint32_t HSIState***
- ***uint32_t HSIDiv***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t CSIState***
- ***uint32_t CSICalibrationValue***
- ***uint32_t HSI48State***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIDiv***
  The division factor of the HSI. This parameter can be a value of ***RCC_HSI_Div***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F on the other devices
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::CSIState***
  The new state of the CSI. This parameter can be a value of ***RCC_CSI_Config***
- ***uint32_t RCC_OscInitTypeDef::CSICalibrationValue***
  The calibration trimming value (default is RCC_CSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F
- ***uint32_t RCC_OscInitTypeDef::HSI48State***
  The new state of the HSI48. This parameter can be a value of ***RCC_HSI48_Config***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  PLL1 structure parameters

59.1.3 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32h5xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB3CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source used as system clock (SYSCLK). This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***

- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB3CLKDivider***
  The APB3 clock (PCLK3) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***

59.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

59.2.1 RCC specific features

After reset the device is running from High Speed Internal oscillator (64 MHz) with Flash 3 wait states. Flash prefetch buffer, D-Cache and I-Cache are disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHBs) and Low speed (APBs) busses: all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analog mode, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (SAIx, RTC, ADC, USB, SDMMC, etc.)

59.2.2 Initialization and de-initialization functions

This section provides functions allowing to configure the internal and external oscillators (HSE, HSI, LSE, CSI, LSI, PLL1, HSE CSS and MCOs) and the System busses clocks (SYSCLK, AHB, APB1, APB2 and APB3).

Internal/external clock and PLL configuration (+) HSI (high-speed internal): 64 MHz factory-trimmed RC used directly or through the PLL as System clock source.

1. CSI is a low-power RC oscillator which can be used directly as system clock, peripheral clock, or PLL input. But even with frequency calibration, is less accurate than an external crystal oscillator or ceramic resonator. (+) LSI (low-speed internal): 32 KHz low consumption RC used as IWDG and/or RTC clock source. (+) HSE (high-speed external): 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also optionally as RTC clock source. (+) LSE (low-speed external): 32.768 KHz oscillator used optionally as RTC clock source. (+) PLL1 (clocked by HSI, HSE or CSI) providing up to three independent output clocks:
   – The first output is used to generate the high speed system clock (up to 250MHz).
   – The second output is used to generate the clock for the USB (48 MHz), the FDCAN1/2, the SPI1/2/3, the OCTOSPI, the RNG (<=48 MHz), the SDMMC1/2 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface. (+) PLL2 (clocked by HSI, HSE or CSI) providing up to three independent output clocks:
   – The first output is used to generate the clock for the LPTIMs, the SPI1/2/3 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface.
   – The second output is used to generate the clock for USARTs, the UARTs, the LPUART1, the FDCAN1/2, the SPI4/5/6 and the USB.
   – The third output is used to generate the clock the SDMMC1/2, the ADC/DAC, the I2C1/2, the I3C1/2 and the OCTOSPI. (+) PLL3 (clocked by HSI , HSE or CSI) providing up to three independent output clocks:
   – The first output is used to generate the clock for SPI1/2/3 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface.
   – The second output is used to generate the clock for USARTs, the UARTs, the LPUART1, the SPI4/5/6 and the USB.
   – The third output is used to generate the clock for the I2Cs, the I3Cs and the LPTIMs. (+) HSE CSS (HSE Clock Security System): once enabled, if a HSE clock failure occurs (HSE used directly or through PLL1 as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M33 NMI (Non-Maskable Interrupt) exception vector.
2. MCO1 (micro controller clock output1), used to output HSI, LSE, HSE, PLL1(PLL1_Q) or HSI48 clock (through a configurable pre-scaler) on PA8 pin.
3. MCO2 (micro controller clock output2), used to output HSE, PLL2(PLL2_P), SYSCLK, LSI, CSI, or PLL1(PLL1_P) clock (through a configurable pre-scaler) on PC9 pin.

Internal/external clock and PLL configuration

- HSI (high-speed internal): 64 MHz factory-trimmed RC used directly or through the PLL as System clock source. (#) CSI is a low-power RC oscillator which can be used directly as system clock, peripheral clock, or PLL input. But even with frequency calibration, is less accurate than an external crystal oscillator or ceramic resonator.
- LSI (low-speed internal): 32 KHz low consumption RC used as IWDG and/or RTC clock source.
- HSE (high-speed external): 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also optionally as RTC clock source.
- LSE (low-speed external): 32.768 KHz oscillator used optionally as RTC clock source.
- PLL1 (clocked by HSI, HSE or CSI) providing up to three independent output clocks:
  – The first output is used to generate the high speed system clock (up to 250MHz).
  – The second output is used to generate the clock for the USB (48 MHz), the FDCAN1/2, the SPI1/2/3, the OCTOSPI, the RNG (<=48 MHz), the SDMMC1/2 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface.
- PLL2 (clocked by HSI, HSE or CSI) providing up to three independent output clocks:
  – The first output is used to generate the clock for the LPTIMs, the SPI1/2/3 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface.
  – The second output is used to generate the clock for USARTs, the UARTs, the LPUART1, the FDCAN1/2, the SPI4/5/6 and the USB.
  – The third output is used to generate the clock the SDMMC1/2, the ADC/DAC, the I2C1/2, the I3C1/2 and the OCTOSPI.

- PLL3 (clocked by HSI , HSE or CSI) providing up to three independent output clocks:
  - The first output is used to generate the clock for SPI1/2/3 and to generate an accurate clock to achieve high-quality audio performance on SAI1/2 interface.
  - The second output is used to generate the clock for USARTs, the UARTs, the LPUART1, the SPI4/5/6 and the USB.
  - The third output is used to generate the clock for the I2Cs, the I3Cs and the LPTIMs.
- HSE CSS (HSE Clock Security System): once enabled, if a HSE clock failure occurs (HSE used directly or through PLL1 as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M33 NMI (Non-Maskable Interrupt) exception vector. (#) MCO1 (micro controller clock output1), used to output HSI, LSE, HSE, PLL1(PLL1_Q) or HSI48 clock (through a configurable pre-scaler) on PA8 pin. (#) MCO2 (micro controller clock output2), used to output HSE, PLL2(PLL2_P), SYSCLK, LSI, CSI, or PLL1(PLL1_P) clock (through a configurable pre-scaler) on PC9 pin.

System, AHB and APB busses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): CSI, HSI, HSE and the main PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1), APB2 (PCLK2) and APB3 (PCLK3) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks.

Note: All the peripheral clocks are derived from the System clock (SYSCLK) except:

- *SAI: the SAI clock can be derived either from specific PLL (PLL1, PLL2 or PLL3), the per_ck clock (HSE, HSI or CSI) or from an external clock mapped on the SAI_CKIN pin. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *SPI/I2S: the SPI1/2/3 clock can be derived either from specific PLL (PLL1, PLL2 or PLL3), the per_ck clock (HSE, HSI or CSI) or from an external clock mapped on the SPI_CKIN pin. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use __HAL_RCC_RTC_ENABLE() and HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *USB: USB requires a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL or PLL2 through PLLQ divider. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *UCPD: the UCPD clock is derived from HSI (divided by 4) clock.*
- *SDMMC: SDMMC1/2 peripherals require a frequency equal or lower than 48 MHz. This clock is derived from the PLL1 or PLL2 through PLL1Q or PLL2R divider. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *IWDG clock which is always the LSI clock. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *RNG: the RNG clock can be derived either from PLL1Q, HSI48, LSE or LSI clock. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *DAC: the DAC clock can be derived either from LSE or LSI clock. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *FDCAN: the FDCAN1/2 clock can be derived either from HSE, PLL1Q or PLL2Q clock. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *CEC: the CEC clock can be derived either from LSE, LSI or CSI (divided by 122) clock.You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *ETH: the Ethernet clock is derived from PLL1Q clock.*

- The maximum frequency of the SYSCLK, HCLK, PCLK1, PCLK2 and PCLK3 is 250 MHz. The clock source frequency should be adapted depending on the device voltage range as listed in the Reference Manual "Clock source frequency versus voltage scaling" chapter.

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

59.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to:

- Output clock to MCO pin.
- Retrieve current clock frequencies.
- Enable the Clock Security System.

This section contains the following APIs:

- *HAL_RCC_MCOConfig()*
- *HAL_RCC_GetSysClockFreq()*
- *HAL_RCC_GetHCLKFreq()*
- *HAL_RCC_GetPCLK1Freq()*
- *HAL_RCC_GetPCLK2Freq()*
- *HAL_RCC_GetPCLK3Freq()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_GetResetSource()*
- *HAL_RCC_EnableCSS()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*

59.2.4 RCC attributes functions

This subsection provides a set of functions allowing to:

- Configure the RCC item(s) attributes.
- Get the attribute of an RCC item.

This section contains the following APIs:

- *HAL_RCC_ConfigAttributes()*
- *HAL_RCC_GetConfigAttributes()*

59.2.5 Detailed description of functions

HAL_RCC_DeInit

Function name

HAL_StatusTypeDef HAL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **HAL:** Status.

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, CSI, PLL, PLL2 and PLL3 OFFAHB, APB1 and APB2 prescaler set to 1.HSECSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (const RCC_OscInitTypeDef * pOscInitStruct)

Function description

Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **pOscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (const RCC_ClkInitTypeDef * pClkInitStruct, uint32_t FLatency)

Function description

Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the pClkInitStruct.

Parameters

- **pClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0 FLASH 0 Latency cycle
  - FLASH_LATENCY_1 FLASH 1 Latency cycle
  - FLASH_LATENCY_2 FLASH 2 Latency cycles
  - FLASH_LATENCY_3 FLASH 3 Latency cycles
  - FLASH_LATENCY_4 FLASH 4 Latency cycles
  - FLASH_LATENCY_5 FLASH 5 Latency cycles

Return values

- **HAL:** Status.

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The HSI is used by default as system clock source after startup from Reset, wake-up from STANDBY mode. After restart from Reset, the HSI frequency is set to its default value 64 MHz.
- The HSI or CSI can be selected as system clock source after wake-up from STOP modes or in case of failure of the HSE when used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source is ready.
- You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.

HAL_RCC_MCOConfig

Function name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function description

Select the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. For STM32H5xx family this parameter can have only one value:
  - RCC_MCO1 Clock source to output on MCO1 pin(PA8).
  - RCC_MCO2 Clock source to output on MCO2 pin(PC9).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_PLL1QCLK: PLL1Q clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI48: HSI48 (48MHZ) selected as MCO1 source
  - RCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC_MCO2SOURCE_PLL2PCLK: PLL2P clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_PLL1PCLK: PLL1P clock selected as MCO2 source
  - RCC_MCO2SOURCE_CSI: CSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSI: LSI clock selected as MCO2 source
- **RCC_MCODiv:** specifies the MCO prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCOx clock

Return values

- **None:**

Notes

- PA8/PC9 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name

void HAL_RCC_EnableCSS (void)

Function description

Enable the HSE Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M NMI (Non-Maskable Interrupt) exception vector.
- The Clock Security System can only be cleared by reset.

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Return the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function may not be the real frequency in the chip. It is calculated based on the predefined constants of the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)
- If SYSCLK source is CSI, function returns values based on CSI_VALUE(**)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(***)
- If SYSCLK source is PLL, function returns values based on HSI_VALUE(*), CSI_VALUE(**) or HSE_VALUE(***) multiplied/divided by the PLL factors.
- (*) HSI_VALUE is a constant defined in stm32h5xx_hal_conf.h file (default value 64 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) CSI_VALUE is a constant defined in stm32h5xx_hal_conf.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (***) HSE_VALUE is a constant defined in stm32h5xx_hal_conf.h file (default value 24 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

uint32_t HAL_RCC_GetHCLKFreq (void)

Function description

Return the HCLK frequency.

Return values

- **HCLK:** frequency in Hz

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency.

HAL_RCC_GetPCLK1Freq

Function name

uint32_t HAL_RCC_GetPCLK1Freq (void)

Function description

Return the PCLK1 frequency.

Return values

- **PCLK1:** frequency in Hz

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

uint32_t HAL_RCC_GetPCLK2Freq (void)

Function description

Return the PCLK2 frequency.

Return values

- **PCLK2:** frequency in Hz

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK3Freq

Function name

uint32_t HAL_RCC_GetPCLK3Freq (void)

Function description

Return the PCLK3 frequency.

Return values

- **PCLK3:** frequency in Hz

Notes

- Each time PCLK3 changes, this function must be called to update the right PCLK3 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * pOscInitStruct)

Function description

Configure the pOscInitStruct according to the internal RCC configuration registers.

Parameters

- **pOscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * pClkInitStruct, uint32_t * pFLatency)

Function description

Configure the pClkInitStruct according to the internal RCC configuration registers.

Parameters

- **pClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

HAL_RCC_NMI_IRQHandler

Function name

void HAL_RCC_NMI_IRQHandler (void)

Function description

Handle the RCC Clock Security System interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name

void HAL_RCC_CSSCallback (void)

Function description

RCC HSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCC_GetResetSource

Function name

uint32_t HAL_RCC_GetResetSource (void)

Function description

Get and clear reset flags.

Return values

- **can:** be a combination of Reset Flag

Notes

- Once reset flags are retrieved, this API is clearing them in order to isolate next reset reason.

HAL_RCC_ConfigAttributes

Function name

void HAL_RCC_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the RCC item(s) attribute(s).

Parameters

- **Item:** Item(s) to set attributes on. This parameter can be a one or a combination of RCC items (**).
- **Attributes:** specifies the RCC secure/privilege attributes. This parameter can be a value of RCC attributes

Return values

- **None:** (*) : For stm32h503xx devices, attributes specifies the privilege attribute only (no items). (**) : For stm32h503xx devices, this parameter is unused, it can take 0 or any other numerical value.

Notes

- Available attributes are to secure items and set RCC as privileged (*). Default state is non-secure and unprivileged access allowed.
- Secure and non-secure attributes can only be set from the secure state when the system implements the security (TZEN=1).

HAL_RCC_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_RCC_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get the attribute of an RCC item.

Parameters

- **Item:** Single item to get secure/non-secure and privilege/non-privilege attribute from. This parameter can be a one value of RCC items except RCC_ALL. (*)
- **pAttributes:** pointer to return the attributes.

Return values

- **HAL:** Status.

Notes

- Secure and non-secure attributes are only available from secure state when the system implements the security (TZEN=1)

59.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

59.3.1 RCC

RCC

AHB1 Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_GPDMA1_FORCE_RESET

__HAL_RCC_GPDMA2_FORCE_RESET

__HAL_RCC_CORDIC_FORCE_RESET

__HAL_RCC_FMAC_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_RAMCFG_FORCE_RESET

__HAL_RCC_ETH_FORCE_RESET

__HAL_RCC_GTZC1_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_GPDMA1_RELEASE_RESET

__HAL_RCC_GPDMA2_RELEASE_RESET

__HAL_RCC_CORDIC_RELEASE_RESET

__HAL_RCC_FMAC_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_RAMCFG_RELEASE_RESET

__HAL_RCC_ETH_RELEASE_RESET

__HAL_RCC_GTZC1_RELEASE_RESET

AHB1 Peripheral Clock Enable Disable

__HAL_RCC_GPDMA1_CLK_ENABLE

__HAL_RCC_GPDMA2_CLK_ENABLE

__HAL_RCC_CORDIC_CLK_ENABLE

__HAL_RCC_FMAC_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_RAMCFG_CLK_ENABLE

__HAL_RCC_FLASH_CLK_ENABLE

__HAL_RCC_ETH_CLK_ENABLE

__HAL_RCC_ETHTX_CLK_ENABLE

__HAL_RCC_ETHRX_CLK_ENABLE

__HAL_RCC_GTZC1_CLK_ENABLE

__HAL_RCC_BKPRAM_CLK_ENABLE

__HAL_RCC_DCACHE1_CLK_ENABLE

__HAL_RCC_SRAM1_CLK_ENABLE

__HAL_RCC_GPDMA1_CLK_DISABLE

__HAL_RCC_GPDMA2_CLK_DISABLE

__HAL_RCC_CORDIC_CLK_DISABLE

__HAL_RCC_FMAC_CLK_DISABLE

__HAL_RCC_FLASH_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_RAMCFG_CLK_DISABLE

__HAL_RCC_ETH_CLK_DISABLE

__HAL_RCC_ETHTX_CLK_DISABLE

__HAL_RCC_ETHRX_CLK_DISABLE

__HAL_RCC_GTZC1_CLK_DISABLE

__HAL_RCC_BKPRAM_CLK_DISABLE

__HAL_RCC_DCACHE1_CLK_DISABLE

__HAL_RCC_SRAM1_CLK_DISABLE

AHB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPDMA1_IS_CLK_ENABLED

__HAL_RCC_GPDMA2_IS_CLK_ENABLED

__HAL_RCC_FLASH_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_CORDIC_IS_CLK_ENABLED

__HAL_RCC_FMAC_IS_CLK_ENABLED

__HAL_RCC_RAMCFG_IS_CLK_ENABLED

__HAL_RCC_ETH_IS_CLK_ENABLED

__HAL_RCC_ETHTX_IS_CLK_ENABLED

__HAL_RCC_ETHRX_IS_CLK_ENABLED

__HAL_RCC_GTZC1_IS_CLK_ENABLED

__HAL_RCC_BKPRAM_IS_CLK_ENABLED

__HAL_RCC_DCACHE1_IS_CLK_ENABLED

__HAL_RCC_SRAM1_IS_CLK_ENABLED

__HAL_RCC_GPDMA1_IS_CLK_DISABLED

__HAL_RCC_GPDMA2_IS_CLK_DISABLED

__HAL_RCC_FLASH_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

__HAL_RCC_CORDIC_IS_CLK_DISABLED

__HAL_RCC_FMAC_IS_CLK_DISABLED

__HAL_RCC_RAMCFG_IS_CLK_DISABLED

__HAL_RCC_ETH_IS_CLK_DISABLED

__HAL_RCC_ETHTX_IS_CLK_DISABLED

__HAL_RCC_ETHRX_IS_CLK_DISABLED

__HAL_RCC_GTZC1_IS_CLK_DISABLED

__HAL_RCC_BKPRAM_IS_CLK_DISABLED

__HAL_RCC_DCACHE1_IS_CLK_DISABLED

__HAL_RCC_SRAM1_IS_CLK_DISABLED

AHB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPDMA1_CLK_SLEEP_ENABLE

__HAL_RCC_GPDMA2_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_CORDIC_CLK_SLEEP_ENABLE

__HAL_RCC_FMAC_CLK_SLEEP_ENABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_ENABLE

__HAL_RCC_ETH_CLK_SLEEP_ENABLE

__HAL_RCC_ETHTX_CLK_SLEEP_ENABLE

__HAL_RCC_ETHRX_CLK_SLEEP_ENABLE

__HAL_RCC_GTZC1_CLK_SLEEP_ENABLE

__HAL_RCC_BKPRAM_CLK_SLEEP_ENABLE

__HAL_RCC_ICACHE_CLK_SLEEP_ENABLE

__HAL_RCC_DCACHE1_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_GPDMA1_CLK_SLEEP_DISABLE

__HAL_RCC_GPDMA2_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_CORDIC_CLK_SLEEP_DISABLE

__HAL_RCC_FMAC_CLK_SLEEP_DISABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_DISABLE

__HAL_RCC_ETH_CLK_SLEEP_DISABLE

__HAL_RCC_ETHTX_CLK_SLEEP_DISABLE

__HAL_RCC_ETHRX_CLK_SLEEP_DISABLE

__HAL_RCC_GTZC1_CLK_SLEEP_DISABLE

__HAL_RCC_BKPRAM_CLK_SLEEP_DISABLE

__HAL_RCC_ICACHE_CLK_SLEEP_DISABLE

__HAL_RCC_DCACHE1_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM1_CLK_SLEEP_DISABLE

AHB2 Peripheral Force Release Reset

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_GPIOI_FORCE_RESET

__HAL_RCC_ADC_FORCE_RESET

__HAL_RCC_DAC1_FORCE_RESET

__HAL_RCC_DCMI_PSSI_FORCE_RESET

__HAL_RCC_DCMI_FORCE_RESET

__HAL_RCC_AES_FORCE_RESET

__HAL_RCC_HASH_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_PKA_FORCE_RESET

__HAL_RCC_SAES_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

__HAL_RCC_GPIOI_RELEASE_RESET

__HAL_RCC_ADC_RELEASE_RESET

__HAL_RCC_DAC1_RELEASE_RESET

__HAL_RCC_DCMI_PSSI_RELEASE_RESET

__HAL_RCC_DCMI_RELEASE_RESET

__HAL_RCC_AES_RELEASE_RESET

__HAL_RCC_HASH_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

__HAL_RCC_PKA_RELEASE_RESET

__HAL_RCC_SAES_RELEASE_RESET

AHB2 Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOI_CLK_ENABLE

__HAL_RCC_ADC_CLK_ENABLE

__HAL_RCC_DAC1_CLK_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_ENABLE

__HAL_RCC_DCMI_CLK_ENABLE

__HAL_RCC_AES_CLK_ENABLE

__HAL_RCC_HASH_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_PKA_CLK_ENABLE

__HAL_RCC_SAES_CLK_ENABLE

__HAL_RCC_SRAM2_CLK_ENABLE

__HAL_RCC_SRAM3_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

__HAL_RCC_GPIOI_CLK_DISABLE

__HAL_RCC_ADC_CLK_DISABLE

__HAL_RCC_DAC1_CLK_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_DISABLE

__HAL_RCC_DCMI_CLK_DISABLE

__HAL_RCC_AES_CLK_DISABLE

__HAL_RCC_HASH_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_PKA_CLK_DISABLE

__HAL_RCC_SAES_CLK_DISABLE

__HAL_RCC_SRAM2_CLK_DISABLE

__HAL_RCC_SRAM3_CLK_DISABLE

AHB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPIOI_IS_CLK_ENABLED

__HAL_RCC_ADC_IS_CLK_ENABLED

__HAL_RCC_DAC1_IS_CLK_ENABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_ENABLED

__HAL_RCC_DCMI_IS_CLK_ENABLED

__HAL_RCC_AES_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_PKA_IS_CLK_ENABLED

__HAL_RCC_SAES_IS_CLK_ENABLED

__HAL_RCC_SRAM2_IS_CLK_ENABLED

__HAL_RCC_SRAM3_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_GPIOH_IS_CLK_DISABLED

__HAL_RCC_GPIOI_IS_CLK_DISABLED

__HAL_RCC_ADC_IS_CLK_DISABLED

__HAL_RCC_DAC1_IS_CLK_DISABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_DISABLED

__HAL_RCC_DCMI_IS_CLK_DISABLED

__HAL_RCC_AES_IS_CLK_DISABLED

__HAL_RCC_HASH_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

__HAL_RCC_PKA_IS_CLK_DISABLED

__HAL_RCC_SAES_IS_CLK_DISABLED

__HAL_RCC_SRAM2_IS_CLK_DISABLED

__HAL_RCC_SRAM3_IS_CLK_DISABLED

AHB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_ADC_CLK_SLEEP_ENABLE

__HAL_RCC_DAC1_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_CLK_SLEEP_ENABLE

__HAL_RCC_AES_CLK_SLEEP_ENABLE

__HAL_RCC_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_PKA_CLK_SLEEP_ENABLE

__HAL_RCC_SAES_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM3_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_ADC_CLK_SLEEP_DISABLE

__HAL_RCC_DAC1_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_CLK_SLEEP_DISABLE

__HAL_RCC_AES_CLK_SLEEP_DISABLE

__HAL_RCC_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_PKA_CLK_SLEEP_DISABLE

__HAL_RCC_SAES_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM3_CLK_SLEEP_DISABLE

AHB4 Peripheral Clock Enable Disable

__HAL_RCC_OTFDEC1_CLK_ENABLE

__HAL_RCC_SDMMC1_CLK_ENABLE

__HAL_RCC_SDMMC2_CLK_ENABLE

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_OSPI1_CLK_ENABLE

__HAL_RCC_OTFDEC1_CLK_DISABLE

__HAL_RCC_SDMMC1_CLK_DISABLE

__HAL_RCC_SDMMC2_CLK_DISABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_OSPI1_CLK_DISABLE

AHB4 Peripheral Clock Sleep Enable Disable

__HAL_RCC_OTFDEC1_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_OSPI1_CLK_SLEEP_ENABLE

__HAL_RCC_OTFDEC1_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_OSPI1_CLK_SLEEP_DISABLE

AHB4 Peripheral Force Release Reset

__HAL_RCC_AHB4_FORCE_RESET

__HAL_RCC_OTFDEC1_FORCE_RESET

__HAL_RCC_SDMMC1_FORCE_RESET

__HAL_RCC_SDMMC2_FORCE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_OSPI1_FORCE_RESET

__HAL_RCC_AHB4_RELEASE_RESET

__HAL_RCC_OTFDEC1_RELEASE_RESET

__HAL_RCC_SDMMC1_RELEASE_RESET

__HAL_RCC_SDMMC2_RELEASE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_OSPI1_RELEASE_RESET

AHB4 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_OTFDEC1_IS_CLK_ENABLED

__HAL_RCC_OSPI1_IS_CLK_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_OTFDEC1_IS_CLK_DISABLED

__HAL_RCC_OSPI1_IS_CLK_DISABLED

__HAL_RCC_SDMMC1_IS_CLK_DISABLED

__HAL_RCC_SDMMC2_IS_CLK_DISABLED

__HAL_RCC_FMC_IS_CLK_DISABLED

AHB APB Branch Clock Disable Clear Disable

__HAL_RCC_AHB1_CLK_DISABLE

__HAL_RCC_AHB2_CLK_DISABLE

__HAL_RCC_AHB4_CLK_DISABLE

__HAL_RCC_APB1_CLK_DISABLE

__HAL_RCC_APB2_CLK_DISABLE

__HAL_RCC_APB3_CLK_DISABLE

__HAL_RCC_AHB1_CLK_ENABLE

__HAL_RCC_AHB2_CLK_ENABLE

__HAL_RCC_AHB4_CLK_ENABLE

__HAL_RCC_APB1_CLK_ENABLE

__HAL_RCC_APB2_CLK_ENABLE

__HAL_RCC_APB3_CLK_ENABLE

AHB APB Branch Clock Disabled Status

__HAL_RCC_AHB1_IS_CLK_DISABLED

__HAL_RCC_AHB2_IS_CLK_DISABLED

__HAL_RCC_AHB4_IS_CLK_DISABLED

__HAL_RCC_APB1_IS_CLK_DISABLED

__HAL_RCC_APB2_IS_CLK_DISABLED

__HAL_RCC_APB3_IS_CLK_DISABLED

AHB Clock Source

RCC_SYSCLK_DIV1
: SYSCLK not divided

RCC_SYSCLK_DIV2
: SYSCLK divided by 2

RCC_SYSCLK_DIV4
: SYSCLK divided by 4

RCC_SYSCLK_DIV8
: SYSCLK divided by 8

RCC_SYSCLK_DIV16
: SYSCLK divided by 16

RCC_SYSCLK_DIV64
: SYSCLK divided by 64

RCC_SYSCLK_DIV128
: SYSCLK divided by 128

RCC_SYSCLK_DIV256
: SYSCLK divided by 256

RCC_SYSCLK_DIV512

SYSCLK divided by 512

APB1 APB2 APB3 Clock Source

RCC_HCLK_DIV1

HCLK not divided

RCC_HCLK_DIV2

HCLK divided by 2

RCC_HCLK_DIV4

HCLK divided by 4

RCC_HCLK_DIV8

HCLK divided by 8

RCC_HCLK_DIV16

HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_TIM12_CLK_ENABLE

__HAL_RCC_TIM13_CLK_ENABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_I3C1_CLK_ENABLE

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_USART6_CLK_ENABLE

__HAL_RCC_USART10_CLK_ENABLE

__HAL_RCC_USART11_CLK_ENABLE

__HAL_RCC_CEC_CLK_ENABLE

__HAL_RCC_UART7_CLK_ENABLE

__HAL_RCC_UART8_CLK_ENABLE

__HAL_RCC_UART9_CLK_ENABLE

__HAL_RCC_UART12_CLK_ENABLE

__HAL_RCC_DTS_CLK_ENABLE

__HAL_RCC_LPTIM2_CLK_ENABLE

__HAL_RCC_FDCAN_CLK_ENABLE

__HAL_RCC_UCPD1_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_TIM12_CLK_DISABLE

__HAL_RCC_TIM13_CLK_DISABLE

__HAL_RCC_TIM14_CLK_DISABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_I3C1_CLK_DISABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_USART6_CLK_DISABLE

__HAL_RCC_USART10_CLK_DISABLE

__HAL_RCC_USART11_CLK_DISABLE

__HAL_RCC_CEC_CLK_DISABLE

__HAL_RCC_UART7_CLK_DISABLE

__HAL_RCC_UART8_CLK_DISABLE

__HAL_RCC_UART9_CLK_DISABLE

__HAL_RCC_UART12_CLK_DISABLE

__HAL_RCC_DTS_CLK_DISABLE

__HAL_RCC_LPTIM2_CLK_DISABLE

__HAL_RCC_FDCAN_CLK_DISABLE

__HAL_RCC_UCPD1_CLK_DISABLE

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_TIM12_CLK_SLEEP_ENABLE

__HAL_RCC_TIM13_CLK_SLEEP_ENABLE

__HAL_RCC_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_I3C1_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_USART10_CLK_SLEEP_ENABLE

__HAL_RCC_USART11_CLK_SLEEP_ENABLE

__HAL_RCC_CEC_CLK_SLEEP_ENABLE

__HAL_RCC_UART7_CLK_SLEEP_ENABLE

__HAL_RCC_UART8_CLK_SLEEP_ENABLE

__HAL_RCC_UART9_CLK_SLEEP_ENABLE

__HAL_RCC_UART12_CLK_SLEEP_ENABLE

__HAL_RCC_DTS_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_UCPD1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_TIM12_CLK_SLEEP_DISABLE

__HAL_RCC_TIM13_CLK_SLEEP_DISABLE

__HAL_RCC_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_I3C1_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_USART10_CLK_SLEEP_DISABLE

__HAL_RCC_USART11_CLK_SLEEP_DISABLE

__HAL_RCC_CEC_CLK_SLEEP_DISABLE

__HAL_RCC_UART7_CLK_SLEEP_DISABLE

__HAL_RCC_UART8_CLK_SLEEP_DISABLE

__HAL_RCC_UART9_CLK_SLEEP_DISABLE

__HAL_RCC_UART12_CLK_SLEEP_DISABLE

__HAL_RCC_DTS_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_UCPD1_CLK_SLEEP_DISABLE

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_TIM12_FORCE_RESET

__HAL_RCC_TIM13_FORCE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_I3C1_FORCE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_USART6_FORCE_RESET

__HAL_RCC_USART10_FORCE_RESET

__HAL_RCC_USART11_FORCE_RESET

__HAL_RCC_CEC_FORCE_RESET

__HAL_RCC_UART7_FORCE_RESET

__HAL_RCC_UART8_FORCE_RESET

__HAL_RCC_UART9_FORCE_RESET

__HAL_RCC_UART12_FORCE_RESET

__HAL_RCC_DTS_FORCE_RESET

__HAL_RCC_LPTIM2_FORCE_RESET

__HAL_RCC_FDCAN_FORCE_RESET

__HAL_RCC_UCPD1_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_TIM12_RELEASE_RESET

__HAL_RCC_TIM13_RELEASE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_I3C1_RELEASE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_USART6_RELEASE_RESET

__HAL_RCC_USART10_RELEASE_RESET

__HAL_RCC_USART11_RELEASE_RESET

__HAL_RCC_CEC_RELEASE_RESET

__HAL_RCC_UART7_RELEASE_RESET

__HAL_RCC_UART8_RELEASE_RESET

__HAL_RCC_UART9_RELEASE_RESET

__HAL_RCC_UART12_RELEASE_RESET

__HAL_RCC_DTS_RELEASE_RESET

__HAL_RCC_LPTIM2_RELEASE_RESET

__HAL_RCC_FDCAN_RELEASE_RESET

__HAL_RCC_UCPD1_RELEASE_RESET

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_TIM12_IS_CLK_ENABLED

__HAL_RCC_TIM13_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I3C1_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_USART6_IS_CLK_ENABLED

__HAL_RCC_USART10_IS_CLK_ENABLED

__HAL_RCC_USART11_IS_CLK_ENABLED

__HAL_RCC_CEC_IS_CLK_ENABLED

__HAL_RCC_UART7_IS_CLK_ENABLED

__HAL_RCC_UART8_IS_CLK_ENABLED

__HAL_RCC_UART9_IS_CLK_ENABLED

__HAL_RCC_UART12_IS_CLK_ENABLED

__HAL_RCC_DTS_IS_CLK_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_ENABLED

__HAL_RCC_FDCAN_IS_CLK_ENABLED

__HAL_RCC_UCPD1_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_TIM12_IS_CLK_DISABLED

__HAL_RCC_TIM13_IS_CLK_DISABLED

__HAL_RCC_TIM14_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_I3C1_IS_CLK_DISABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_USART6_IS_CLK_DISABLED

__HAL_RCC_USART10_IS_CLK_DISABLED

__HAL_RCC_USART11_IS_CLK_DISABLED

__HAL_RCC_CEC_IS_CLK_DISABLED

__HAL_RCC_UART7_IS_CLK_DISABLED

__HAL_RCC_UART8_IS_CLK_DISABLED

__HAL_RCC_UART9_IS_CLK_DISABLED

__HAL_RCC_UART12_IS_CLK_DISABLED

__HAL_RCC_DTS_IS_CLK_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_DISABLED

__HAL_RCC_FDCAN_IS_CLK_DISABLED

__HAL_RCC_UCPD1_IS_CLK_DISABLED

APB2 Peripheral Clock Enable Disable

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_SPI4_CLK_ENABLE

__HAL_RCC_SPI6_CLK_ENABLE

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_SAI2_CLK_ENABLE

__HAL_RCC_USB_CLK_ENABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_SPI4_CLK_DISABLE

__HAL_RCC_SPI6_CLK_DISABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_SAI2_CLK_DISABLE

__HAL_RCC_USB_CLK_DISABLE

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_SPI6_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_USB_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_SPI6_CLK_SLEEP_DISABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_USB_CLK_SLEEP_DISABLE

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_SPI4_FORCE_RESET

__HAL_RCC_SPI6_FORCE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_SAI2_FORCE_RESET

__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_SPI4_RELEASE_RESET

__HAL_RCC_SPI6_RELEASE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_SAI2_RELEASE_RESET

__HAL_RCC_USB_RELEASE_RESET

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_SPI4_IS_CLK_ENABLED

__HAL_RCC_SPI6_IS_CLK_ENABLED

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_SAI2_IS_CLK_ENABLED

__HAL_RCC_USB_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_SPI4_IS_CLK_DISABLED

__HAL_RCC_SPI6_IS_CLK_DISABLED

__HAL_RCC_SAI1_IS_CLK_DISABLED

__HAL_RCC_SAI2_IS_CLK_DISABLED

__HAL_RCC_USB_IS_CLK_DISABLED

APB3 Peripheral Clock Enable Disable

__HAL_RCC_SBS_CLK_ENABLE

__HAL_RCC_SPI5_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_LPTIM3_CLK_ENABLE

__HAL_RCC_LPTIM4_CLK_ENABLE

__HAL_RCC_LPTIM5_CLK_ENABLE

__HAL_RCC_LPTIM6_CLK_ENABLE

__HAL_RCC_VREF_CLK_ENABLE

__HAL_RCC_RTC_CLK_ENABLE

__HAL_RCC_SBS_CLK_DISABLE

__HAL_RCC_SPI5_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_LPTIM3_CLK_DISABLE

__HAL_RCC_LPTIM4_CLK_DISABLE

__HAL_RCC_LPTIM5_CLK_DISABLE

__HAL_RCC_LPTIM6_CLK_DISABLE

__HAL_RCC_VREF_CLK_DISABLE

__HAL_RCC_RTC_CLK_DISABLE

APB3 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SBS_CLK_SLEEP_ENABLE

__HAL_RCC_SPI5_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM6_CLK_SLEEP_ENABLE

__HAL_RCC_VREF_CLK_SLEEP_ENABLE

__HAL_RCC_RTC_CLK_SLEEP_ENABLE

__HAL_RCC_SBS_CLK_SLEEP_DISABLE

__HAL_RCC_SPI5_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM6_CLK_SLEEP_DISABLE

__HAL_RCC_VREF_CLK_SLEEP_DISABLE

__HAL_RCC_RTC_CLK_SLEEP_DISABLE

APB3 Peripheral Force Release Reset

__HAL_RCC_APB3_FORCE_RESET

__HAL_RCC_SPI5_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_LPTIM3_FORCE_RESET

__HAL_RCC_LPTIM4_FORCE_RESET

__HAL_RCC_LPTIM5_FORCE_RESET

__HAL_RCC_LPTIM6_FORCE_RESET

__HAL_RCC_VREF_FORCE_RESET

__HAL_RCC_APB3_RELEASE_RESET

__HAL_RCC_SPI5_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_LPTIM3_RELEASE_RESET

__HAL_RCC_LPTIM4_RELEASE_RESET

__HAL_RCC_LPTIM5_RELEASE_RESET

__HAL_RCC_LPTIM6_RELEASE_RESET

__HAL_RCC_VREF_RELEASE_RESET

APB3 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SBS_IS_CLK_ENABLED

__HAL_RCC_SPI5_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_ENABLED

__HAL_RCC_LPTIM5_IS_CLK_ENABLED

__HAL_RCC_LPTIM6_IS_CLK_ENABLED

__HAL_RCC_VREF_IS_CLK_ENABLED

__HAL_RCC_RTC_IS_CLK_ENABLED

__HAL_RCC_SBS_IS_CLK_DISABLED

__HAL_RCC_SPI5_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_I2C4_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

__HAL_RCC_LPTIM3_IS_CLK_DISABLED

__HAL_RCC_LPTIM4_IS_CLK_DISABLED

__HAL_RCC_LPTIM5_IS_CLK_DISABLED

__HAL_RCC_LPTIM6_IS_CLK_DISABLED

__HAL_RCC_VREF_IS_CLK_DISABLED

__HAL_RCC_RTC_IS_CLK_DISABLED

RCC attributes

RCC_NSEC_PRIV

Non-secure Privilege attribute item

RCC_NSEC_NPRIV

Non-secure Non-privilege attribute item

RCC_SEC_PRIV

Secure Privilege attribute item

RCC_SEC_NPRIV

Secure Non-privilege attribute item

RCC Backup Domain Reset

__HAL_RCC_BACKUPRESET_FORCE

Description:

- Macros to force or release the Backup domain reset.

Return value:

- None

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

CSI Config

RCC_CSI_OFF

CSI clock deactivation

RCC_CSI_ON

CSI clock activation

RCC_CSICALIBRATION_DEFAULT

Default CSI calibration trimming value

RCC Exported Macros

__HAL_RCC_HSI_DIVIDER_CONFIG

Description:

- Macro to configure the Internal High Speed oscillator (HSI).

Parameters:

- __HSIDIV__: specifies the HSI division factor. This parameter can be one of the following values:
  - RCC_HSI_DIV1 Divide the HSI oscillator clock by 1 (default after reset)
  - RCC_HSI_DIV2 Divide the HSI oscillator clock by 2
  - RCC_HSI_DIV4 Divide the HSI oscillator clock by 4
  - RCC_HSI_DIV8 Divide the HSI oscillator clock by 8

__HAL_RCC_GET_HSI_DIVIDER

Description:

- Macro to get the HSI divider.

Return value:

- The: HSI divider. The returned value can be one of the following:
  - RCC_HSI_DIV1 HSI oscillator divided by 1
  - RCC_HSI_DIV2 HSI oscillator divided by 2
  - RCC_HSI_DIV4 HSI oscillator divided by 4
  - RCC_HSI_DIV8 HSI oscillator divided by 8

__HAL_RCC_HSI_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 64MHz oscillator (HSI).

Return value:

- None

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the HSE Clock Security System HSECSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 64MHz oscillator (HSI) calibration value.

Parameters:

- __HSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x7F.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USARTs, LPUART and I2Cs.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI startup time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_CSI_ENABLE

Description:

- Macros to enable or disable the Internal Low-power oscillator (CSI).

Return value:

- None

Notes:

- The CSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the HSE Clock Security System HSECSS is enabled and CSI is selected as system clock after wake up from system stop). CSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the CSI. After enabling the CSI, the application software should wait on CSIRDY flag to be set indicating that CSI clock is stable and can be used as system clock source. When the CSI is stopped, CSIRDY flag goes low after 6 CSI oscillator clock cycles.

__HAL_RCC_CSI_DISABLE

__HAL_RCC_CSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro Adjusts the Internal oscillator (CSI) calibration value.

Parameters:

- __CSICalibrationValue__: specifies the calibration trimming value. This parameter must be a number between 0 and 0x3F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal CSI RC.

__HAL_RCC_CSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Low-power Internal oscillator (CSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the CSI ON in STOP mode allows to avoid slowing down the communication speed because of the CSI start-up time. The enable of this function has not effect on the CSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_CSISTOP_DISABLE

__HAL_RCC_LSI_ENABLE

Description:

- Macros to enable or disable the Internal Low Speed oscillator (LSI).

Return value:

- None

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF Turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON Turn ON the HSE oscillator.
  - RCC_HSE_BYPASS HSE oscillator bypassed with external clock.
  - RCC_HSE_BYPASS_DIGITAL HSE oscillator bypassed with digital external clock.

Return value:

- None

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLLs and/or system clock. HSE state can not be changed if it is used directly or through the PLL1 as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes.

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF Turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON Turn ON the LSE oscillator.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock.
  - RCC_LSE_BYPASS_DIGITAL LSE oscillator bypassed with external digital clock.

Return value:

- None

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

__HAL_RCC_HSI48_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 48MHz oscillator (HSI48).

Return value:

- None

Notes:

- The HSI48 is stopped by hardware when entering STOP and STANDBY modes. After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSI48_DISABLE

__HAL_RCC_RTC_CLKPRESCALER

Description:

- Macros to configure the RTC clock (RTCCLK).

Parameters:

- __RTCCLKSource__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIVx HSE clock divided by x selected as RTC clock, where x can be between 2 and 63

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using __HAL_RCC_BackupReset_RELEASE() macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_RTC_CONFIG

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: returned value can be one of the following:
  - RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIVx HSE clock divided by x selected as RTC clock, where x can be between 2 and 63 (x can be retrieved with __HAL_RCC_GET_RTC_HSE_PRESCALER())

__HAL_RCC_GET_RTC_HSE_PRESCALER

Description:

- Macro to get the HSE division factor for RTC clock.

Return value:

- The: HSE division factor for RTC clock. The returned value can be one of the following:
  - RCC_RTC_HSE_NOCLOCK : No HSE Clock selected as RTC clock
  - RCC_RTC_HSE_DIV2 : HSE Divided by 2 selected as RTC clock
  - RCC_RTC_HSE_DIV3 : HSE Divided by 3 selected as RTC clock
  - RCC_RTC_HSE_DIV4 : HSE Divided by 4 selected as RTC clock
  - RCC_RTC_HSE_DIV5 : HSE Divided by 5 selected as RTC clock
  - RCC_RTC_HSE_DIV6 : HSE Divided by 6 selected as RTC clock
  - RCC_RTC_HSE_DIV7 : HSE Divided by 7 selected as RTC clock
  - RCC_RTC_HSE_DIV8 : HSE Divided by 8 selected as RTC clock
  - RCC_RTC_HSE_DIV9 : HSE Divided by 9 selected as RTC clock
  - RCC_RTC_HSE_DIV10 : HSE Divided by 10 selected as RTC clock
  - RCC_RTC_HSE_DIV11 : HSE Divided by 11 selected as RTC clock
  - RCC_RTC_HSE_DIV12 : HSE Divided by 12 selected as RTC clock
  - RCC_RTC_HSE_DIV13 : HSE Divided by 13 selected as RTC clock
  - RCC_RTC_HSE_DIV14 : HSE Divided by 14 selected as RTC clock
  - RCC_RTC_HSE_DIV15 : HSE Divided by 15 selected as RTC clock
  - RCC_RTC_HSE_DIV16 : HSE Divided by 16 selected as RTC clock
  - RCC_RTC_HSE_DIV17 : HSE Divided by 17 selected as RTC clock
  - RCC_RTC_HSE_DIV18 : HSE Divided by 18 selected as RTC clock
  - RCC_RTC_HSE_DIV19 : HSE Divided by 19 selected as RTC clock
  - RCC_RTC_HSE_DIV20 : HSE Divided by 20 selected as RTC clock
  - RCC_RTC_HSE_DIV21 : HSE Divided by 21 selected as RTC clock
  - RCC_RTC_HSE_DIV22 : HSE Divided by 22 selected as RTC clock
  - RCC_RTC_HSE_DIV23 : HSE Divided by 23 selected as RTC clock
  - RCC_RTC_HSE_DIV24 : HSE Divided by 24 selected as RTC clock
  - RCC_RTC_HSE_DIV25 : HSE Divided by 25 selected as RTC clock
  - RCC_RTC_HSE_DIV26 : HSE Divided by 26 selected as RTC clock
  - RCC_RTC_HSE_DIV27 : HSE Divided by 27 selected as RTC clock
  - RCC_RTC_HSE_DIV28 : HSE Divided by 28 selected as RTC clock
  - RCC_RTC_HSE_DIV29 : HSE Divided by 29 selected as RTC clock
  - RCC_RTC_HSE_DIV30 : HSE Divided by 30 selected as RTC clock
  - RCC_RTC_HSE_DIV31 : HSE Divided by 31 selected as RTC clock
  - RCC_RTC_HSE_DIV32 : HSE Divided by 32 selected as RTC clock
  - RCC_RTC_HSE_DIV33 : HSE Divided by 33 selected as RTC clock
  - RCC_RTC_HSE_DIV34 : HSE Divided by 34 selected as RTC clock
  - RCC_RTC_HSE_DIV35 : HSE Divided by 35 selected as RTC clock
  - RCC_RTC_HSE_DIV36 : HSE Divided by 36 selected as RTC clock
  - RCC_RTC_HSE_DIV37 : HSE Divided by 37 selected as RTC clock
  - RCC_RTC_HSE_DIV38 : HSE Divided by 38 selected as RTC clock
  - RCC_RTC_HSE_DIV39 : HSE Divided by 39 selected as RTC clock
  - RCC_RTC_HSE_DIV40 : HSE Divided by 40 selected as RTC clock
  - RCC_RTC_HSE_DIV41 : HSE Divided by 41 selected as RTC clock
  - RCC_RTC_HSE_DIV42 : HSE Divided by 42 selected as RTC clock
  - RCC_RTC_HSE_DIV43 : HSE Divided by 43 selected as RTC clock
  - RCC_RTC_HSE_DIV44 : HSE Divided by 44 selected as RTC clock
  - RCC_RTC_HSE_DIV45 : HSE Divided by 45 selected as RTC clock
  - RCC_RTC_HSE_DIV46 : HSE Divided by 46 selected as RTC clock
  - RCC_RTC_HSE_DIV47 : HSE Divided by 47 selected as RTC clock
  - RCC_RTC_HSE_DIV48 : HSE Divided by 48 selected as RTC clock

- RCC_RTC_HSE_DIV49 : HSE Divided by 49 selected as RTC clock
- RCC_RTC_HSE_DIV50 : HSE Divided by 50 selected as RTC clock
- RCC_RTC_HSE_DIV51 : HSE Divided by 51 selected as RTC clock
- RCC_RTC_HSE_DIV52 : HSE Divided by 52 selected as RTC clock
- RCC_RTC_HSE_DIV53 : HSE Divided by 53 selected as RTC clock
- RCC_RTC_HSE_DIV54 : HSE Divided by 54 selected as RTC clock
- RCC_RTC_HSE_DIV55 : HSE Divided by 55 selected as RTC clock
- RCC_RTC_HSE_DIV56 : HSE Divided by 56 selected as RTC clock
- RCC_RTC_HSE_DIV57 : HSE Divided by 57 selected as RTC clock
- RCC_RTC_HSE_DIV58 : HSE Divided by 58 selected as RTC clock
- RCC_RTC_HSE_DIV59 : HSE Divided by 59 selected as RTC clock
- RCC_RTC_HSE_DIV60 : HSE Divided by 60 selected as RTC clock
- RCC_RTC_HSE_DIV61 : HSE Divided by 61 selected as RTC clock
- RCC_RTC_HSE_DIV62 : HSE Divided by 62 selected as RTC clock
- RCC_RTC_HSE_DIV63 : HSE Divided by 63 selected as RTC clock

__HAL_RCC_PLL1_ENABLE

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL1_DISABLE

__HAL_RCC_PLL1_CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL1P_CLK, PLL1Q_CLK, PLL1R_CLK)

Parameters:

- __PLL1_CLOCKOUT__: specifies the PLL clock to be outputted This parameter can be one of the following values:
  - RCC_PLL1_DIVP: This Clock is used to generate the high speed system clock (up to 250MHz)
  - RCC_PLL1_DIVQ: This Clock is used to generate the clock for USB (48 MHz), RNG (<=48 MHz), OCTOSPI, SPI, SAI and Ethernet
  - RCC_PLL1_DIVR: This Clock is used to generate an accurate clock

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL1, (except the ck_pll_p of the System PLL that cannot be stopped if used as System Clock. This is mainly used to save Power.

__HAL_RCC_PLL1_CLKOUT_DISABLE

__HAL_RCC_GET_PLL1_CLKOUT_CONFIG

Description:

- Macro to get the PLL clock output enable status.

Parameters:

- __PLL1_CLOCKOUT__: specifies the PLL1 clock to be output. This parameter can be one of the following values:
  - RCC_PLL1_DIVP: This Clock is used to generate the high speed system clock (up to 250MHz)
  - RCC_PLL1_DIVQ: This Clock is used to generate the clock for USB (48 MHz), RNG (<=48 MHz), OCTOSPI, SPI, SAI and Ethernet
  - RCC_PLL1_DIVR: This Clock is used to generate an accurate clock

Return value:

- SET: / RESET

__HAL_RCC_PLL1_FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL1 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL1

__HAL_RCC_PLL1_FRACN_DISABLE

__HAL_RCC_PLL1_CONFIG

Description:

- Macro to configures the main PLL (PLL1) clock source, multiplication and division factors.

Parameters:

- __PLL1SOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLL1_SOURCE_CSI: CSI oscillator clock selected as PLL1 clock entry
  - RCC_PLL1_SOURCE_HSI: HSI oscillator clock selected as PLL1 clock entry
  - RCC_PLL1_SOURCE_HSE: HSE oscillator clock selected as PLL1 clock entry
- __PLL1M__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLL1N__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 4 and 512.
- __PLL1P__: specifies the division factor for system clock. This parameter must be a number between 2 and 128 (where odd numbers not allowed)
- __PLL1Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL1R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when the main PLL1 is disabled.
- This clock source (__PLL1SOURCE__) is the clock source for PLL1 (main PLL) and is different from PLL2 & PLL3 clock sources.
- You have to set the PLL1M parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLL1N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ (when in wide VCO range)

__HAL_RCC_PLL1_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL1 clock source.

Parameters:

- __PLL1SOURCE__: specifies the PLL1 entry clock source. This parameter can be one of the following values:
  - RCC_PLL1_SOURCE_CSI: CSI oscillator clock selected as PLL1 clock entry
  - RCC_PLL1_SOURCE_HSI: HSI oscillator clock selected as PLL1 clock entry
  - RCC_PLL1_SOURCE_HSE: HSE oscillator clock selected as PLL1 clock entry

Notes:

- This function must be used only when PLL1 is disabled.

__HAL_RCC_PLL1_DIVM_CONFIG

Description:

- Macro to configure the PLL1 input clock division factor M.

Parameters:

- __PLL1M__: specifies the division factor for PLL1 clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63. In order to save power when PLL1 is not used, the value of PLL1M must be set to 0.

Return value:

- None

Notes:

- This function must be used only when the PLL1 is disabled. PLL1 clock source is common with the main PLL (configured through __HAL_RCC_PLL1_CONFIG() macro)

__HAL_RCC_PLL1_FRACN_CONFIG

Description:

- Macro to configures the main PLL clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLL1FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL1 VCO
- Warning: The software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL1VCOSEL = 0 150 to 420 MHz if PLL1VCOSEL = 1.

__HAL_RCC_PLL1_VCIRANGE

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- __PLL1VCIRange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL1_VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL1_VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL1_VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL1_VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL1_VCORANGE

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- __RCC_PLL1VCORange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL1_VCORANGE_WIDE: Range frequency is between 192 and 836 MHz
  - RCC_PLL1_VCORANGE_MEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_GET_PLL1_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL1 clock source.

Return value:

- The: oscillator used as PLL1 clock source. The returned value can be one of the following:
  - RCC_PLL1_SOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLL1_SOURCE_CSI: CSI oscillator is used as PLL clock source.
  - RCC_PLL1_SOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLL1_SOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_CSI: CSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_PLL1CLK: PLL1P output is used as system clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_CSI: CSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_PLL1CLK: PLL1P used as system clock.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - RCC_LSEDRIVE_LOW LSE oscillator low drive capability.
  - RCC_LSEDRIVE_MEDIUMLOW LSE oscillator medium low drive capability.
  - RCC_LSEDRIVE_MEDIUMHIGH LSE oscillator medium high drive capability.
  - RCC_LSEDRIVE_HIGH LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable the write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset).

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the wake up from stop clock.

Parameters:

- __STOPWUCLK__: specifies the clock source used after wake up from stop. This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_HSI HSI selected as system clock source
  - RCC_STOP_WAKEUPCLOCK_CSI CSI selected as system clock source

Return value:

- None

Notes:

- The configured clock is also used as emergency clock for the Clock Security System on HSE (HSECSS).

__HAL_RCC_HSECSS_RECOVCLK_CONFIG

__HAL_RCC_KERWAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the Kernel wake up from stop clock.

Parameters:

- __RCC_STOPKERWUCLK__: specifies the Kernel clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI selected as Kernel clock source
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI selected as Kernel clock source

Return value:

- None

Flags

RCC_FLAG_CSIRDY

CSI Ready flag

RCC_FLAG_HSIRDY

HSI Ready flag

RCC_FLAG_HSIDIVF

HSI divider flag

RCC_FLAG_HSERDY

HSE Ready flag

RCC_FLAG_PLL1RDY

PLL1 Ready flag

RCC_FLAG_PLL2RDY

PLL2 Ready flag

RCC_FLAG_PLL3RDY

PLL3 Ready flag

RCC_FLAG_HSI48RDY

HSI48 Ready flag

RCC_FLAG_LSERDY

LSE Ready flag

RCC_FLAG_LSECSSD

LSE Clock Security System Interrupt flag

RCC_FLAG_LSIRDY

LSI Ready flag

RCC_FLAG_RMVF

Remove reset flag

RCC_FLAG_PINRST

PIN reset flag

RCC_FLAG_BORRST

BOR reset flag

RCC_FLAG_SFTRST

Software Reset flag

RCC_FLAG_IWDGRST

Independent Watchdog reset flag

RCC_FLAG_WWDGRST

Window watchdog reset flag

RCC_FLAG_LPWRRST

Low-Power reset flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt (Perform access to RCC_CIER[8:0] bits to enable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_CSIRDY CSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt (*)
  - RCC_IT_HSECSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt (Perform access to RCC_CIER[8:0] bits to disable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_CSIRDY CSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt (*)
  - RCC_IT_HSECSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CICR[10:0] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_CSIRDY CSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt (*)
  - RCC_IT_HSECSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_IT

Description:

- Check whether the RCC interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_CSIRDY CSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLL1RDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt (*)
  - RCC_IT_HSECSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

Description:

- Set RMVF bit to clear the reset flags.

Return value:

- None

__HAL_RCC_GET_FLAG

Description:

- Check whether the selected RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_CSIRDY CSI oscillator clock ready
  - RCC_FLAG_HSIRDY HSI oscillator clock ready
  - RCC_FLAG_HSERDY HSE oscillator clock ready
  - RCC_FLAG_PLL1RDY Main PLL1 clock ready
  - RCC_FLAG_PLL2RDY PLL2 clock ready
  - RCC_FLAG_PLL3RDY PLL3 clock ready (*)
  - RCC_FLAG_HSI48RDY HSI48 clock ready
  - RCC_FLAG_LSERDY LSE oscillator clock ready
  - RCC_FLAG_LSECSSD Clock security system failure on LSE oscillator detection
  - RCC_FLAG_LSIRDY LSI oscillator clock ready
  - RCC_FLAG_HSIDIVF HSI Divider
  - RCC_FLAG_BORRST BOR reset
  - RCC_FLAG_PINRST Pin reset
  - RCC_FLAG_RMVF Remove reset Flag
  - RCC_FLAG_SFTRST Software reset
  - RCC_FLAG_IWDGRST Independent Watchdog reset
  - RCC_FLAG_WWDGRST Window Watchdog reset
  - RCC_FLAG_LPWRRST Low Power reset

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

RTC HSE Prescaler

RCC_RTC_HSE_NOCLOCK

RCC_RTC_HSE_DIV2

RCC_RTC_HSE_DIV3

RCC_RTC_HSE_DIV4

RCC_RTC_HSE_DIV5

RCC_RTC_HSE_DIV6

RCC_RTC_HSE_DIV7

RCC_RTC_HSE_DIV8

RCC_RTC_HSE_DIV9

RCC_RTC_HSE_DIV10

RCC_RTC_HSE_DIV11

RCC_RTC_HSE_DIV12

RCC_RTC_HSE_DIV13

RCC_RTC_HSE_DIV14

RCC_RTC_HSE_DIV15

RCC_RTC_HSE_DIV16

RCC_RTC_HSE_DIV17

RCC_RTC_HSE_DIV18

RCC_RTC_HSE_DIV19

RCC_RTC_HSE_DIV20

RCC_RTC_HSE_DIV21

RCC_RTC_HSE_DIV22

RCC_RTC_HSE_DIV23

RCC_RTC_HSE_DIV24

RCC_RTC_HSE_DIV25

RCC_RTC_HSE_DIV26

RCC_RTC_HSE_DIV27

RCC_RTC_HSE_DIV28

RCC_RTC_HSE_DIV29

RCC_RTC_HSE_DIV30

RCC_RTC_HSE_DIV31

RCC_RTC_HSE_DIV32

RCC_RTC_HSE_DIV33

RCC_RTC_HSE_DIV34

RCC_RTC_HSE_DIV35

RCC_RTC_HSE_DIV36

RCC_RTC_HSE_DIV37

RCC_RTC_HSE_DIV38

RCC_RTC_HSE_DIV39

RCC_RTC_HSE_DIV40

RCC_RTC_HSE_DIV41

RCC_RTC_HSE_DIV42

RCC_RTC_HSE_DIV43

RCC_RTC_HSE_DIV44

RCC_RTC_HSE_DIV45

RCC_RTC_HSE_DIV46

RCC_RTC_HSE_DIV47

RCC_RTC_HSE_DIV48

RCC_RTC_HSE_DIV49

RCC_RTC_HSE_DIV50

RCC_RTC_HSE_DIV51

RCC_RTC_HSE_DIV52

RCC_RTC_HSE_DIV53

RCC_RTC_HSE_DIV54

RCC_RTC_HSE_DIV55

RCC_RTC_HSE_DIV56

RCC_RTC_HSE_DIV57

RCC_RTC_HSE_DIV58

RCC_RTC_HSE_DIV59

RCC_RTC_HSE_DIV60

RCC_RTC_HSE_DIV61

RCC_RTC_HSE_DIV62

RCC_RTC_HSE_DIV63

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

External Analog clock source for HSE clock

RCC_HSE_BYPASS_DIGITAL

External Digital clock source for HSE clock

HSI48 Config

RCC_HSI48_OFF

HSI48 clock deactivation

RCC_HSI48_ON

HSI48 clock activation

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_DEFAULT

HSI Div

RCC_HSI_DIV1

HSI clock is not divided

RCC_HSI_DIV2

HSI clock is divided by 2

RCC_HSI_DIV4

HSI clock is divided by 4

RCC_HSI_DIV8

HSI clock is divided by 8

Interrupts

RCC_IT_LSIRDY

LSI Ready Interrupt flag

RCC_IT_LSERDY

LSE Ready Interrupt flag

RCC_IT_CSIRDY

CSI Ready Interrupt flag

RCC_IT_HSIRDY

HSI16 Ready Interrupt flag

RCC_IT_HSERDY

HSE Ready Interrupt flag

RCC_IT_HSI48RDY

HSI48 Ready Interrupt flag

RCC_IT_PLL1RDY

PLL1 Ready Interrupt flag

RCC_IT_PLL2RDY

PLL2 Ready Interrupt flag

RCC_IT_PLL3RDY

PLL3 Ready Interrupt flag

RCC_IT_HSECSS

HSE Clock Security System Interrupt flag

RCC items

RCC_HSI

RCC_HSE

RCC_CSI

RCC_LSI

RCC_LSE

RCC_SYSCLK

RCC_PRESC

RCC_PLL1

RCC_PLL2

RCC_PLL3

RCC_HSI48

RCC_RMVF

RCC_CKPERSEL

RCC_ALL

LSE Drive Config

RCC_LSEDRIVE_LOW

LSE low drive capability

RCC_LSEDRIVE_MEDIUMLOW

LSE medium low drive capability

RCC_LSEDRIVE_MEDIUMHIGH

LSE medium high drive capability

RCC_LSEDRIVE_HIGH

LSE high drive capability

LSE Config

RCC_LSE_OFF

LSE clock deactivation

RCC_LSE_ON

LSE clock activation

RCC_LSE_BYPASS

External Analog clock source for LSE clock

RCC_LSE_BYPASS_DIGITAL

External Digital clock source for LSE clock

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

RCC MCO1 Clock Source

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLL1Q

RCC_MCO1SOURCE_HSI48

RCC MCO2 Clock Source

RCC_MCO2SOURCE_SYSCLK

RCC_MCO2SOURCE_PLL2P

RCC_MCO2SOURCE_HSE

RCC_MCO2SOURCE_PLL1P

RCC_MCO2SOURCE_CSI

RCC_MCO2SOURCE_LSI

MCOx Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_3

RCC_MCODIV_4

RCC_MCODIV_5

RCC_MCODIV_6

RCC_MCODIV_7

RCC_MCODIV_8

RCC_MCODIV_9

RCC_MCODIV_10

RCC_MCODIV_11

RCC_MCODIV_12

RCC_MCODIV_13

RCC_MCODIV_14

RCC_MCODIV_15

MCO Index

RCC_MCO1

RCC_MCO2

Oscillator Type

RCC_OSCILLATORTYPE_NONE

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_HSE

HSE to configure

RCC_OSCILLATORTYPE_HSI

HSI to configure

RCC_OSCILLATORTYPE_LSE

LSE to configure

RCC_OSCILLATORTYPE_LSI

LSI to configure

RCC_OSCILLATORTYPE_CSI

CSI to configure

RCC_OSCILLATORTYPE_HSI48

HSI48 to configure

RCC PLL1 Clock Output

RCC_PLL1_DIVP

RCC_PLL1_DIVQ

RCC_PLL1_DIVR

RCC PLL1 Clock Source

RCC_PLL1_SOURCE_NONE

RCC_PLL1_SOURCE_HSI

RCC_PLL1_SOURCE_CSI

RCC_PLL1_SOURCE_HSE

RCC PLL1 Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

RCC PLL1 VCI Range

RCC_PLL1_VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL1_VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL1_VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL1_VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL1 VCO Range

RCC_PLL1_VCORANGE_WIDE

Clock range frequency between 192 and 836 MHz

RCC_PLL1_VCORANGE_MEDIUM

Clock range frequency between 150 and 420 MHz

RCC PLL2 Clock Output

RCC_PLL2_DIVP

RCC_PLL2_DIVQ

RCC_PLL2_DIVR

RCC PLL2 Clock Source

RCC_PLL2_SOURCE_NONE

RCC_PLL2_SOURCE_HSI

RCC_PLL2_SOURCE_CSI

RCC_PLL2_SOURCE_HSE

RCC PLL2 VCI Range

RCC_PLL2_VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL2_VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL2_VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL2_VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL2 VCO Range

RCC_PLL2_VCORANGE_WIDE

Wide VCO range: 192 to 836 MHz

RCC_PLL2_VCORANGE_MEDIUM

Medium VCO range: 150 to 420 MHz

RCC PLL3 Clock Output

RCC_PLL3_DIVP

RCC_PLL3_DIVQ

RCC_PLL3_DIVR

RCC PLL3 Clock Source

RCC_PLL3_SOURCE_NONE

RCC_PLL3_SOURCE_HSI

RCC_PLL3_SOURCE_CSI

RCC_PLL3_SOURCE_HSE

RCC PLL3 VCI Range

RCC_PLL3_VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL3_VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL3_VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL3_VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL3 VCO Range

RCC_PLL3_VCORANGE_WIDE

Wide VCO range: 192 to 836 MHz

RCC_PLL3_VCORANGE_MEDIUM

Medium VCO range: 150 to 420 MHz

Reset Flag

RCC_RESET_FLAG_PIN

PIN reset flag

RCC_RESET_FLAG_PWR

BOR or POR/PDR reset flag

RCC_RESET_FLAG_SW

Software Reset flag

RCC_RESET_FLAG_IWDG

Independent Watchdog reset flag

RCC_RESET_FLAG_WWDG

Window watchdog reset flag

RCC_RESET_FLAG_LPWR

Low power reset flag

RCC_RESET_FLAG_ALL

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE

Description:

- Macros to enable or disable the RTC clock.

Return value:

- None

Notes:

- As the RTC is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the RTC (to be done once after reset). These macros must be used after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK

No clock used as RTC clock source

RCC_RTCCLKSOURCE_LSE

LSE oscillator clock used as RTC clock source

RCC_RTCCLKSOURCE_LSI

LSI oscillator clock used as RTC clock source

RCC_RTCCLKSOURCE_HSE_DIVx

HSE oscillator clock divided by X used as RTC clock source

RCC_RTCCLKSOURCE_HSE_DIV2

RCC_RTCCLKSOURCE_HSE_DIV3

RCC_RTCCLKSOURCE_HSE_DIV4

RCC_RTCCLKSOURCE_HSE_DIV5

RCC_RTCCLKSOURCE_HSE_DIV6

RCC_RTCCLKSOURCE_HSE_DIV7

RCC_RTCCLKSOURCE_HSE_DIV8

RCC_RTCCLKSOURCE_HSE_DIV9

RCC_RTCCLKSOURCE_HSE_DIV10

RCC_RTCCLKSOURCE_HSE_DIV11

RCC_RTCCLKSOURCE_HSE_DIV12

RCC_RTCCLKSOURCE_HSE_DIV13

RCC_RTCCLKSOURCE_HSE_DIV14

RCC_RTCCLKSOURCE_HSE_DIV15

RCC_RTCCLKSOURCE_HSE_DIV16

RCC_RTCCLKSOURCE_HSE_DIV17

RCC_RTCCLKSOURCE_HSE_DIV18

RCC_RTCCLKSOURCE_HSE_DIV19

RCC_RTCCLKSOURCE_HSE_DIV20

RCC_RTCCLKSOURCE_HSE_DIV21

RCC_RTCCLKSOURCE_HSE_DIV22

RCC_RTCCLKSOURCE_HSE_DIV23

RCC_RTCCLKSOURCE_HSE_DIV24

RCC_RTCCLKSOURCE_HSE_DIV25

RCC_RTCCLKSOURCE_HSE_DIV26

RCC_RTCCLKSOURCE_HSE_DIV27

RCC_RTCCLKSOURCE_HSE_DIV28

RCC_RTCCLKSOURCE_HSE_DIV29

RCC_RTCCLKSOURCE_HSE_DIV30

RCC_RTCCLKSOURCE_HSE_DIV31

RCC_RTCCLKSOURCE_HSE_DIV32

RCC_RTCCLKSOURCE_HSE_DIV33

RCC_RTCCLKSOURCE_HSE_DIV34

RCC_RTCCLKSOURCE_HSE_DIV35

RCC_RTCCLKSOURCE_HSE_DIV36

RCC_RTCCLKSOURCE_HSE_DIV37

RCC_RTCCLKSOURCE_HSE_DIV38

RCC_RTCCLKSOURCE_HSE_DIV39

RCC_RTCCLKSOURCE_HSE_DIV40

RCC_RTCCLKSOURCE_HSE_DIV41

RCC_RTCCLKSOURCE_HSE_DIV42

RCC_RTCCLKSOURCE_HSE_DIV43

RCC_RTCCLKSOURCE_HSE_DIV44

RCC_RTCCLKSOURCE_HSE_DIV45

RCC_RTCCLKSOURCE_HSE_DIV46

RCC_RTCCLKSOURCE_HSE_DIV47

RCC_RTCCLKSOURCE_HSE_DIV48

RCC_RTCCLKSOURCE_HSE_DIV49

RCC_RTCCLKSOURCE_HSE_DIV50

RCC_RTCCLKSOURCE_HSE_DIV51

RCC_RTCCLKSOURCE_HSE_DIV52

RCC_RTCCLKSOURCE_HSE_DIV53

RCC_RTCCLKSOURCE_HSE_DIV54

RCC_RTCCLKSOURCE_HSE_DIV55

RCC_RTCCLKSOURCE_HSE_DIV56

RCC_RTCCLKSOURCE_HSE_DIV57

RCC_RTCCLKSOURCE_HSE_DIV58

RCC_RTCCLKSOURCE_HSE_DIV59

RCC_RTCCLKSOURCE_HSE_DIV60

RCC_RTCCLKSOURCE_HSE_DIV61

RCC_RTCCLKSOURCE_HSE_DIV62

RCC_RTCCLKSOURCE_HSE_DIV63

RCC Stop KernelWakeUpClock

RCC_STOP_KERWAKEUPCLOCK_HSI

HSI kernel clock selection after wake-up from STOP

RCC_STOP_KERWAKEUPCLOCK_CSI

CSI kernel clock selection after wake-up from STOP

Wake-Up from STOP Clock

RCC_STOP_WAKEUPCLOCK_HSI

HSI selection after wake-up from STOP

RCC_STOP_WAKEUPCLOCK_CSI

CSI selection after wake-up from STOP

System Clock Source

RCC_SYSCLKSOURCE_HSI

HSI selection as system clock

RCC_SYSCLKSOURCE_CSI

CSI selection as system clock

RCC_SYSCLKSOURCE_HSE

HSE selection as system clock

RCC_SYSCLKSOURCE_PLLCLK

PLL1 selection as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system clock

RCC_SYSCLKSOURCE_STATUS_CSI

CSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system clock

RCC_SYSCLKSOURCE_STATUS_PLLCLK

PLL1 used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK

SYSCLK to configure

RCC_CLOCKTYPE_HCLK

HCLK to configure

RCC_CLOCKTYPE_PCLK1

PCLK1 to configure

RCC_CLOCKTYPE_PCLK2

PCLK2 to configure

RCC_CLOCKTYPE_PCLK3

PCLK3 to configure

Timeout Values

RCC_LSI_TIMEOUT_VALUE

RCC_HSI48_TIMEOUT_VALUE

RCC_PLL_TIMEOUT_VALUE

RCC_CLOCKSWITCH_TIMEOUT_VALUE

RCC_PLL_FRAC_WAIT_VALUE

RCC_HSE_TIMEOUT_VALUE

RCC_HSI_TIMEOUT_VALUE

RCC_CSI_TIMEOUT_VALUE

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

60 HAL RCC Extension Driver

60.1 RCCEx Firmware driver registers structures

60.1.1 RCC_PLL2InitTypeDef

RCC_PLL2InitTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2Source***
- ***uint32_t PLL2M***
- ***uint32_t PLL2N***
- ***uint32_t PLL2P***
- ***uint32_t PLL2Q***
- ***uint32_t PLL2R***
- ***uint32_t PLL2RGE***
- ***uint32_t PLL2VCOSEL***
- ***uint32_t PLL2FRACN***
- ***uint32_t PLL2ClockOut***

Field Documentation

- ***uint32_t RCC_PLL2InitTypeDef::PLL2Source***
  RCC_PLL2Source: PLL2 entry clock source. This parameter must be a value of ***RCC_PLL2_Clock_Source***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2M***
  PLL2M: Division factor for PLL2 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL2InitTypeDef::PLL2N***
  PLL2N: Multiplication factor for PLL2 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLL2InitTypeDef::PLL2P***
  PLL2P: Division factor for peripheral clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2Q***
  PLL2Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2R***
  PLL2R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLL2InitTypeDef::PLL2RGE***
  PLL2RGE: PLL2 clock Input range This parameter must be a value of ***RCC_PLL2_VCI_Range***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2VCOSEL***
  PLL2VCOSEL: PLL2 clock Output range This parameter must be a value of ***RCC_PLL2_VCO_Range***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2FRACN***
  PLL2FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL2 VCO It should be a value between 0 and 8191
- ***uint32_t RCC_PLL2InitTypeDef::PLL2ClockOut***
  PLL2ClockOut: specifies PLL2 output clock to be enabled. This parameter must be a value or a combination of ***RCC_PLL2_Clock_Output***

60.1.2 RCC_PLL3InitTypeDef

RCC_PLL3InitTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL3Source***
- ***uint32_t PLL3M***

- ***uint32_t PLL3N***
- ***uint32_t PLL3P***
- ***uint32_t PLL3Q***
- ***uint32_t PLL3R***
- ***uint32_t PLL3RGE***
- ***uint32_t PLL3VCOSEL***
- ***uint32_t PLL3FRACN***
- ***uint32_t PLL3ClockOut***

Field Documentation

- ***uint32_t RCC_PLL3InitTypeDef::PLL3Source***
  RCC_PLL3Source: PLL3 entry clock source. This parameter must be a value of ***RCC_PLL3_Clock_Source***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3M***
  PLL3M: Division factor for PLL3 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL3InitTypeDef::PLL3N***
  PLL3N: Multiplication factor for PLL3 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLL3InitTypeDef::PLL3P***
  PLL3P: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3Q***
  PLL3Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3R***
  PLL3R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLL3InitTypeDef::PLL3RGE***
  PLL3RGE: PLL3 clock Input range This parameter must be a value of ***RCC_PLL3_VCI_Range***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3VCOSEL***
  PLL3VCOSEL: PLL3 clock Output range This parameter must be a value of ***RCC_PLL3_VCO_Range***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3FRACN***
  PLL3FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191
- ***uint32_t RCC_PLL3InitTypeDef::PLL3ClockOut***
  PLL3ClockOut: specifies PLL3 output clock to be enabled. This parameter must be a value or a combination of ***RCC_PLL3_Clock_Output***

60.1.3 PLL1_ClocksTypeDef

PLL1_ClocksTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL1_P_Frequency***
- ***uint32_t PLL1_Q_Frequency***
- ***uint32_t PLL1_R_Frequency***

Field Documentation

- ***uint32_t PLL1_ClocksTypeDef::PLL1_P_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_Q_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_R_Frequency***

60.1.4 PLL2_ClocksTypeDef

PLL2_ClocksTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2_P_Frequency***

- *uint32_t PLL2_Q_Frequency*
- *uint32_t PLL2_R_Frequency*

Field Documentation

- *uint32_t PLL2_ClocksTypeDef::PLL2_P_Frequency*
- *uint32_t PLL2_ClocksTypeDef::PLL2_Q_Frequency*
- *uint32_t PLL2_ClocksTypeDef::PLL2_R_Frequency*

60.1.5 PLL3_ClocksTypeDef

PLL3_ClocksTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- *uint32_t PLL3_P_Frequency*
- *uint32_t PLL3_Q_Frequency*
- *uint32_t PLL3_R_Frequency*

Field Documentation

- *uint32_t PLL3_ClocksTypeDef::PLL3_P_Frequency*
- *uint32_t PLL3_ClocksTypeDef::PLL3_Q_Frequency*
- *uint32_t PLL3_ClocksTypeDef::PLL3_R_Frequency*

60.1.6 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- *uint64_t PeriphClockSelection*
- *RCC_PLL2InitTypeDef PLL2*
- *RCC_PLL3InitTypeDef PLL3*
- *uint32_t CkperClockSelection*
- *uint32_t Usart1ClockSelection*
- *uint32_t Usart2ClockSelection*
- *uint32_t Usart3ClockSelection*
- *uint32_t Uart4ClockSelection*
- *uint32_t Uart5ClockSelection*
- *uint32_t Usart6ClockSelection*
- *uint32_t Uart7ClockSelection*
- *uint32_t Uart8ClockSelection*
- *uint32_t Uart9ClockSelection*
- *uint32_t Usart10ClockSelection*
- *uint32_t Usart11ClockSelection*
- *uint32_t Uart12ClockSelection*
- *uint32_t Lpuart1ClockSelection*
- *uint32_t I2c1ClockSelection*
- *uint32_t I2c2ClockSelection*
- *uint32_t I2c3ClockSelection*
- *uint32_t I2c4ClockSelection*
- *uint32_t I3c1ClockSelection*
- *uint32_t Lptim1ClockSelection*
- *uint32_t Lptim2ClockSelection*
- *uint32_t Lptim3ClockSelection*
- *uint32_t Lptim4ClockSelection*
- *uint32_t Lptim5ClockSelection*
- *uint32_t Lptim6ClockSelection*
- *uint32_t FdcanClockSelection*

- ***uint32_t Sai1ClockSelection***
- ***uint32_t Sai2ClockSelection***
- ***uint32_t RngClockSelection***
- ***uint32_t Sdmmc1ClockSelection***
- ***uint32_t Sdmmc2ClockSelection***
- ***uint32_t AdcDacClockSelection***
- ***uint32_t DacLowPowerClockSelection***
- ***uint32_t OspiClockSelection***
- ***uint32_t Spi1ClockSelection***
- ***uint32_t Spi2ClockSelection***
- ***uint32_t Spi3ClockSelection***
- ***uint32_t Spi4ClockSelection***
- ***uint32_t Spi5ClockSelection***
- ***uint32_t Spi6ClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t CecClockSelection***
- ***uint32_t UsbClockSelection***
- ***uint32_t TimPresSelection***

Field Documentation

- ***uint64_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***RCC_PLL2InitTypeDef RCC_PeriphCLKInitTypeDef::PLL2***
  PLL2structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***RCC_PLL3InitTypeDef RCC_PeriphCLKInitTypeDef::PLL3***
  PLL3 structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***uint32_t RCC_PeriphCLKInitTypeDef::CkperClockSelection***
  Specifies CKPER clock source This parameter can be a value of ***RCCEx_CLKP_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  Specifies USART2 clock source. This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
  Specifies USART3 clock source. This parameter can be a value of ***RCCEx_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
  Specifies UART4 clock source. This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
  Specifies UART5 clock source. This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart6ClockSelection***
  Specifies USART6 clock source. This parameter can be a value of ***RCCEx_USART6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart7ClockSelection***
  Specifies UART7 clock source. This parameter can be a value of ***RCCEx_UART7_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart8ClockSelection***
  Specifies UART8 clock source. This parameter can be a value of ***RCCEx_UART8_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart9ClockSelection***
  Specifies UART9 clock source. This parameter can be a value of ***RCCEx_UART9_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart10ClockSelection***
  Specifies USART10 clock source. This parameter can be a value of ***RCCEx_USART10_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart11ClockSelection***
  Specifies USART11 clock source. This parameter can be a value of ***RCCEx_USART11_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart12ClockSelection***
  Specifies UART12 clock source. This parameter can be a value of ***RCCEx_UART12_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  Specifies I2C1 clock source. This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
  Specifies I2C2 clock source. This parameter can be a value of ***RCCEx_I2C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  Specifies I2C3 clock source. This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
  Specifies I2C4 clock source. This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I3c1ClockSelection***
  Specifies I3C1 clock source. This parameter can be a value of ***RCCEx_I3C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
  Specifies LPTIM1 clock source. This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection***
  Specifies LPTIM2 clock source. This parameter can be a value of ***RCCEx_LPTIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim3ClockSelection***
  Specifies LPTIM3 clock source. This parameter can be a value of ***RCCEx_LPTIM3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim4ClockSelection***
  Specifies LPTIM4 clock source. This parameter can be a value of ***RCCEx_LPTIM4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim5ClockSelection***
  Specifies LPTIM5 clock source. This parameter can be a value of ***RCCEx_LPTIM5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim6ClockSelection***
  Specifies LPTIM6 clock source. This parameter can be a value of ***RCCEx_LPTIM6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::FdcanClockSelection***
  Specifies FDCAN kernel clock source. This parameter can be a value of ***RCCEx_FDCAN_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
  Specifies SAI1 clock source. This parameter can be a value of ***RCCEx_SAI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai2ClockSelection***
  Specifies SAI2 clock source. This parameter can be a value of ***RCCEx_SAI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection***
  Specifies RNG clock source. This parameter can be a value of ***RCCEx_RNG_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc1ClockSelection***
  Specifies SDMMC1 clock source. This parameter can be a value of ***RCCEx_SDMMC1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc2ClockSelection***
  Specifies SDMMC2 clock source. This parameter can be a value of ***RCCEx_SDMMC2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcDacClockSelection***
  Specifies ADC1, ADC2 and DAC interface clock source. This parameter can be a value of ***RCCEx_ADCDAC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::DacLowPowerClockSelection***
  Specifies DAC low-power clock source used in Sample and hold mode. This parameter can be a value of ***RCCEx_DAC_Low_Power_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::OspiClockSelection***
  Specifies OctoSPI clock source. This parameter can be a value of ***RCCEx_OSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi1ClockSelection***
  Specifies SPI1 clock source. This parameter can be a value of ***RCCEx_SPI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi2ClockSelection***
  Specifies SPI2 clock source. This parameter can be a value of ***RCCEx_SPI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi3ClockSelection***
  Specifies SPI3 clock source. This parameter can be a value of ***RCCEx_SPI3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi4ClockSelection***
  Specifies SPI4 clock source. This parameter can be a value of ***RCCEx_SPI4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi5ClockSelection***
  Specifies SPI5 clock source. This parameter can be a value of ***RCCEx_SPI5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi6ClockSelection***
  Specifies SPI6 clock source. This parameter can be a value of ***RCCEx_SPI6_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection***
  Specifies CEC clock source. This parameter can be a value of ***RCCEx_CEC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
  Specifies USB clock source. This parameter can be a value of ***RCCEx_USB_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::TimPresSelection***
  Specifies TIM Clock Prescalers Selection. This parameter can be a value of ***RCCEx_TIM_Prescaler_Selection***

60.1.7 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***
- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .
- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***
  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***
  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x7F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

60.1.8 RCC_CRSSynchroInfoTypeDef

RCC_CRSSynchroInfoTypeDef is defined in the stm32h5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***
  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***
  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F

- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***
  Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***
  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

60.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

60.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: *Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.*

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPLL1ClockFreq()***
- ***HAL_RCCEx_GetPLL2ClockFreq()***
- ***HAL_RCCEx_GetPLL3ClockFreq()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***

60.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation of PLL2, PLL3, LSE CSS, Low speed clock output and clock after wake-up from STOP mode.

This section contains the following APIs:

- ***HAL_RCCEx_EnablePLL2()***
- ***HAL_RCCEx_DisablePLL2()***
- ***HAL_RCCEx_EnablePLL3()***
- ***HAL_RCCEx_DisablePLL3()***
- ***HAL_RCCEx_WakeUpStopCLKConfig()***
- ***HAL_RCCEx_KerWakeUpStopCLKConfig()***
- ***HAL_RCCEx_EnableLSECSS()***
- ***HAL_RCCEx_DisableLSECSS()***
- ***HAL_RCCEx_LSECSS_IRQHandler()***
- ***HAL_RCCEx_LSECSS_Callback()***
- ***HAL_RCCEx_EnableLSCO()***
- ***HAL_RCCEx_DisableLSCO()***

60.2.3 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extension HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled
2. Enable CRS clock in IP MSP init which will use CRS functions

3. Call CRS functions as follows:
   a. Prepare synchronization configuration necessary for HSI48 calibration
      - Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
      - Macro __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and synchronization frequencies values
   b. Call function HAL_RCCEx_CRSConfig which
      - Resets CRS registers to their default values.
      - Configures CRS registers with synchronization configuration
      - Enables automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.
   c. A polling function is provided to wait for complete synchronization
      - Call function HAL_RCCEx_CRSWaitSynchronization()
      - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function HAL_RCCEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (CRS_IRQn/CRS_IRQHandler)
   – Call function HAL_RCCEx_CRSConfig()
   – Enable CRS_IRQn (thanks to NVIC functions)
   – Enable CRS interrupt (__HAL_RCC_CRS_ENABLE_IT)
   – Implement CRS status management in the following user callbacks called from HAL_RCCEx_CRS_IRQHandler():
      - HAL_RCCEx_CRS_SyncOkCallback()
      - HAL_RCCEx_CRS_SyncWarnCallback()
      - HAL_RCCEx_CRS_ExpectedSyncCallback()
      - HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- ***HAL_RCCEx_CRSConfig()***
- ***HAL_RCCEx_CRSSoftwareSynchronizationGenerate()***
- ***HAL_RCCEx_CRSGetSynchronizationInfo()***
- ***HAL_RCCEx_CRSWaitSynchronization()***
- ***HAL_RCCEx_CRS_IRQHandler()***
- ***HAL_RCCEx_CRS_SyncOkCallback()***
- ***HAL_RCCEx_CRS_SyncWarnCallback()***
- ***HAL_RCCEx_CRS_ExpectedSyncCallback()***
- ***HAL_RCCEx_CRS_ErrorCallback()***

60.2.4 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (const RCC_PeriphCLKInitTypeDef * pPeriphClkInit)

Function description

Initialize the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **pPeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains a field PeriphClockSelection which can be a combination of the following values:
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 UART4 peripheral clock (*)
  - RCC_PERIPHCLK_UART5 UART5 peripheral clock (*)
  - RCC_PERIPHCLK_USART6 USART6 peripheral clock (*)
  - RCC_PERIPHCLK_UART7 UART7 peripheral clock (*)
  - RCC_PERIPHCLK_UART8 UART8 peripheral clock (*)
  - RCC_PERIPHCLK_UART9 UART9 peripheral clock (*)
  - RCC_PERIPHCLK_USART10 USART10 peripheral clock (*)
  - RCC_PERIPHCLK_USART11 USART11 peripheral clock (*)
  - RCC_PERIPHCLK_UART12 UART12 peripheral clock (*)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C3 I2C3 peripheral clock (*)
  - RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (*)
  - RCC_PERIPHCLK_I3C1 I3C1 peripheral clock
  - RCC_PERIPHCLK_I3C2 I3C2 peripheral clock (***)
  - RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock
  - RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock
  - RCC_PERIPHCLK_SAI1 SAI1 peripheral clock (*)
  - RCC_PERIPHCLK_SAI2 SAI2 peripheral clock (*)
  - RCC_PERIPHCLK_ADCDAC ADCDAC peripheral clock
  - RCC_PERIPHCLK_ADC ADC peripheral clock
  - RCC_PERIPHCLK_SDMMC1 SDMMC1 peripheral clock (*)
  - RCC_PERIPHCLK_SDMMC2 SDMMC2 peripheral clock (**)
  - RCC_PERIPHCLK_CKPER CKPER peripheral clock
  - RCC_PERIPHCLK_RTC RTC peripheral clock
  - RCC_PERIPHCLK_RNG RNG peripheral clock
  - RCC_PERIPHCLK_SPI1 SPI1 peripheral clock
  - RCC_PERIPHCLK_SPI2 SPI2 peripheral clock
  - RCC_PERIPHCLK_SPI3 SPI3 peripheral clock
  - RCC_PERIPHCLK_SPI4 SPI4 peripheral clock (*)
  - RCC_PERIPHCLK_SPI5 SPI5 peripheral clock (*)
  - RCC_PERIPHCLK_SPI6 SPI6 peripheral clock (*)
  - RCC_PERIPHCLK_OSPI OCTOSPI peripheral clock (*)
  - RCC_PERIPHCLK_FDCAN FDCAN peripheral clock
  - RCC_PERIPHCLK_CEC CEC peripheral clock (*)
  - RCC_PERIPHCLK_USB USB peripheral clock
  - RCC_PERIPHCLK_LPTIM3 LPTIM3 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM4 LPTIM4 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM5 LPTIM5 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM6 LPTIM6 peripheral clock (*)
  - RCC_PERIPHCLK_DAC_LP DAC peripheral low-power clock
  - RCC_PERIPHCLK_TIM TIM peripheral clock

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source: in this case the access to Backup domain is enabled.

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * pPeriphClkInit)

Function description

Get the pPeriphClkInit according to the internal RCC configuration registers.

Parameters

- **pPeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks (ADC12, DAC, SDMMC1, SDMMC2, OCTOSPI1, TIM, LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5, LPTIM6, SPI1, SPI2, SPI3, SPI4, SPI5, SPI6, USART1, USART2, USART3, UART4, UART5, USART6, UART7, UART8, UART9, USART10, USART11, UART12, LPUART1, I2C1, I2C2, I2C3, I2C4, I3C1, I3C2, CEC, FDCAN, SAI1, SAI2, USB,), PLL2 and PLL3.

Return values

- **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint64_t PeriphClk)

Function description

Return the peripheral clock frequency for peripherals.

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 UART4 peripheral clock (*)
  - RCC_PERIPHCLK_UART5 UART5 peripheral clock (*)
  - RCC_PERIPHCLK_USART6 USART6 peripheral clock (*)
  - RCC_PERIPHCLK_UART7 UART7 peripheral clock (*)
  - RCC_PERIPHCLK_UART8 UART8 peripheral clock (*)
  - RCC_PERIPHCLK_UART9 UART9 peripheral clock (*)
  - RCC_PERIPHCLK_USART10 USART10 peripheral clock (*)
  - RCC_PERIPHCLK_USART11 USART11 peripheral clock (*)
  - RCC_PERIPHCLK_UART12 UART12 peripheral clock (*)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C3 I2C3 peripheral clock (*)
  - RCC_PERIPHCLK_I2C4 I2C4 peripheral clock (*)
  - RCC_PERIPHCLK_I3C1 I3C1 peripheral clock
  - RCC_PERIPHCLK_I3C2 I3C2 peripheral clock (***)
  - RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock
  - RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock
  - RCC_PERIPHCLK_SAI1 SAI1 peripheral clock (*)
  - RCC_PERIPHCLK_SAI2 SAI2 peripheral clock (*)
  - RCC_PERIPHCLK_ADCDAC ADCDAC peripheral clock
  - RCC_PERIPHCLK_ADC ADC peripheral clock
  - RCC_PERIPHCLK_SDMMC1 SDMMC1 peripheral clock (*)
  - RCC_PERIPHCLK_SDMMC2 SDMMC2 peripheral clock (**)
  - RCC_PERIPHCLK_CKPER CKPER peripheral clock
  - RCC_PERIPHCLK_RTC RTC peripheral clock
  - RCC_PERIPHCLK_RNG RNG peripheral clock
  - RCC_PERIPHCLK_SPI1 SPI1 peripheral clock
  - RCC_PERIPHCLK_SPI2 SPI2 peripheral clock
  - RCC_PERIPHCLK_SPI3 SPI3 peripheral clock
  - RCC_PERIPHCLK_SPI4 SPI4 peripheral clock (*)
  - RCC_PERIPHCLK_SPI5 SPI5 peripheral clock (*)
  - RCC_PERIPHCLK_SPI6 SPI6 peripheral clock (*)
  - RCC_PERIPHCLK_OSPI OCTOSPI peripheral clock (*)
  - RCC_PERIPHCLK_FDCAN FDCAN peripheral clock
  - RCC_PERIPHCLK_CEC CEC peripheral clock (*)
  - RCC_PERIPHCLK_USB USB peripheral clock
  - RCC_PERIPHCLK_LPTIM3 LPTIM3 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM4 LPTIM4 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM5 LPTIM5 peripheral clock (*)
  - RCC_PERIPHCLK_LPTIM6 LPTIM6 peripheral clock (*)
  - RCC_PERIPHCLK_DAC_LP DAC low-power peripheral clock
  - RCC_PERIPHCLK_TIM TIM peripheral clock

Return values

- **Frequency:** in Hz

Notes

- Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEx_GetPLL1ClockFreq

Function name

void HAL_RCCEx_GetPLL1ClockFreq (PLL1_ClocksTypeDef * pPLL1_Clocks)

Function description

Returns the PLL1 clock frequencies : PLL1_P_Frequency, PLL1_R_Frequency and PLL1_Q_Frequency.

Parameters

- **pPLL1_Clocks:** pointer to PLL1_ClocksTypeDef structure.

Return values

- **None:**

Notes

- The PLL1 clock frequencies computed by this function may not be the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL1CLK changes, this function must be called to update the right PLL1CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL2ClockFreq

Function name

void HAL_RCCEx_GetPLL2ClockFreq (PLL2_ClocksTypeDef * pPLL2_Clocks)

Function description

Returns the PLL2 clock frequencies: PLL2_P_Frequency, PLL2_R_Frequency and PLL2_Q_Frequency.

Parameters

- **pPLL2_Clocks:** pointer to PLL2_ClocksTypeDef structure.

Return values

- **None:**

Notes

- The PLL2 clock frequencies computed by this function may not be the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL2CLK changes, this function must be called to update the right PLL2CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL3ClockFreq

Function name

void HAL_RCCEx_GetPLL3ClockFreq (PLL3_ClocksTypeDef * pPLL3_Clocks)

Function description

Returns the PLL3 clock frequencies: PLL3_P_Frequency, PLL3_R_Frequency and PLL3_Q_Frequency.

Parameters

- **pPLL3_Clocks:** pointer to PLL3_ClocksTypeDef structure.

Return values

- **None:**

Notes

- The PLL3 clock frequencies computed by this function may not be the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL3CLK changes, this function must be called to update the right PLL3CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_EnablePLL2

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLL2 (RCC_PLL2InitTypeDef * pPLL2Init)

Function description

Initialize and Enable the PLL2 according to the specified parameters in the RCC_PLL2InitTypeDef.

Parameters

- **pPLL2Init:** pointer to an RCC_PLL2InitTypeDef structure that contains the configuration information for the PLL2

Return values

- **HAL:** status

HAL_RCCEx_DisablePLL2

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLL2 (void)

Function description

Disable PLL2.

Return values

- **HAL:** status

HAL_RCCEx_EnablePLL3

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLL3 (RCC_PLL3InitTypeDef * pPLL3Init)

Function description

Initialize and Enable the PLL3 according to the specified parameters in the RCC_PLL3InitTypeDef.

Parameters

- **pPLL3Init:** pointer to an RCC_PLL3InitTypeDef structure that contains the configuration information for the PLL3

Return values

- **HAL:** status.

HAL_RCCEx_DisablePLL3

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLL3 (void)

Function description

Disable PLL3.

Return values

- **HAL:** status.

HAL_RCCEx_WakeUpStopCLKConfig

Function name

void HAL_RCCEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator clock source for wakeup from Stop and HSE CSS backup clock.

Parameters

- **WakeUpClk:** Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_HSI HSI oscillator selection
  - RCC_STOP_WAKEUPCLOCK_CSI CSI oscillator selection

Return values

- **None:**

Notes

- This function shall not be called after the Clock Security System on HSE has been enabled.

HAL_RCCEx_KerWakeUpStopCLKConfig

Function name

void HAL_RCCEx_KerWakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator Kernel clock source for wakeup from Stop.

Parameters

- **WakeUpClk:** Kernel Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI oscillator selection
  - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI oscillator selection

Return values

- **None:**

HAL_RCCEx_EnableLSECSS

Function name

void HAL_RCCEx_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().

HAL_RCCEx_DisableLSECSS

Function name

void HAL_RCCEx_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEx_LSECSS_IRQHandler

Function name

void HAL_RCCEx_LSECSS_IRQHandler (void)

Function description

Handle the RCC LSE Clock Security System interrupt request.

Return values

- **None:**

HAL_RCCEx_LSECSS_Callback

Function name

void HAL_RCCEx_LSECSS_Callback (void)

Function description

RCCEx LSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCCEx_EnableLSCO

Function name

void HAL_RCCEx_EnableLSCO (uint32_t LSCOSource)

Function description

Select the Low Speed Microcontroller Clock source to output on LSCO pin (PB2).

Parameters

- **LSCOSource:** specifies the Low Speed clock source to output. This parameter can be one of the following values:
  - RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source
  - RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source

Return values

- **None:**

HAL_RCCEx_DisableLSCO

Function name

void HAL_RCCEx_DisableLSCO (void)

Function description

Disable the Low Speed Microcontroller Clock Output.

Return values

- **None:**

HAL_RCCEx_CRSConfig

Function name

void HAL_RCCEx_CRSConfig (const RCC_CRSInitTypeDef * pInit)

Function description

Start automatic synchronization for polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSInitTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name

void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)

Function description

Generate the software synchronization event.

Return values

- **None:**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name

void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)

Function description

Return synchronization info.

Parameters

- **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSWaitSynchronization

Function name

uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)

Function description

Wait for CRS Synchronization status.

Parameters

- **Timeout:** Duration of the timeout

Return values

- **Combination:** of Synchronization status This parameter can be a combination of the following values:
  - RCC_CRS_TIMEOUT
  - RCC_CRS_SYNCOK
  - RCC_CRS_SYNCWARN
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name

void HAL_RCCEx_CRS_IRQHandler (void)

Function description

Handle the Clock Recovery System interrupt request.

Return values

- **None:**

HAL_RCCEx_CRS_SyncOkCallback

Function name

void HAL_RCCEx_CRS_SyncOkCallback (void)

Function description

RCCEx Clock Recovery System SYNCOK interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_SyncWarnCallback

Function name

void HAL_RCCEx_CRS_SyncWarnCallback (void)

Function description

RCCEx Clock Recovery System SYNCWARN interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name

void HAL_RCCEx_CRS_ExpectedSyncCallback (void)

Function description

RCCEx Clock Recovery System Expected SYNC interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ErrorCallback

Function name

void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)

Function description

RCCEx Clock Recovery System Error interrupt callback.

Parameters

- **Error:** Combination of Error status. This parameter can be a combination of the following values:
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Return values

- **none:**

60.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

60.3.1 RCCEx

RCCEx

ADCDAC Kernel Clock Source

RCC_ADCDACCLKSOURCE_HCLK

RCC_ADCDACCLKSOURCE_SYSCLK

RCC_ADCDACCLKSOURCE_PLL2R

RCC_ADCDACCLKSOURCE_HSE

RCC_ADCDACCLKSOURCE_HSI

RCC_ADCDACCLKSOURCE_CSI

CEC Clock Source

RCC_CECCLKSOURCE_LSE

RCC_CECCLKSOURCE_LSI

RCC_CECCLKSOURCE_CSI_DIV122

RCCEx CLKP Clock Source

RCC_CLKPSOURCE_HSI

RCC_CLKPSOURCE_CSI

RCC_CLKPSOURCE_HSE

RCCEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT

Default Frequency error limit

RCCEx CRS Extended Features

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disable the oscillator clock for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustment of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustment of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK

SYNC event OK flag

RCC_CRS_FLAG_SYNCWARN

SYNC warning flag

RCC_CRS_FLAG_ERR

Error flag

RCC_CRS_FLAG_ESYNC

Expected SYNC flag

RCC_CRS_FLAG_SYNCERR

SYNC error

RCC_CRS_FLAG_SYNCMISS

SYNC missed

RCC_CRS_FLAG_TRIMOVF

Trimming overflow or underflow

RCCEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP

Upcounting direction, the actual frequency is above the target

RCC_CRS_FREQERRORDIR_DOWN

Downcounting direction, the actual frequency is below the target

RCCEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK

SYNC event OK

RCC_CRS_IT_SYNCWARN

SYNC warning

RCC_CRS_IT_ERR

Error

RCC_CRS_IT_ESYNC

Expected SYNC

RCC_CRS_IT_SYNCERR

SYNC error

RCC_CRS_IT_SYNCMISS

SYNC missed

RCC_CRS_IT_TRIMOVF

Trimming overflow or underflow

RCCEx CRS ReloadValueDefault

RCC_CRS_RELOADVALUE_DEFAULT

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS SynchroDivider

RCC_CRS_SYNC_DIV1

Synchro Signal not divided (default)

RCC_CRS_SYNC_DIV2

Synchro Signal divided by 2

RCC_CRS_SYNC_DIV4

Synchro Signal divided by 4

RCC_CRS_SYNC_DIV8

Synchro Signal divided by 8

RCC_CRS_SYNC_DIV16

Synchro Signal divided by 16

RCC_CRS_SYNC_DIV32

Synchro Signal divided by 32

RCC_CRS_SYNC_DIV64

Synchro Signal divided by 64

RCC_CRS_SYNC_DIV128

Synchro Signal divided by 128

RCCEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

RCC_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

RCCEx CRS SynchroSource

RCC_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

RCC_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

RCC_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

DAC Low Power Kernel Clock Source

RCC_DACLPCLKSOURCE_LSE

RCC_DACLPCLKSOURCE_LSI

RCCEx Exported Macros

__HAL_RCC_TIMIC_ENABLE

Description:

- Macros to enable or disable the HSI and CSI divided clock for Timers 12, 15 and LPTimer2 input capture.

Return value:

- None: HSI/1024, CSI/128 and HSI/8 generation for Timers 12,15 and LPTimer2 Input capture

Notes:

- HSI and CSI clocks without division are available when TIMIC is enabled

__HAL_RCC_TIMIC_DISABLE

No clock available for Timers Input capture

__HAL_RCC_PLL2_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL2 clock source.

Parameters:

- __PLL2SOURCE__: specifies the PLL2 entry clock source. This parameter can be one of the following values:
  - RCC_PLL2_SOURCE_NONE: No oscillator clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_CSI: CSI oscillator clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_HSI: HSI oscillator clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_HSE: HSE oscillator clock selected as PLL2 clock entry

Notes:

- This function must be used only when all PLL2 is disabled.

__HAL_RCC_GET_PLL2_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL2 clock source.

Return value:

- The: oscillator used as PLL2 clock source. The returned value can be one of the following:
  - RCC_PLL2_SOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLL2_SOURCE_CSI: CSI oscillator is used as PLL clock source.
  - RCC_PLL2_SOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLL2_SOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_PLL2_CONFIG

Description:

- Macro to configures the PLL2 source, multiplication and division factors.

Parameters:

- __PLL2SOURCE__: specifies the PLL2 entry clock source. This parameter can be one of the following values:
  - RCC_PLL2_SOURCE_NONE No clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_CSI CSI oscillator clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_HSI HSI oscillator clock selected as PLL2 clock entry
  - RCC_PLL2_SOURCE_HSE HSE oscillator clock selected as PLL2 clock entry
- __PLL2M__: specifies the division factor of PLL2 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63.
- __PLL2N__: specifies the multiplication factor for PLL2 VCO output clock This parameter must be a number between 4 and 512.
- __PLL2P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL2Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL2R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL2 is disabled.
- You have to set the PLL2N parameter correctly to ensure that the VCO output frequency is between 192 and 836 MHz (Wide range) or 150 and 420 Mhz (Medium range). PLL2 clock frequency = f(PLL2) multiplied by PLL2N

__HAL_RCC_PLL2_MULN_CONFIG

Description:

- Macro to configure the PLL2 clock multiplication factor N.

Parameters:

- __PLL2N__: specifies the multiplication factor for PLL2 VCO output clock. This parameter must be a number between 4 and 512.

Return value:

- None

Notes:

- This function must be used only when the PLL2 is disabled. PLL2 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL2_CONFIG() macro.
- You have to set the PLL2N parameter correctly to ensure that the VCO output frequency is between 192 and 836 MHz (Wide range) or 150 and 420 Mhz (Medium range). PLL2 clock frequency = f(PLL2) multiplied by PLL2N

__HAL_RCC_PLL2_DIVM_CONFIG

Description:

- Macro to configure the PLL2 input clock division factor M.

Parameters:

- __PLL2M__: specifies the division factor for PLL2 clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63. In order to save power when PLL2 is not used, the value of PLL2M must be set to 0.

Return value:

- None

Notes:

- This function must be used only when the PLL2 is disabled. PLL2 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL2_CONFIG() macro.

__HAL_RCC_PLL2_DIVP_CONFIG

Description:

- Macro to configure the PLL2 clock division factor P.

Parameters:

- __PLL2P__: specifies the division factor for PLL2 output P clock. This parameter must be a number in the range (1 to 128). Use to set PLL2 output P clock frequency = f(PLL2) / PLL2P

Return value:

- None

Notes:

- This function must be used only when the PLL2 is disabled. PLL2 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL2_CONFIG() macro.

__HAL_RCC_PLL2_DIVQ_CONFIG

Description:

- Macro to configure the PLL2 clock division factor Q.

Parameters:

- __PLL2Q__: specifies the division factor for PLL2 output Q clock. This parameter must be a number in the range (1 to 128). Use to set PLL2 output Q clock frequency = f(PLL2) / PLL2Q

Return value:

- None

Notes:

- This function must be used only when the PLL2 is disabled. PLL2 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL2_CONFIG() macro.

__HAL_RCC_PLL2_DIVR_CONFIG

Description:

- Macro to configure the PLL2 clock division factor R.

Parameters:

- __PLL2R__: specifies the division factor for PLL2 output R clock. This parameter must be a number in the range (1 to 128). Use to set PLL2 output R clock frequency = f(PLL2) / PLL2R

Return value:

- None

Notes:

- This function must be used only when the PLL2 is disabled. PLL2 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL2_CONFIG() macro.

__HAL_RCC_PLL2_ENABLE

Description:

- Macros to enable or disable the PLL2.

Return value:

- None

Notes:

- After enabling PLL2, the application software should wait on PLL2RDY flag to be set indicating that PLL2 clock is stable and can be used as kernel clock source. The PLL2 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL2_DISABLE

__HAL_RCC_PLL2_CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL2_P_CLK, PLL2_Q_CLK, PLL2_R_CLK)

Parameters:

- __PLL2_CLOCKOUT__: specifies the PLL2 clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLL2_DIVP: This clock is used to generate an accurate kernel clock to achieve high-quality audio performance on SAI interface, SPI/I2S and LPTIM peripherals.
  - RCC_PLL2_DIVQ: This clock is used to generate kernel clock for the random number generator RNG (<=48 MHz), SPI, FDCAN and UART/USART peripherals.
  - RCC_PLL2_DIVR: This clock is used to generate kernel clock for ADC and DAC peripherals.

Return value:

- None

Notes:

- Enabling/disabling those clocks can be done at any time without the need to stop the PLL2, This is mainly used to save Power.

__HAL_RCC_PLL2_CLKOUT_DISABLE

__HAL_RCC_GET_PLL2_CLKOUT_CONFIG

Description:

- Macro to get the PLL2 clock output enable status.

Parameters:

- __PLL2_CLOCKOUT__: specifies the PLL2 clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLL2_DIVP: This clock is used to generate an accurate kernel clock to achieve high-quality audio performance on SAI interface, SPI/I2S and LPTIM peripherals.
  - RCC_PLL2_DIVQ: This clock is used to generate kernel clock for the random number generator RNG (<=48 MHz), SPI, FDCAN and UART/USART peripherals.
  - RCC_PLL2_DIVR: This clock is used to generate kernel clock for ADC and DAC peripherals.

Return value:

- SET: / RESET

__HAL_RCC_PLL2_FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL2 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be done at any time without the need to stop the PLL2

__HAL_RCC_PLL2_FRACN_DISABLE

__HAL_RCC_PLL2_FRACN_CONFIG

Description:

- Macro to configures PLL2 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLL2FRACN__: Specifies Fractional Part Of The Multiplication factor for PLL2 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL2 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL2VCOSEL = 0 150 to 420 MHz if PLL2VCOSEL = 1.

__HAL_RCC_PLL2_VCIRANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __PLL2VCIRange__: specifies the PLL2 input frequency range This parameter can be one of the following values:
  - RCC_PLL2_VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL2_VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL2_VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL2_VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL2_VCORANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __RCC_PLL2VCORange__: specifies the PLL2 output frequency range This parameter can be one of the following values:
  - RCC_PLL2_VCORANGE_WIDE: Range frequency is between 192 and 836 MHz
  - RCC_PLL2_VCORANGE_MEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_PLL3_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL3 clock source.

Parameters:

- __PLL3SOURCE__: specifies the PLL3 entry clock source. This parameter can be one of the following values:
  - RCC_PLL3_SOURCE_NONE: No oscillator clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_CSI: CSI oscillator clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_HSI: HSI oscillator clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_HSE: HSE oscillator clock selected as PLL3 clock entry

Notes:

- This function must be used only when all PLL3 is disabled.

__HAL_RCC_GET_PLL3_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL3 clock source.

Return value:

- The: oscillator used as PLL3 clock source. The returned value can be one of the following:
  - RCC_PLL3_SOURCE_NONE: No oscillator is used as PLL3 clock source.
  - RCC_PLL3_SOURCE_CSI: CSI oscillator is used as PLL3 clock source.
  - RCC_PLL3_SOURCE_HSI: HSI oscillator is used as PLL3 clock source.
  - RCC_PLL3_SOURCE_HSE: HSE oscillator is used as PLL3 clock source.

__HAL_RCC_PLL3_CONFIG

Description:

- Macro to configures the PLL3 source, multiplication and division factors.

Parameters:

- __PLL3SOURCE__: specifies the PLL3 entry clock source. This parameter can be one of the following values:
  - RCC_PLL3_SOURCE_NONE No clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_CSI CSI oscillator clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_HSI HSI oscillator clock selected as PLL3 clock entry
  - RCC_PLL3_SOURCE_HSE HSE oscillator clock selected as PLL3 clock entry
- __PLL3M__: specifies the division factor of PLL3 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63.
- __PLL3N__: specifies the multiplication factor for PLL3 VCO output clock This parameter must be a number between 4 and 512.
- __PLL3P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL3Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL3R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL3 is disabled.
- You have to set the PLL3N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ (when in wide VCO range)

__HAL_RCC_PLL3_MULN_CONFIG

Description:

- Macro to configure the PLL3 clock multiplication factor N.

Parameters:

- __PLL3N__: specifies the multiplication factor for PLL3 VCO output clock. This parameter must be a number between 8 and 86.

Return value:

- None

Notes:

- This function must be used only when the PLL3 is disabled. PLL3 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL3_CONFIG() macro.
- You have to set the PLL3N parameter correctly to ensure that the VCO output frequency is between 192 and 836 MHz (Wide range) or 150 and 420 Mhz (Medium range). PLL3 clock frequency = f(PLL3) multiplied by PLL3N

__HAL_RCC_PLL3_DIVM_CONFIG

Description:

- Macro to configure the PLL3 input clock division factor M.

Parameters:

- __PLL3M__: specifies the division factor for PLL3 clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63. In order to save power when PLL3 is not used, the value of PLL3M must be set to 0.

Return value:

- None

Notes:

- This function must be used only when the PLL3 is disabled. PLL3 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL3_CONFIG() macro.

__HAL_RCC_PLL3_DIVP_CONFIG

Description:

- Macro to configure the PLL3 clock division factor P.

Parameters:

- __PLL3P__: specifies the division factor for PLL3 output P clock. This parameter must be a number in the range (1 to 128). Use to set PLL3 output P clock frequency = f(PLL3) / PLL3P

Return value:

- None

Notes:

- This function must be used only when the PLL3 is disabled. PLL3 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL3_CONFIG() macro.

__HAL_RCC_PLL3_DIVQ_CONFIG

Description:

- Macro to configure the PLL3 clock division factor Q.

Parameters:

- __PLL3Q__: specifies the division factor for PLL3 output Q clock. This parameter must be a number in the range (1 to 128). Use to set PLL3 output Q clock frequency = f(PLL3) / PLL3Q

Return value:

- None

Notes:

- This function must be used only when the PLL3 is disabled. PLL3 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL3_CONFIG() macro.

__HAL_RCC_PLL3_DIVR_CONFIG

Description:

- Macro to configure the PLL3 clock division factor R.

Parameters:

- __PLL3R__: specifies the division factor for PLL3 output R clock. This parameter must be a number in the range (1 to 128). Use to set PLL3 output R clock frequency = f(PLL3) / PLL3R

Return value:

- None

Notes:

- This function must be used only when the PLL3 is disabled. PLL3 clock source is independent from the main PLL and is configured through __HAL_RCC_PLL3_CONFIG() macro.

__HAL_RCC_PLL3_FRACN_CONFIG

Description:

- Macro to configures PLL3 clock Fractional Part of The Multiplication Factor.

Parameters:

- __PLL3FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL3 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL3VCOSEL = 0 150 to 420 MHz if PLL3VCOSEL = 1.

__HAL_RCC_PLL3_VCIRANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- __PLL3VCIRange__: specifies the PLL3 input frequency range This parameter can be one of the following values:
  - RCC_PLL3_VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL3_VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL3_VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL3_VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL3_VCORANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- __RCC_PLL3VCORange__: specifies the PLL3 input frequency range This parameter can be one of the following values:
  - RCC_PLL3VCOWIDE: Range frequency is between 192 and 836 MHz
  - RCC_PLL3VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_PLL3_ENABLE

Description:

- Macros to enable or disable the PLL3.

Return value:

- NoneMacros: to enable or disable the main PLL3.

Notes:

- The PLL3 is disabled by hardware when entering STOP and STANDBY modes.
- After enabling PLL3, the application software should wait on PLL3RDY flag to be set indicating that PLL3 clock is stable and can be used as kernel clock source. PLL3 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL3_DISABLE

__HAL_RCC_PLL3_FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL3 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be done at any time without the need to stop the PLL3

__HAL_RCC_PLL3_FRACN_DISABLE

__HAL_RCC_PLL3_CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL3_P_CLK, PLL3_Q_CLK, PLL3_R_CLK)

Parameters:

- __PLL3_CLOCKOUT__: specifies the PLL3 clock to be outputted This parameter can be one of the following values:
  - RCC_PLL3_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI and SPI/I2S interfaces.
  - RCC_PLL3_DIVQ: This clock is used to generate kernel clock for SPI, LPUART, UART/USART and USB peripherals.
  - RCC_PLL3_DIVR: This clock is used to generate kernel clock for I2C, I3C and LPTIM peripherals.

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL3, This is mainly used to save Power.

__HAL_RCC_PLL3_CLKOUT_DISABLE

__HAL_RCC_GET_PLL3_CLKOUT_CONFIG

Description:

- Macro to get clock output enable status (PLL3_SAI2).

Parameters:

- __PLL3_CLOCKOUT__: specifies the PLL3 clock to be output. This parameter can be one or a combination of the following values:
  - RCC_PLL3_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI and SPI/I2S interfaces.
  - RCC_PLL3_DIVQ: This clock is used to generate kernel clock for SPI, LPUART, UART/USART and USB peripherals.
  - RCC_PLL3_DIVR: This clock is used to generate kernel clock for I2C, I3C and LPTIM peripherals.

Return value:

- SET: / RESET

__HAL_RCC_ADCDAC_CONFIG

Description:

- Macro to configure the ADC and DAC kernel clock source.

Parameters:

- __ADCDAC_CLKSOURCE__: specifies the ADC and DAC kernel clock source. This parameter can be one of the following values:
  - RCC_ADCDACCLKSOURCE_HCLK AHB bus clock selected as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_SYSCLK System clock selected as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_PLL2R PLL2R clock selected as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_HSE HSE clock selected as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_HSI HSI clock selected as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_CSI CSI clock selected as ADC and DAC kernel clock

Return value:

- None

__HAL_RCC_GET_ADCDAC_SOURCE

Description:

- Macro to get the ADC and DAC kernel clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCDACCLKSOURCE_HCLK AHB Bus clock used as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_SYSCLK System clock used as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_PLL2R PLL2R clock used as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_HSE HSE oscillator used as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_HSI HSI oscillator used as ADC and DAC kernel clock
  - RCC_ADCDACCLKSOURCE_CSI CSI oscillator used as ADC and DAC kernel clock

__HAL_RCC_DAC_LP_CONFIG

Description:

- Macro to configure the DAC kernel clock source in low-power mode.

Parameters:

- __DACLPCLKSOURCE__: specifies the DAC kernel clock source in low-power mode. This parameter can be one of the following values:
  - RCC_DACLPCLKSOURCE_LSE LSE oscillator selected as DAC kernel clock in low-power mode
  - RCC_DACLPCLKSOURCE_LSI LSI oscillator selected as DAC kernel clock in low-power mode

Return value:

- None

__HAL_RCC_GET_DAC_LP_SOURCE

Description:

- Macro to get the DAC kernel clock source in low-power mode.

Return value:

- The: clock source can be one of the following values:
  - RCC_DACLPCLKSOURCE_LSE LSE oscillator used as DAC kernel clock in low-power mode
  - RCC_DACLPCLKSOURCE_LSI LSI oscillator used as DAC kernel clock in low-power mode

__HAL_RCC_FDCAN_CONFIG

Description:

- Macro to configure the FDCAN kernel clock (FDCANCLK).

Parameters:

- __FDCAN_CLKSOURCE__: specifies the FDCAN kernel clock source. This parameter can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE HSE oscillator selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_PLL1Q PLL1Q Clock selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_PLL2Q PLL2Q Clock selected as FDCAN kernel clock

Return value:

- None

__HAL_RCC_GET_FDCAN_SOURCE

Description:

- Macro to get the FDCAN clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE HSE oscillator selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_PLL1Q PLL1Q Clock selected as FDCAN kernel clock
  - RCC_FDCANCLKSOURCE_PLL2Q PLL2Q Clock selected as FDCAN kernel clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK3 PCLK3 selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL2P PLL2P selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL3R PLL3R selected as LPTIM1 clock (*)
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP CLKP selected as LPTIM1 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_PCLK3 PCLK3 selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL2P PLL2P selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL3R PLL3R selected as LPTIM1 clock (*)
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP CLKP selected as LPTIM1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- Macro to configure the LPTIM2 clock (LPTIM2CLK).

Parameters:

- __LPTIM2_CLKSOURCE__: specifies the LPTIM2 clock source. This parameter can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL2P PLL2P selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL3R PLL3R selected as LPTIM2 clock (*)
  - RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI LSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP CLKP selected as LPTIM2 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- Macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL2P PLL2P selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL3R PLL3R selected as LPTIM2 clock (*)
  - RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI LSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP CLKP selected as LPTIM2 clock

__HAL_RCC_LPTIM3_CONFIG

Description:

- Macro to configure the LPTIM3 clock (LPTIM3CLK).

Parameters:

- __LPTIM3_CLKSOURCE__: specifies the LPTIM3 clock source. This parameter can be one of the following values:
  - RCC_LPTIM3CLKSOURCE_PCLK3 PCLK3 selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL2P PLL2P selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL3R PLL3R selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE LSE selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI LSI selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP CLKP selected as LPTIM3 clock

Return value:

- None

__HAL_RCC_GET_LPTIM3_SOURCE

Description:

- Macro to get the LPTIM3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM3CLKSOURCE_PCLK3 PCLK3 selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL2P PLL2P selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL3R PLL3R selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE LSE selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI LSI selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP CLKP selected as LPTIM3 clock

__HAL_RCC_LPTIM4_CONFIG

Description:

- Macro to configure the LPTIM4 clock (LPTIM4CLK).

Parameters:

- __LPTIM4_CLKSOURCE__: specifies the LPTIM4 clock source. This parameter can be one of the following values:
  - RCC_LPTIM4CLKSOURCE_PCLK3 PCLK3 selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL2P PLL2P selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL3R PLL3R selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE LSE selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI LSI selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP CLKP selected as LPTIM4 clock

Return value:

- None

__HAL_RCC_GET_LPTIM4_SOURCE

Description:

- Macro to get the LPTIM4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM4CLKSOURCE_PCLK3 PCLK3 selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL2P PLL2P selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL3R PLL3R selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE LSE selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI LSI selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP CLKP selected as LPTIM4 clock

__HAL_RCC_LPTIM5_CONFIG

Description:

- Macro to configure the LPTIM5 clock (LPTIM5CLK).

Parameters:

- __LPTIM5_CLKSOURCE__: specifies the LPTIM5 clock source. This parameter can be one of the following values:
  - RCC_LPTIM5CLKSOURCE_PCLK3 PCLK3 selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL2P PLL2P selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL3R PLL3R selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE LSE selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI LSI selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP CLKP selected as LPTIM5 clock

Return value:

- None

__HAL_RCC_GET_LPTIM5_SOURCE

Description:

- Macro to get the LPTIM5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM5CLKSOURCE_PCLK3 PCLK3 selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL2P PLL2P selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL3R PLL3R selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE LSE selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI LSI selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP CLKP selected as LPTIM5 clock

__HAL_RCC_LPTIM6_CONFIG

Description:

- Macro to configure the LPTIM6 clock (LPTIM6CLK).

Parameters:

- __LPTIM6_CLKSOURCE__: specifies the LPTIM6 clock source. This parameter can be one of the following values:
  - RCC_LPTIM6CLKSOURCE_PCLK3 PCLK3 selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_PLL2P PLL2P selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_PLL3R PLL3R selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_LSE LSE selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_LSI LSI selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_CLKP CLKP selected as LPTIM6 clock

Return value:

- None

__HAL_RCC_GET_LPTIM6_SOURCE

Description:

- Macro to get the LPTIM6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM6CLKSOURCE_PCLK3 PCLK3 selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_PLL2P PLL2P selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_PLL3R PLL3R selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_LSE LSE selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_LSI LSI selected as LPTIM6 clock
  - RCC_LPTIM6CLKSOURCE_CLKP CLKP selected as LPTIM6 clock

__HAL_RCC_SPI1_CONFIG

Description:

- macro to configure the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI1CLKSOURCE_PLL1Q PLL1Q selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PLL2P PLL2P selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PLL3P PLL3P selected as SPI1 clock (*)
  - RCC_SPI1CLKSOURCE_PIN External Clock selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_CLKP CLKP selected as SPI1 clock

__HAL_RCC_GET_SPI1_SOURCE

Description:

- macro to get the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI1CLKSOURCE_PLL1Q PLL1Q selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PLL2P PLL2P selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_PLL3P PLL3P selected as SPI1 clock (*)
  - RCC_SPI1CLKSOURCE_PIN External Clock selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_CLKP CLKP selected as SPI1 clock

__HAL_RCC_SPI2_CONFIG

Description:

- macro to configure the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI2CLKSOURCE_PLL1Q PLL1Q selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PLL2P PLL2P selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PLL3P PLL3P selected as SPI2 clock (*)
  - RCC_SPI2CLKSOURCE_PIN External Clock selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_CLKP CLKP selected as SPI2 clock

__HAL_RCC_GET_SPI2_SOURCE

Description:

- macro to get the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI2CLKSOURCE_PLL1Q PLL1Q selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PLL2P PLL2P selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_PLL3P PLL3P selected as SPI2 clock (*)
  - RCC_SPI2CLKSOURCE_PIN External Clock selected as SPI2 clock
  - RCC_SPI2CLKSOURCE_CLKP CLKP selected as SPI2 clock

__HAL_RCC_SPI3_CONFIG

Description:

- macro to configure the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI3CLKSOURCE_PLL1Q PLL1Q used as SPI3 clock
  - RCC_SPI3CLKSOURCE_PLL2P PLL2P used as SPI3 clock
  - RCC_SPI3CLKSOURCE_PLL3P PLL3P used as SPI3 clock (*)
  - RCC_SPI3CLKSOURCE_PIN External Clock used as SPI3 clock
  - RCC_SPI3CLKSOURCE_CLKP CLKP used as SPI3 clock

__HAL_RCC_GET_SPI3_SOURCE

Description:

- macro to get the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI3CLKSOURCE_PLL1Q PLL1Q used as SPI3 clock
  - RCC_SPI3CLKSOURCE_PLL2P PLL2P used as SPI3 clock
  - RCC_SPI3CLKSOURCE_PLL3P PLL3P used as SPI3 clock (*)
  - RCC_SPI3CLKSOURCE_PIN External Clock used as SPI3 clock
  - RCC_SPI3CLKSOURCE_CLKP CLKP used as SPI3 clock

__HAL_RCC_SPI4_CONFIG

Description:

- macro to configure the SPI4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI4CLKSOURCE_PCLK2 PCLK2 used as SPI4 clock
  - RCC_SPI4CLKSOURCE_PLL2Q PLL2Q used as SPI4 clock
  - RCC_SPI4CLKSOURCE_PLL3Q PLL3Q used as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSI HSI used as SPI4 clock
  - RCC_SPI4CLKSOURCE_CSI CSI Clock used as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSE HSE Clock used as SPI4 clock

__HAL_RCC_GET_SPI4_SOURCE

Description:

- macro to get the SPI4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI4CLKSOURCE_PCLK2 PCLK2 used as SPI4 clock
  - RCC_SPI4CLKSOURCE_PLL2Q PLL2Q used as SPI4 clock
  - RCC_SPI4CLKSOURCE_PLL3Q PLL3Q used as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSI HSI used as SPI4 clock
  - RCC_SPI4CLKSOURCE_CSI CSI used as SPI4 clock
  - RCC_SPI4CLKSOURCE_HSE HSE used as SPI4 clock

__HAL_RCC_SPI5_CONFIG

Description:

- macro to configure the SPI5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI5CLKSOURCE_PCLK3 PCLK3 used as SPI5 clock
  - RCC_SPI5CLKSOURCE_PLL2Q PLL2Q used as SPI5 clock
  - RCC_SPI5CLKSOURCE_PLL3Q PLL3Q used as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSI HSI used as SPI5 clock
  - RCC_SPI5CLKSOURCE_CSI CSI Clock used as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSE HSE Clock used as SPI5 clock

__HAL_RCC_GET_SPI5_SOURCE

Description:

- macro to get the SPI5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI5CLKSOURCE_PCLK3 PCLK3 used as SPI5 clock
  - RCC_SPI5CLKSOURCE_PLL2Q PLL2Q used as SPI5 clock
  - RCC_SPI5CLKSOURCE_PLL3Q PLL3Q used as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSI HSI used as SPI5 clock
  - RCC_SPI5CLKSOURCE_CSI CSI used as SPI5 clock
  - RCC_SPI5CLKSOURCE_HSE HSE used as SPI5 clock

__HAL_RCC_SPI6_CONFIG

Description:

- macro to configure the SPI6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI6CLKSOURCE_PCLK2 PCLK2 used as SPI6 clock
  - RCC_SPI6CLKSOURCE_PLL2Q PLL2Q used as SPI6 clock
  - RCC_SPI6CLKSOURCE_PLL3Q PLL3Q used as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSI HSI used as SPI6 clock
  - RCC_SPI6CLKSOURCE_CSI CSI used as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSE HSE used as SPI6 clock

__HAL_RCC_GET_SPI6_SOURCE

Description:

- macro to get the SPI6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI6CLKSOURCE_PCLK2 PCLK2 used as SPI6 clock
  - RCC_SPI6CLKSOURCE_PLL2Q PLL2Q used as SPI6 clock
  - RCC_SPI6CLKSOURCE_PLL3Q PLL3Q used as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSI HSI used as SPI6 clock
  - RCC_SPI6CLKSOURCE_CSI CSI used as SPI6 clock
  - RCC_SPI6CLKSOURCE_HSE HSE used as SPI6 clock

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_PLL3R PLL3R selected as I2C1 clock (*)
  - RCC_I2C1CLKSOURCE_PLL2R PLL2R selected as I2C1 clock (**)
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CSI CSI selected as I2C1 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines. (**) : For stm32h503xx family line.

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_PLL3R PLL3R selected as I2C1 clock (*)
  - RCC_I2C1CLKSOURCE_PLL2R PLL2R selected as I2C1 clock (**)
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CSI CSI selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2_CLKSOURCE__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_PLL3R PLL3R selected as I2C2 clock (*)
  - RCC_I2C2CLKSOURCE_PLL2R PLL2R selected as I2C2 clock (**)
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CSI CSI selected as I2C2 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines. (**) : For stm32h503xx family line.

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_PLL3R PLL3R selected as I2C2 clock (*)
  - RCC_I2C2CLKSOURCE_PLL2R PLL2R selected as I2C2 clock (**)
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CSI CSI selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK3 PCLK3 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_PLL3R PLL3R selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CSI CSI selected as I2C3 clock

Return value:

- None

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_PCLK3 PCLK3 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_PLL3R PLL3R selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CSI CSI selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- Macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4_CLKSOURCE__: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK3 PCLK3 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_PLL3R PLL3R selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CSI CSI selected as I2C4 clock

Return value:

- None

__HAL_RCC_GET_I2C4_SOURCE

Description:

- Macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C4CLKSOURCE_PCLK3 PCLK3 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_PLL3R PLL3R selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CSI CSI selected as I2C4 clock

__HAL_RCC_I3C1_CONFIG

Description:

- Macro to configure the I3C1 clock (I3C1CLK).

Parameters:

- __I3C1_CLKSOURCE__: specifies the I3C1 clock source. This parameter can be one of the following values:
  - RCC_I3C1CLKSOURCE_PCLK1 PCLK1 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_PLL3R PLL3R selected as I3C1 clock (*)
  - RCC_I3C1CLKSOURCE_PLL2R PLL2R selected as I3C1 clock (**)
  - RCC_I3C1CLKSOURCE_HSI HSI selected as I3C1 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines. (**) : For stm32h503xx family line.

__HAL_RCC_GET_I3C1_SOURCE

Description:

- Macro to get the I3C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I3C1CLKSOURCE_PCLK1 PCLK1 selected as I3C1 clock
  - RCC_I3C1CLKSOURCE_PLL3R PLL3R selected as I3C1 clock (*)
  - RCC_I3C1CLKSOURCE_PLL2R PLL2R selected as I3C1 clock (**)
  - RCC_I3C1CLKSOURCE_HSI HSI selected as I3C1 clock

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL2Q PLL2Q selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL3Q PLL3Q selected as USART1 clock (*)
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_CSI CSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK2 selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL2Q PLL2Q selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL3Q PLL3Q selected as USART1 clock (*)
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_CSI CSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK2 selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL2Q PLL2Q selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL3Q PLL3Q selected as USART2 clock (*)
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_CSI CSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK2 selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL2Q PLL2Q selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL3Q PLL3Q selected as USART2 clock (*)
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_CSI CSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3_CLKSOURCE__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK2 selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL2Q PLL2Q selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL3Q PLL3Q selected as USART3 clock (*)
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_CSI CSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

Return value:

- None: (*) : For stm32h56xxx and stm32h57xxx family lines.

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK2 selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL2Q PLL2Q selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL3Q PLL3Q selected as USART3 clock (*)
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_CSI CSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4_CLKSOURCE__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL2Q PLL2Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL3Q PLL3Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_CSI CSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

Return value:

- None

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL2Q PLL2Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL3Q PLL3Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_CSI CSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5_CLKSOURCE__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL2Q PLL2Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL3Q PLL3Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_CSI CSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

Return value:

- None

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL2Q PLL2Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL3Q PLL3Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_CSI CSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

__HAL_RCC_USART6_CONFIG

Description:

- Macro to configure the USART6 clock (USART6CLK).

Parameters:

- __USART6_CLKSOURCE__: specifies the USART6 clock source. This parameter can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK1 PCLK2 selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL2Q PLL2Q selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL3Q PLL3Q selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_CSI CSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock

Return value:

- None

__HAL_RCC_GET_USART6_SOURCE

Description:

- Macro to get the USART6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK1 PCLK1 selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL2Q PLL2Q selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL3Q PLL3Q selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_CSI CSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock

__HAL_RCC_UART7_CONFIG

Description:

- Macro to configure the UART7 clock (UART7CLK).

Parameters:

- __UART7_CLKSOURCE__: specifies the UART7 clock source. This parameter can be one of the following values:
  - RCC_UART7CLKSOURCE_PCLK1 PCLK1 selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL2Q PLL2Q selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL3Q PLL3Q selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI HSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_CSI CSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE LSE selected as UART7 clock

Return value:

- None

__HAL_RCC_GET_UART7_SOURCE

Description:

- Macro to get the UART7 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART7CLKSOURCE_PCLK1 PCLK1 selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL2Q PLL2Q selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL3Q PLL3Q selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI HSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_CSI CSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE LSE selected as UART7 clock

__HAL_RCC_UART8_CONFIG

Description:

- Macro to configure the UART8 clock (UART8CLK).

Parameters:

- __UART8_CLKSOURCE__: specifies the UART8 clock source. This parameter can be one of the following values:
  - RCC_UART8CLKSOURCE_PCLK1 PCLK1 selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL2Q PLL2Q selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL3Q PLL3Q selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI HSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_CSI CSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE LSE selected as UART8 clock

Return value:

- None

__HAL_RCC_GET_UART8_SOURCE

Description:

- Macro to get the UART8 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART8CLKSOURCE_PCLK1 PCLK1 selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL2Q PLL2Q selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL3Q PLL3Q selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI HSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_CSI CSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE LSE selected as UART8 clock

__HAL_RCC_UART9_CONFIG

Description:

- Macro to configure the UART9 clock (UART9CLK).

Parameters:

- __UART9_CLKSOURCE__: specifies the UART9 clock source. This parameter can be one of the following values:
  - RCC_UART9CLKSOURCE_PCLK1 PCLK1 selected as UART9 clock
  - RCC_UART9CLKSOURCE_PLL2Q PLL2Q selected as UART9 clock
  - RCC_UART9CLKSOURCE_PLL3Q PLL3Q selected as UART9 clock
  - RCC_UART9CLKSOURCE_HSI HSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_CSI CSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_LSE LSE selected as UART9 clock

Return value:

- None

__HAL_RCC_GET_UART9_SOURCE

Description:

- Macro to get the UART9 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART9CLKSOURCE_PCLK1 PCLK1 selected as UART9 clock
  - RCC_UART9CLKSOURCE_PLL2Q PLL2Q selected as UART9 clock
  - RCC_UART9CLKSOURCE_PLL3Q PLL3Q selected as UART9 clock
  - RCC_UART9CLKSOURCE_HSI HSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_CSI CSI selected as UART9 clock
  - RCC_UART9CLKSOURCE_LSE LSE selected as UART9 clock

__HAL_RCC_USART10_CONFIG

Description:

- Macro to configure the USART10 clock (USART10CLK).

Parameters:

- __USART10_CLKSOURCE__: specifies the USART10 clock source. This parameter can be one of the following values:
  - RCC_USART10CLKSOURCE_PCLK1 PCLK1 selected as USART10 clock
  - RCC_USART10CLKSOURCE_PLL2Q PLL2Q selected as USART10 clock
  - RCC_USART10CLKSOURCE_PLL3Q PLL3Q selected as USART10 clock
  - RCC_USART10CLKSOURCE_HSI HSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_CSI CSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_LSE LSE selected as USART10 clock

Return value:

- None

__HAL_RCC_GET_USART10_SOURCE

Description:

- Macro to get the USART10 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART10CLKSOURCE_PCLK1 PCLK1 selected as USART10 clock
  - RCC_USART10CLKSOURCE_PLL2Q PLL2Q selected as USART10 clock
  - RCC_USART10CLKSOURCE_PLL3Q PLL3Q selected as USART10 clock
  - RCC_USART10CLKSOURCE_HSI HSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_CSI CSI selected as USART10 clock
  - RCC_USART10CLKSOURCE_LSE LSE selected as USART10 clock

__HAL_RCC_USART11_CONFIG

Description:

- Macro to configure the USART11 clock (USART11CLK).

Parameters:

- __USART11_CLKSOURCE__: specifies the USART11 clock source. This parameter can be one of the following values:
  - RCC_USART11CLKSOURCE_PCLK1 PCLK1 selected as USART11 clock
  - RCC_USART11CLKSOURCE_PLL2Q PLL2Q selected as USART11 clock
  - RCC_USART11CLKSOURCE_PLL3Q PLL3Q selected as USART11 clock
  - RCC_USART11CLKSOURCE_HSI HSI selected as USART11 clock
  - RCC_USART11CLKSOURCE_CSI CSI selected as USART11 clock
  - RCC_USART11CLKSOURCE_LSE LSE selected as USART11 clock

Return value:

- None

__HAL_RCC_GET_USART11_SOURCE

Description:

- Macro to get the USART11 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART11CLKSOURCE_PCLK1 PCLK1 selected as USART11 clock
  - RCC_USART11CLKSOURCE_PLL2Q PLL2Q selected as USART11 clock
  - RCC_USART11CLKSOURCE_PLL3Q PLL3Q selected as USART11 clock
  - RCC_USART11CLKSOURCE_HSI HSI selected as USART11 clock
  - RCC_USART11CLKSOURCE_CSI CSI selected as USART11 clock
  - RCC_USART11CLKSOURCE_LSE LSE selected as USART11 clock

__HAL_RCC_UART12_CONFIG

Description:

- Macro to configure the UART12 clock (UART12CLK).

Parameters:

- __UART12_CLKSOURCE__: specifies the UART12 clock source. This parameter can be one of the following values:
  - RCC_UART12CLKSOURCE_PCLK1 PCLK1 selected as UART12 clock
  - RCC_UART12CLKSOURCE_PLL2Q PLL2Q selected as UART12 clock
  - RCC_UART12CLKSOURCE_PLL3Q PLL3Q selected as UART12 clock
  - RCC_UART12CLKSOURCE_HSI HSI selected as UART12 clock
  - RCC_UART12CLKSOURCE_CSI CSI selected as UART12 clock
  - RCC_UART12CLKSOURCE_LSE LSE selected as UART12 clock

Return value:

- None

__HAL_RCC_GET_UART12_SOURCE

Description:

- Macro to get the UART12 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART12CLKSOURCE_PCLK1 PCLK1 selected as UART12 clock
  - RCC_UART12CLKSOURCE_PLL2Q PLL2Q selected as UART12 clock
  - RCC_UART12CLKSOURCE_PLL3Q PLL3Q selected as UART12 clock
  - RCC_UART12CLKSOURCE_HSI HSI selected as UART12 clock
  - RCC_UART12CLKSOURCE_CSI CSI selected as UART12 clock
  - RCC_UART12CLKSOURCE_LSE LSE selected as UART12 clock

__HAL_RCC_LPUART1_CONFIG

Description:

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK3 PCLK3 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL2Q PLL2Q selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL3Q PLL3Q selected as LPUART1 clock (*)
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CSI CSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

Return value:

- None

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_PCLK3 PCLK3 selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL2Q PLL2Q selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL3Q PLL3Q selected as LPUART1 clock (*)
  - RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CSI CSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_OSPI_CONFIG

Description:

- Macro to configure the OctoSPI clock.

Parameters:

- __OSPI_CLKSOURCE__: specifies the OctoSPI clock source. This parameter can be one of the following values:
  - RCC_OSPICLKSOURCE_HCLK HCLK selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_PLL1Q PLL1Q divider clock selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_PLL2R PLL2R divider clock selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_CLKP CLKP selected as OctoSPI clock

Return value:

- None

__HAL_RCC_GET_OSPI_SOURCE

Description:

- Macro to get the OctoSPI clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_OSPICLKSOURCE_HCLK HCLK selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_PLL1Q PLL1Q divider clock selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_PLL2R PLL2R divider clock selected as OctoSPI clock
  - RCC_OSPICLKSOURCE_CLKP CLKP selected as OctoSPI clock

__HAL_RCC_SDMMC1_CONFIG

Description:

- Macro to configure the SDMMC1 clock (SDMMCCLK).

Parameters:

- __SDMMC1_CLKSOURCE__: specifies the SDMMC1 clock source. This parameter can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_PLL1Q PLL1Q selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLL2R PLL2R selected as SDMMC1 clock

__HAL_RCC_GET_SDMMC1_SOURCE

Description:

- macro to get the SDMMC1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SDMMC1CLKSOURCE_PLL1Q PLL1Q selected as SDMMC1 clock
  - RCC_SDMMC1CLKSOURCE_PLL2R PLL2R selected as SDMMC1 clock

__HAL_RCC_SDMMC2_CONFIG

Description:

- Macro to configure the SDMMC2 clock (SDMMCCLK).

Parameters:

- __SDMMC2_CLKSOURCE__: specifies the SDMMC2 clock source. This parameter can be one of the following values:
  - RCC_SDMMC2CLKSOURCE_PLL1Q PLL1Q selected as SDMMC2 clock
  - RCC_SDMMC2CLKSOURCE_PLL2R PLL2R selected as SDMMC2 clock

__HAL_RCC_GET_SDMMC2_SOURCE

Description:

- macro to get the SDMMC2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SDMMC2CLKSOURCE_PLL1Q PLL1Q selected as SDMMC2 clock
  - RCC_SDMMC2CLKSOURCE_PLL2R PLL2R selected as SDMMC2 clock

__HAL_RCC_RNG_CONFIG

Description:

- macro to configure the RNG clock (RNGCLK).

Parameters:

- __RNGCLKSource__: specifies the RNG clock source. This parameter can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL1Q: PLL1Q selected as RNG clock
  - RCC_RNGCLKSOURCE_LSE: LSE selected as RNG clock
  - RCC_RNGCLKSOURCE_LSI: LSI selected as RNG clock

__HAL_RCC_GET_RNG_SOURCE

Description:

- macro to get the RNG clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL1Q: PLL1Q selected as RNG clock
  - RCC_RNGCLKSOURCE_LSE: LSE selected as RNG clock
  - RCC_RNGCLKSOURCE_LSI: LSI selected as RNG clock

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to configure the SAI1 clock source.

Parameters:

- __SAI1_CLKSOURCE__: defines the SAI1 clock source. This parameter can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLL1Q PLL1Q selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PLL2P PLL2Pselected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PLL3P PLL3P selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PIN External clock selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_CLKP CLKP selected as SAI1 clock

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLL1Q PLL1Q selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PLL2P PLL2P selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PLL3P PLL3P selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_PIN External clock selected as SAI1 clock
  - RCC_SAI1CLKSOURCE_CLKP CLKP selected as SAI1 clock

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to configure the SAI2 clock source.

Parameters:

- __SAI2_CLKSOURCE__: defines the SAI2 clock source. This parameter can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL1Q PLL1Q selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PLL2P PLL2P selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PLL3P PLL3P selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PIN External clock selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_CLKP CLKP selected as SAI2 clock

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL1Q PLL1Q selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PLL2P PLL2P selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PLL3P PLL3P selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_PIN External clock selected as SAI2 clock
  - RCC_SAI2CLKSOURCE_CLKP CLKP selected as SAI2 clock

__HAL_RCC_CLKP_CONFIG

Description:

- Macro to configure the CLKP : Oscillator clock for peripheral.

Parameters:

- __CLKPSource__: specifies Oscillator clock for peripheral This parameter can be one of the following values:
  - RCC_CLKPSOURCE_HSI HSI oscillator selected as clock for peripheral
  - RCC_CLKPSOURCE_CSI CSI oscillator selected as clock for peripheral
  - RCC_CLKPSOURCE_HSE HSE oscillator selected as clock for peripheral

__HAL_RCC_GET_CLKP_SOURCE

Description:

- Macro to get the oscillator clock for peripheral clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CLKPSOURCE_HSI HSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_CSI CSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_HSE HSE selected Oscillator clock for peripheral

__HAL_RCC_CEC_CONFIG

Description:

- Macro to configure the CEC clock (CECCLK)

Parameters:

- __CECCLKSource__: specifies the CEC clock source. This parameter can be one of the following values:
  - RCC_CECCLKSOURCE_LSE LSE selected as CEC clock
  - RCC_CECCLKSOURCE_LSI LSI selected as CEC clock
  - RCC_CECCLKSOURCE_CSI_DIV122 CSI Divided by 122 selected as CEC clock

__HAL_RCC_GET_CEC_SOURCE

Description:

- Macro to get the CEC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CECCLKSOURCE_LSE: LSE selected as CEC clock
  - RCC_CECCLKSOURCE_LSI: LSI selected as CEC clock
  - RCC_CECCLKSOURCE_CSI_DIV122: CSI Divided by 122 selected as CEC clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USBCLKSource__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_PLL1Q PLL1Q selected as USB clock
  - RCC_USBCLKSOURCE_PLL3Q PLL3Q selected as USB clock (*)
  - RCC_USBCLKSOURCE_PLL2Q PLL2Q selected as USB clock (**)
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBCLKSOURCE_PLL1Q PLL1Q selected as USB clock
  - RCC_USBCLKSOURCE_PLL3Q PLL3Q selected as USB clock (*)
  - RCC_USBCLKSOURCE_PLL2Q PLL2Q selected as USB clock (**)
  - RCC_USBCLKSOURCE_HSI48 HSI48 selected as USB clock

__HAL_RCC_TIMCLKPRESCALER

Description:

- Macro to configure the Timers clocks prescalers.

Parameters:

- __PRESC__: specifies the Timers clocks prescalers selection This parameter can be one of the following values:
  - RCC_TIMPRES_DEACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if PPREx is corresponding to division by 1 or 2, else it is equal to 2 x Frcc_pclkx (default after reset)
  - RCC_TIMPRES_ACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if PPREx is corresponding to division by 1, 2 or 4, else it is equal to 4 x Frcc_pclkx

__HAL_RCC_CRS_ENABLE_IT

Description:

- Enable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_DISABLE_IT

Description:

- Disable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:

- Check whether the CRS interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- The: new state of __INTERRUPT__ (0 or 1).

RCC_CRS_IT_ERROR_MASK

Description:

- Clear the CRS interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt
  - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
  - RCC_CRS_IT_SYNCERR SYNC error interrupt
  - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

__HAL_RCC_CRS_CLEAR_IT

__HAL_RCC_CRS_GET_FLAG

Description:

- Check whether the specified CRS flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- The: new state of _FLAG_ (TRUE or FALSE).

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

__HAL_RCC_CRS_CLEAR_FLAG

FDCAN Kernel Clock Source

RCC_FDCANCLKSOURCE_HSE

RCC_FDCANCLKSOURCE_PLL1Q

RCC_FDCANCLKSOURCE_PLL2Q

I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_PLL3R

RCC_I2C1CLKSOURCE_HSI

RCC_I2C1CLKSOURCE_CSI

I2C2 Clock Source

RCC_I2C2CLKSOURCE_PCLK1

RCC_I2C2CLKSOURCE_PLL3R

RCC_I2C2CLKSOURCE_HSI

RCC_I2C2CLKSOURCE_CSI

I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK3

RCC_I2C3CLKSOURCE_PLL3R

RCC_I2C3CLKSOURCE_HSI

RCC_I2C3CLKSOURCE_CSI

I2C4 Clock Source

RCC_I2C4CLKSOURCE_PCLK3

RCC_I2C4CLKSOURCE_PLL3R

RCC_I2C4CLKSOURCE_HSI

RCC_I2C4CLKSOURCE_CSI

I3C1 Clock Source

RCC_I3C1CLKSOURCE_PCLK1

RCC_I3C1CLKSOURCE_PLL3R

RCC_I3C1CLKSOURCE_HSI

LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK3

RCC_LPTIM1CLKSOURCE_PLL2P

RCC_LPTIM1CLKSOURCE_PLL3R

RCC_LPTIM1CLKSOURCE_LSE

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_CLKP

LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_PCLK1

RCC_LPTIM2CLKSOURCE_PLL2P

RCC_LPTIM2CLKSOURCE_PLL3R

RCC_LPTIM2CLKSOURCE_LSE

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_CLKP

LPTIM3 Clock Source

RCC_LPTIM3CLKSOURCE_PCLK3

RCC_LPTIM3CLKSOURCE_PLL2P

RCC_LPTIM3CLKSOURCE_PLL3R

RCC_LPTIM3CLKSOURCE_LSE

RCC_LPTIM3CLKSOURCE_LSI

RCC_LPTIM3CLKSOURCE_CLKP

LPTIM4 Clock Source

RCC_LPTIM4CLKSOURCE_PCLK3

RCC_LPTIM4CLKSOURCE_PLL2P

RCC_LPTIM4CLKSOURCE_PLL3R

RCC_LPTIM4CLKSOURCE_LSE

RCC_LPTIM4CLKSOURCE_LSI

RCC_LPTIM4CLKSOURCE_CLKP

LPTIM5 Clock Source

RCC_LPTIM5CLKSOURCE_PCLK3

RCC_LPTIM5CLKSOURCE_PLL2P

RCC_LPTIM5CLKSOURCE_PLL3R

RCC_LPTIM5CLKSOURCE_LSE

RCC_LPTIM5CLKSOURCE_LSI

RCC_LPTIM5CLKSOURCE_CLKP

LPTIM6 Clock Source

RCC_LPTIM6CLKSOURCE_PCLK3

RCC_LPTIM6CLKSOURCE_PLL2P

RCC_LPTIM6CLKSOURCE_PLL3R

RCC_LPTIM6CLKSOURCE_LSE

RCC_LPTIM6CLKSOURCE_LSI

RCC_LPTIM6CLKSOURCE_CLKP

LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK3

RCC_LPUART1CLKSOURCE_PLL2Q

RCC_LPUART1CLKSOURCE_PLL3Q

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_CSI

RCC_LPUART1CLKSOURCE_LSE

Low Speed Microcontroller Clock Output Source

RCC_LSCOSOURCE_LSI

LSI selection for low speed microcontroller clock output

RCC_LSCOSOURCE_LSE

LSE selection for low speed microcontroller clock output

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO1 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO1 clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_PLL1QCLK: PLL1Q clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI48: HSI48 (48MHZ) selected as MCO1 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCO1 clock

__HAL_RCC_MCO2_CONFIG

Description:

- Macro to configure the MCO2 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO2 clock source. This parameter can be one of the following values:
  - RCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC_MCO2SOURCE_PLL2PCLK: PLL2P clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_PLL1PCLK: PLL1P clock selected as MCO2 source
  - RCC_MCO2SOURCE_CSI: CSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSI: LSI clock selected as MCO2 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCO2 clock

OctoSPI Clock Source

RCC_OSPICLKSOURCE_HCLK

RCC_OSPICLKSOURCE_PLL1Q

RCC_OSPICLKSOURCE_PLL2R

RCC_OSPICLKSOURCE_CLKP

RCCEx Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_USART6

RCC_PERIPHCLK_UART7

RCC_PERIPHCLK_UART8

RCC_PERIPHCLK_UART9

RCC_PERIPHCLK_USART10

RCC_PERIPHCLK_USART11

RCC_PERIPHCLK_UART12

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_I3C1

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LPTIM2

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_SAI2

RCC_PERIPHCLK_ADCDAC

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_DAC

RCC_PERIPHCLK_SDMMC1

RCC_PERIPHCLK_SDMMC2

RCC_PERIPHCLK_CKPER

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_RNG

RCC_PERIPHCLK_SPI1

RCC_PERIPHCLK_SPI2

RCC_PERIPHCLK_SPI3

RCC_PERIPHCLK_SPI4

RCC_PERIPHCLK_SPI5

RCC_PERIPHCLK_SPI6

RCC_PERIPHCLK_OSPI

RCC_PERIPHCLK_FDCAN

RCC_PERIPHCLK_CEC

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_LPTIM3

RCC_PERIPHCLK_LPTIM4

RCC_PERIPHCLK_LPTIM5

RCC_PERIPHCLK_LPTIM6

RCC_PERIPHCLK_DAC_LP

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_TIM

RCCEx RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLL1Q

RCC_RNGCLKSOURCE_LSE

RCC_RNGCLKSOURCE_LSI

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLL1Q

RCC_SAI1CLKSOURCE_PLL2P

RCC_SAI1CLKSOURCE_PLL3P

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_CLKP

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLL1Q

RCC_SAI2CLKSOURCE_PLL2P

RCC_SAI2CLKSOURCE_PLL3P

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_CLKP

SDMMC1 Clock Source

RCC_SDMMC1CLKSOURCE_PLL1Q

RCC_SDMMC1CLKSOURCE_PLL2R

SDMMC2 Clock Source

RCC_SDMMC2CLKSOURCE_PLL1Q

RCC_SDMMC2CLKSOURCE_PLL2R

SPI1 Clock Source

RCC_SPI1CLKSOURCE_PLL1Q

RCC_SPI1CLKSOURCE_PLL2P

RCC_SPI1CLKSOURCE_PLL3P

RCC_SPI1CLKSOURCE_PIN

RCC_SPI1CLKSOURCE_CLKP

SPI2 Clock Source

RCC_SPI2CLKSOURCE_PLL1Q

RCC_SPI2CLKSOURCE_PLL2P

RCC_SPI2CLKSOURCE_PLL3P

RCC_SPI2CLKSOURCE_PIN

RCC_SPI2CLKSOURCE_CLKP

SPI3 Clock Source

RCC_SPI3CLKSOURCE_PLL1Q

RCC_SPI3CLKSOURCE_PLL2P

RCC_SPI3CLKSOURCE_PLL3P

RCC_SPI3CLKSOURCE_PIN

RCC_SPI3CLKSOURCE_CLKP

SPI4 Clock Source

RCC_SPI4CLKSOURCE_PCLK2

RCC_SPI4CLKSOURCE_PLL2Q

RCC_SPI4CLKSOURCE_PLL3Q

RCC_SPI4CLKSOURCE_HSI

RCC_SPI4CLKSOURCE_CSI

RCC_SPI4CLKSOURCE_HSE

SPI5 Clock Source

RCC_SPI5CLKSOURCE_PCLK3

RCC_SPI5CLKSOURCE_PLL2Q

RCC_SPI5CLKSOURCE_PLL3Q

RCC_SPI5CLKSOURCE_HSI

RCC_SPI5CLKSOURCE_CSI

RCC_SPI5CLKSOURCE_HSE

SPI6 Clock Source

RCC_SPI6CLKSOURCE_PCLK2

RCC_SPI6CLKSOURCE_PLL2Q

RCC_SPI6CLKSOURCE_PLL3Q

RCC_SPI6CLKSOURCE_HSI

RCC_SPI6CLKSOURCE_CSI

RCC_SPI6CLKSOURCE_HSE

RCCEx TIM Prescaler Selection

RCC_TIMPRES_DEACTIVATED

RCC_TIMPRES_ACTIVATED

UART12 Clock Source

RCC_UART12CLKSOURCE_PCLK1

RCC_UART12CLKSOURCE_PLL2Q

RCC_UART12CLKSOURCE_PLL3Q

RCC_UART12CLKSOURCE_HSI

RCC_UART12CLKSOURCE_CSI

RCC_UART12CLKSOURCE_LSE

UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_PLL2Q

RCC_UART4CLKSOURCE_PLL3Q

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_CSI

RCC_UART4CLKSOURCE_LSE

UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_PLL2Q

RCC_UART5CLKSOURCE_PLL3Q

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_CSI

RCC_UART5CLKSOURCE_LSE

UART7 Clock Source

RCC_UART7CLKSOURCE_PCLK1

RCC_UART7CLKSOURCE_PLL2Q

RCC_UART7CLKSOURCE_PLL3Q

RCC_UART7CLKSOURCE_HSI

RCC_UART7CLKSOURCE_CSI

RCC_UART7CLKSOURCE_LSE

UART8 Clock Source

RCC_UART8CLKSOURCE_PCLK1

RCC_UART8CLKSOURCE_PLL2Q

RCC_UART8CLKSOURCE_PLL3Q

RCC_UART8CLKSOURCE_HSI

RCC_UART8CLKSOURCE_CSI

RCC_UART8CLKSOURCE_LSE

UART9 Clock Source

RCC_UART9CLKSOURCE_PCLK1

RCC_UART9CLKSOURCE_PLL2Q

RCC_UART9CLKSOURCE_PLL3Q

RCC_UART9CLKSOURCE_HSI

RCC_UART9CLKSOURCE_CSI

RCC_UART9CLKSOURCE_LSE

USART10 Clock Source

RCC_USART10CLKSOURCE_PCLK1

RCC_USART10CLKSOURCE_PLL2Q

RCC_USART10CLKSOURCE_PLL3Q

RCC_USART10CLKSOURCE_HSI

RCC_USART10CLKSOURCE_CSI

RCC_USART10CLKSOURCE_LSE

USART11 Clock Source

RCC_USART11CLKSOURCE_PCLK1

RCC_USART11CLKSOURCE_PLL2Q

RCC_USART11CLKSOURCE_PLL3Q

RCC_USART11CLKSOURCE_HSI

RCC_USART11CLKSOURCE_CSI

RCC_USART11CLKSOURCE_LSE

USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_PLL2Q

RCC_USART1CLKSOURCE_PLL3Q

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_CSI

RCC_USART1CLKSOURCE_LSE

USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_PLL2Q

RCC_USART2CLKSOURCE_PLL3Q

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_CSI

RCC_USART2CLKSOURCE_LSE

USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_PLL2Q

RCC_USART3CLKSOURCE_PLL3Q

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_CSI

RCC_USART3CLKSOURCE_LSE

USART6 Clock Source

RCC_USART6CLKSOURCE_PCLK1

RCC_USART6CLKSOURCE_PLL2Q

RCC_USART6CLKSOURCE_PLL3Q

RCC_USART6CLKSOURCE_HSI

RCC_USART6CLKSOURCE_CSI

RCC_USART6CLKSOURCE_LSE

RCCEx USB Clock Source

RCC_USBCLKSOURCE_PLL1Q

RCC_USBCLKSOURCE_PLL3Q

RCC_USBCLKSOURCE_HSI48

61 HAL RNG Generic Driver

61.1 RNG Firmware driver registers structures

61.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32h5xx_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection

61.1.2 __RNG_HandleTypeDef

__RNG_HandleTypeDef is defined in the stm32h5xx_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***
- ***void(* ReadyDataCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RNG_TypeDef* __RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef __RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef __RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef __RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t __RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t __RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data
- ***void(* __RNG_HandleTypeDef::ReadyDataCallback)(struct __RNG_HandleTypeDef *hrng, uint32_t random32bit)***
  RNG Data Ready Callback
- ***void(* __RNG_HandleTypeDef::ErrorCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Error Callback
- ***void(* __RNG_HandleTypeDef::MspInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp Init callback
- ***void(* __RNG_HandleTypeDef::MspDeInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp DeInit callback

61.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

61.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

61.2.2 Callback registration

The compilation define USE_HAL_RNG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_RNG_RegisterCallback() to register a user callback. Function HAL_RNG_RegisterCallback() allows to register following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_RNG_UnRegisterCallback() to reset a callback to the default weak (overridden) function. HAL_RNG_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit.

For specific callback ReadyDataCallback, use dedicated register callbacks: respectively HAL_RNG_RegisterReadyDataCallback() , HAL_RNG_UnRegisterReadyDataCallback().

By default, after the HAL_RNG_Init() and when the state is HAL_RNG_STATE_RESET all callbacks are set to the corresponding weak (overridden) functions: example HAL_RNG_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (overridden) functions in the HAL_RNG_Init() and HAL_RNG_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_RNG_Init() and HAL_RNG_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_RNG_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_RNG_STATE_READY or HAL_RNG_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_RNG_RegisterCallback() before calling HAL_RNG_DeInit() or HAL_RNG_Init() function.

When The compilation define USE_HAL_RNG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (overridden) callbacks are used.

61.2.3 Initialization and configuration functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- ***HAL_RNG_Init()***
- ***HAL_RNG_DeInit()***
- ***HAL_RNG_MspInit()***
- ***HAL_RNG_MspDeInit()***
- ***HAL_RNG_RegisterCallback()***
- ***HAL_RNG_UnRegisterCallback()***

- *HAL_RNG_RegisterReadyDataCallback()*
- *HAL_RNG_UnRegisterReadyDataCallback()*

61.2.4 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_GenerateRandomNumber_IT()*
- *HAL_RNG_IRQHandler()*
- *HAL_RNG_ReadLastRandomNumber()*
- *HAL_RNG_ReadyDataCallback()*
- *HAL_RNG_ErrorCallback()*

61.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_RNG_GetState()*
- *HAL_RNG_GetError()*

61.2.6 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID, pRNG_CallbackTypeDef pCallback)

Function description

Register a User RNG Callback To be used instead of the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_RNG_ERROR_CB_ID Error callback ID
  - HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  - HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID)

Function description

Unregister an RNG Callback RNG callback is redirected to the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_RNG_ERROR_CB_ID Error callback ID
  - HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  - HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_RNG_RegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterReadyDataCallback (RNG_HandleTypeDef * hrng, pRNG_ReadyDataCallbackTypeDef pCallback)

Function description

Register Data Ready RNG Callback To be used instead of the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle
- **pCallback:** pointer to the Data Ready Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterReadyDataCallback (RNG_HandleTypeDef * hrng)

Function description

UnRegister the Data Ready RNG Callback Data Ready RNG Callback is redirected to the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle

Return values

- **HAL:** status

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)

Function description

Generates a 32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- This function checks value of RNG_FLAG_DRDY flag to know if valid random number is available in the DR register (RNG_FLAG_DRDY flag set whenever a random number is available through the RNG_DR register). After transitioning from 0 to 1 (random number available), RNG_FLAG_DRDY flag remains high until output buffer becomes empty after reading four words from the RNG_DR register, i.e. further function calls will immediately return a new u32 random number (additional words are available and can be read by the application, till RNG_FLAG_DRDY flag remains high).
- When no more random number data is available in DR register, RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

Generates a 32-bit random number in interrupt mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (const RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

Handles RNG interrupt request.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

RNG error callbacks.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

Data Ready callback in non-blocking mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** generated random number.

Return values

- **None:**

Notes

- When RNG_FLAG_DRDY flag value is set, first random number has been read from DR register in IRQ Handler and is provided as callback parameter. Depending on valid data available in the conditioning output buffer, additional words can be read by the application from DR register till DRDY bit remains high.

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (const RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (const RNG_HandleTypeDef * hrng)

Function description

Return the RNG handle error code.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **RNG:** Error Code

RNG_RecoverSeedError

Function name

HAL_StatusTypeDef RNG_RecoverSeedError (RNG_HandleTypeDef * hrng)

Function description

RNG sequence to recover from a seed error.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **HAL:** status

61.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

61.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

HAL_RNG_ERROR_SEED

Seed error

HAL_RNG_ERROR_CLOCK

Clock error

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection

RNG_CED_ENABLE

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check the selected RNG flag status.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  - RNG_FLAG_DRDY: Data ready
  - RNG_FLAG_CECS: Clock error current status
  - RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clears the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

Auto Reset configuration

RNG_ARDIS_ENABLE

automatic reset after seed error

RNG_ARDIS_DISABLE

Disable automatic reset after seed error

Value used to configure an internal

RNG_CLKDIV_BY_1

No clock division

RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

NIST Compliance configuration

RNG_NIST_COMPLIANT

NIST compliant configuration

RNG_CUSTOM_NIST

Custom NIST configuration

62 HAL RNG Extension Driver

62.1 RNGEx Firmware driver registers structures

62.1.1 RNG_ConfigTypeDef

RNG_ConfigTypeDef is defined in the stm32h5xx_hal_rng_ex.h

Data Fields

- ***uint32_t Config1***
- ***uint32_t Config2***
- ***uint32_t Config3***
- ***uint32_t ClockDivider***
- ***uint32_t NistCompliance***
- ***uint32_t AutoReset***
- ***uint32_t HealthTest***

Field Documentation

- ***uint32_t RNG_ConfigTypeDef::Config1***
  Config1 must be a value between 0 and 0x3F
- ***uint32_t RNG_ConfigTypeDef::Config2***
  Config2 must be a value between 0 and 0x7
- ***uint32_t RNG_ConfigTypeDef::Config3***
  Config3 must be a value between 0 and 0xF
- ***uint32_t RNG_ConfigTypeDef::ClockDivider***
  Clock Divider factor.This parameter can be a value of ***RNG_Ex_Clock_Divider_Factor***
- ***uint32_t RNG_ConfigTypeDef::NistCompliance***
  NIST compliance.This parameter can be a value of ***RNG_Ex_NIST_Compliance***
- ***uint32_t RNG_ConfigTypeDef::AutoReset***
  automatic reset When a noise source error occurs value of ***RNG_Ex_Auto_Reset***
- ***uint32_t RNG_ConfigTypeDef::HealthTest***
  RNG health test control must be a value between 0x0FFCABFF and 0x00005200

62.2 RNGEx Firmware driver defines

The following section lists the various define and macros of the module.

62.2.1 RNGEx

RNGEx

63 HAL RTC Generic Driver

63.1 RTC Firmware driver registers structures

63.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32h5xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***
- ***uint32_t OutPutPullUp***
- ***uint32_t BinMode***
- ***uint32_t BinMixBcdU***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***
  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::OutPutPullUp***
  Specifies the RTC Output Pull-Up mode. This parameter can be a value of ***RTC_Output_PullUp_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::BinMode***
  Specifies the RTC binary mode. This parameter can be a value of ***RTCEx_Binary_Mode***
- ***uint32_t RTC_InitTypeDef::BinMixBcdU***
  Specifies the BCD calendar update if and only if BinMode = RTC_BINARY_MIX. This parameter can be a value of ***RTCEx_Binary_mix_BCDU***

63.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32h5xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hour. This parameter must be a number between: Min_Data = 0 and Max_Data = 12 if the RTC_HOURFORMAT_12 is selected. This parameter must be a number between: Min_Data = 0 and Max_Data = 23 if the RTC_HOURFORMAT_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  Specifies the RTC_SSR RTC Sub Second register content. This field is not used by HAL_RTC_SetTime. If the free running 32 bit counter is not activated (mode binary none)
  - This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity else
  - This parameter corresponds to the free running 32 bit counter.
- ***uint32_t RTC_TimeTypeDef::SecondFraction***
  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

63.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32h5xx_hal_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***
  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***
  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

63.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32h5xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t BinaryAutoClr***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t FlagAutoClr***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. if Binary mode is RTC_BINARY_ONLY or is RTC_BINARY_MIX This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Masks_Definitions*** else if Binary mode is RTC_BINARY_NONE This parameter can be a value of ***RTC_Alarm_Sub_Seconds_BCD_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::BinaryAutoClr***
  Clear synchronously counter (RTC_SSR) on binary alarm. RTC_ALARMSUBSECONDBIN_AUTOCLR_YES must only be used if Binary mode is RTC_BINARY_ONLY This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Clear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::FlagAutoClr***
  Specifies the alarm trigger generation. This feature is meaningful to avoid any RTC software execution after configuration. This parameter can be a value of ***RTC_ALARM_Flag_AutoClear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm. This parameter can be a value of ***RTC_Alarms_Definitions***

63.1.5 __RTC_HandleTypeDef

__RTC_HandleTypeDef is defined in the stm32h5xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***
- ***void(* AlarmAEventCallback***
- ***void(* AlarmBEventCallback***
- ***void(* TimeStampEventCallback***
- ***void(* WakeUpTimerEventCallback***
- ***void(* SSRUEventCallback***
- ***void(* Tamper1EventCallback***
- ***void(* Tamper2EventCallback***
- ***void(* Tamper3EventCallback***
- ***void(* Tamper4EventCallback***

- ***void(* Tamper5EventCallback***
- ***void(* Tamper6EventCallback***
- ***void(* Tamper7EventCallback***
- ***void(* Tamper8EventCallback***
- ***void(* InternalTamper1EventCallback***
- ***void(* InternalTamper2EventCallback***
- ***void(* InternalTamper3EventCallback***
- ***void(* InternalTamper4EventCallback***
- ***void(* InternalTamper5EventCallback***
- ***void(* InternalTamper6EventCallback***
- ***void(* InternalTamper7EventCallback***
- ***void(* InternalTamper8EventCallback***
- ***void(* InternalTamper9EventCallback***
- ***void(* InternalTamper11EventCallback***
- ***void(* InternalTamper12EventCallback***
- ***void(* InternalTamper13EventCallback***
- ***void(* InternalTamper15EventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RTC_TypeDef* __RTC_HandleTypeDef::Instance***
  Legacy register base address. Not used anymore, the driver directly uses cmsis base address
- ***RTC_InitTypeDef __RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef __RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef __RTC_HandleTypeDef::State***
  Time communication state
- ***void(* __RTC_HandleTypeDef::AlarmAEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm A Event callback
- ***void(* __RTC_HandleTypeDef::AlarmBEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm B Event callback
- ***void(* __RTC_HandleTypeDef::TimeStampEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC TimeStamp Event callback
- ***void(* __RTC_HandleTypeDef::WakeUpTimerEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC WakeUpTimer Event callback
- ***void(* __RTC_HandleTypeDef::SSRUEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC SSRU Event callback
- ***void(* __RTC_HandleTypeDef::Tamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 1 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper4EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 4 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 7 Event callback

- ***void(* __RTC_HandleTypeDef::Tamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 1 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper4EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 4 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 7 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper9EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 9 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper11EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 11 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper12EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 12 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper13EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 13 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper15EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 15 Event callback
- ***void(* __RTC_HandleTypeDef::MspInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp Init callback
- ***void(* __RTC_HandleTypeDef::MspDeInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp DeInit callback

63.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

63.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

63.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

63.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.

- Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
- Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() function.

63.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

63.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

63.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   – A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   – When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_RegisterCallback()***

- *HAL_RTC_UnRegisterCallback()*
- *HAL_RTC_MspInit()*
- *HAL_RTC_MspDeInit()*

63.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features
This section contains the following APIs:

- *HAL_RTC_SetTime()*
- *HAL_RTC_GetTime()*
- *HAL_RTC_SetDate()*
- *HAL_RTC_GetDate()*
- *HAL_RTC_DST_Add1Hour()*
- *HAL_RTC_DST_Sub1Hour()*
- *HAL_RTC_DST_SetStoreOperation()*
- *HAL_RTC_DST_ClearStoreOperation()*
- *HAL_RTC_DST_ReadStoreOperation()*

63.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature
This section contains the following APIs:

- *HAL_RTC_SetAlarm()*
- *HAL_RTC_SetAlarm_IT()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_GetAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

63.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- *HAL_RTC_WaitForSynchro()*

63.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

63.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function does not reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_RegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID, pRTC_CallbackTypeDef pCallback)

Function description

Register a User RTC Callback To be used instead of the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  - HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  - HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  - HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  - HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  - HAL_RTC_TAMPER4_EVENT_CB_ID Tamper 4 Callback ID
  - HAL_RTC_TAMPER5_EVENT_CB_ID Tamper 5 Callback ID
  - HAL_RTC_TAMPER6_EVENT_CB_ID Tamper 6 Callback ID
  - HAL_RTC_TAMPER7_EVENT_CB_ID Tamper 7 Callback ID
  - HAL_RTC_TAMPER8_EVENT_CB_ID Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  - HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  - HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER4_EVENT_CB_ID Internal Tamper 4 Callback ID
  - HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  - HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  - HAL_RTC_INTERNAL_TAMPER7_EVENT_CB_ID Internal Tamper 7 Callback ID
  - HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER9_EVENT_CB_ID Internal Tamper 9 Callback ID
  - HAL_RTC_INTERNAL_TAMPER11_EVENT_CB_ID Internal Tamper 11 Callback ID
  - HAL_RTC_INTERNAL_TAMPER12_EVENT_CB_ID Internal Tamper 12 Callback ID
  - HAL_RTC_INTERNAL_TAMPER13_EVENT_CB_ID Internal Tamper 13 Callback ID
  - HAL_RTC_INTERNAL_TAMPER15_EVENT_CB_ID Internal Tamper 15 Callback ID
  - HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  - HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_RTC_RegisterCallback() may be called before HAL_RTC_Init() in HAL_RTC_STATE_RESET to register callbacks for HAL_RTC_MSPINIT_CB_ID and HAL_RTC_MSPDEINIT_CB_ID.

HAL_RTC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_UnRegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID)

Function description

Unregister an RTC Callback RTC callback is redirected to the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  - HAL_RTC_SSRU_EVENT_CB_ID SSRU Callback ID
  - HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  - HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  - HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  - HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  - HAL_RTC_TAMPER4_EVENT_CB_ID Tamper 4 Callback ID
  - HAL_RTC_TAMPER5_EVENT_CB_ID Tamper 5 Callback ID
  - HAL_RTC_TAMPER6_EVENT_CB_ID Tamper 6 Callback ID
  - HAL_RTC_TAMPER7_EVENT_CB_ID Tamper 7 Callback ID
  - HAL_RTC_TAMPER8_EVENT_CB_ID Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  - HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  - HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER4_EVENT_CB_ID Internal Tamper 4 Callback ID
  - HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  - HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  - HAL_RTC_INTERNAL_TAMPER7_EVENT_CB_ID Internal Tamper 7 Callback ID
  - HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER9_EVENT_CB_ID Internal Tamper 9 Callback ID
  - HAL_RTC_INTERNAL_TAMPER11_EVENT_CB_ID Internal Tamper 11 Callback ID
  - HAL_RTC_INTERNAL_TAMPER12_EVENT_CB_ID Internal Tamper 12 Callback ID
  - HAL_RTC_INTERNAL_TAMPER13_EVENT_CB_ID Internal Tamper 13 Callback ID
  - HAL_RTC_INTERNAL_TAMPER15_EVENT_CB_ID Internal Tamper 15 Callback ID
  - HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  - HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_RTC_UnRegisterCallback() may be called before HAL_RTC_Init() in HAL_RTC_STATE_RESET to un-register callbacks for HAL_RTC_MSPINIT_CB_ID and HAL_RTC_MSPDEINIT_CB_ID.

HAL_RTC_SetTime

Function name

HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure if Binary mode is RTC_BINARY_ONLY, this parameter is not used and RTC_SSR will be automatically reset to 0xFFFFFFFF else sTime->SubSeconds is not used and RTC_SSR will be automatically reset to the A 7-bit async prescaler (RTC_PRER_PREDIV_A)
- **Format:** Format of sTime->Hours, sTime->Minutes and sTime->Seconds. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (const RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** if Binary mode is RTC_BINARY_ONLY, sTime->SubSeconds only is updated else Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Format of sTime->Hours, sTime->Minutes and sTime->Seconds. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Set RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to date structure
- **Format:** Format of sDate->Year, sDate->Month and sDate->Weekday. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_GetDate

Function name

HAL_StatusTypeDef HAL_RTC_GetDate (const RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Get RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to Date structure
- **Format:** Format of sDate->Year, sDate->Month and sDate->Weekday. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

Notes

- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_DST_Add1Hour

Function name

void HAL_RTC_DST_Add1Hour (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Add one hour to the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_Sub1Hour

Function name

void HAL_RTC_DST_Sub1Hour (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Subtract one hour from the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_SetStoreOperation

Function name

void HAL_RTC_DST_SetStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Set the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

Notes

- It can be used by the software in order to memorize the DST status.

HAL_RTC_DST_ClearStoreOperation

Function name

void HAL_RTC_DST_ClearStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Clear the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_ReadStoreOperation

Function name

uint32_t HAL_RTC_DST_ReadStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Read the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **operation:** see RTC_StoreOperation_Definitions

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure if Binary mode is RTC_BINARY_ONLY, 3 fields only are used sAlarm->AlarmTime.SubSeconds sAlarm->AlarmSubSecondMask sAlarm->BinaryAutoClr
- **Format:** of the entered parameters. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure if Binary mode is RTC_BINARY_ONLY, 3 fields only are used sAlarm->AlarmTime.SubSeconds sAlarm->AlarmSubSecondMask sAlarm->BinaryAutoClr
- **Format:** Specifies the format of the entered parameters. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

Notes

- The application must ensure that the EXTI RTC interrupt line is enabled.

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

Deactivate the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (const RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

Get the RTC Alarm value and masks.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Alarm secure interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm A Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm A secure callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (const RTC_HandleTypeDef * hrtc)

Function description

Return the RTC handle state.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** state

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enter the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ExitInitMode

Function name

HAL_StatusTypeDef RTC_ExitInitMode (RTC_HandleTypeDef * hrtc)

Function description

Exit the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t Value)

Function description

Convert a 2 digit decimal to BCD format.

Parameters

- **Value:** Byte to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t Value)

Function description

Convert from 2 digit BCD to Binary.

Parameters

- **Value:** BCD value to be converted

Return values

- **Converted:** word

63.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

63.3.1 RTC

RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarms Flag Auto Clear Definitions

ALARM_FLAG_AUTOCLR_ENABLE

ALARM_FLAG_AUTOCLR_DISABLE

RTC Alarm Sub Seconds BCD Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] not used in Alarmcomparison. Only SS[0] is compared

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] not used in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] not used in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] not used in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] not used in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] not used in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] not used in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] not used in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] not used in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] not used in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] not used in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] not used in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] not used in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] not used in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC Clear Flags Definitions

RTC_CLEAR_SSRUF

Clear SSR underflow flag

RTC_CLEAR_ITSF

Clear Internal Time-stamp flag

RTC_CLEAR_TSOVF

Clear Time-stamp overflow flag

RTC_CLEAR_TSF

Clear Time-stamp flag

RTC_CLEAR_WUTF

Clear Wakeup timer flag

RTC_CLEAR_ALRBF

Clear Alarm B flag

RTC_CLEAR_ALRAF

Clear Alarm A flag

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_ADD1H

Description:

- Add 1 hour (summer time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

Notes:

- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_DAYLIGHT_SAVING_TIME_SUB1H

Description:

- Subtract 1 hour (winter time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

Notes:

- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarms flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarms pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_IS_CALENDAR_INITIALIZED

Description:

- Check whether if the RTC Calendar is initialized.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

RTC Flags Definitions

RTC_FLAG_RECALPF

Recalibration pending flag

RTC_FLAG_INITF

Initialization flag

RTC_FLAG_RSF

Registers synchronization flag

RTC_FLAG_INITS

Initialization status flag

RTC_FLAG_SHPF

Shift operation pending flag

RTC_FLAG_WUTWF

Wakeup timer write flag

RTC_FLAG_SSRUF

Clear SSR underflow flag

RTC_FLAG_ITSF

Clear Internal Time-stamp flag

RTC_FLAG_TSOVF

Clear Time-stamp overflow flag

RTC_FLAG_TSF

Clear Time-stamp flag

RTC_FLAG_WUTF

Clear Wakeup timer flag

RTC_FLAG_ALRBF

Clear Alarm B flag

RTC_FLAG_ALRAF

Clear Alarm A flag

RTC Flag Mask (5bits) for __HAL_RTC_GET_FLAG()

RTC_FLAG_MASK

RTC flags mask (5bits)

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_WUT

Enable Wakeup timer Interrupt

RTC_IT_SSRU

Enable SSR Underflow Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC_IT_ALRB

Enable Alarm B Interrupt

RTC Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_PULLUP

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Pull-Up ALARM OUT

RTC_OUTPUT_PULLUP_NONE

RTC_OUTPUT_PULLUP_ON

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_PUSHPULL

RTC_OUTPUT_TYPE_OPENDRAIN

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

64 HAL RTC Extension Driver

64.1 RTCEx Firmware driver registers structures

64.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger***
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the TAMP Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration. This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp. This parameter can be a value of ***RTCEx_Tamper_Pull_UP***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***

64.1.2 RTC_ATampInputTypeDef

RTC_ATampInputTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Enable***
- ***uint32_t Interrupt***
- ***uint32_t Output***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***

Field Documentation

- ***uint32_t RTC_ATampInputTypeDef::Enable***
  Specifies the Tamper input is active. This parameter can be a value of ***RTCEx_ActiveTamper_Enable***

- ***uint32_t RTC_ATampInputTypeDef::Interrupt***
  Specifies the interrupt mode. This parameter can be a value of ***RTCEx_ActiveTamper_Interrupt***
- ***uint32_t RTC_ATampInputTypeDef::Output***
  Specifies the TAMP output to be compared with. The same output can be used for several tamper inputs. This parameter can be a value of ***RTCEx_ActiveTamper_Sel***
- ***uint32_t RTC_ATampInputTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_ATampInputTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***

64.1.3 RTC_ActiveTampersTypeDef

RTC_ActiveTampersTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t ActiveFilter***
- ***uint32_t ActiveAsyncPrescaler***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t ActiveOutputChangePeriod***
- ***uint32_t Seed***
- ***RTC_ATampInputTypeDef TampInput***

Field Documentation

- ***uint32_t RTC_ActiveTampersTypeDef::ActiveFilter***
  Specifies the Active tamper filter enable. This parameter can be a value of ***RTCEx_ActiveTamper_Filter***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveAsyncPrescaler***
  Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of ***RTCEx_ActiveTamper_Async_prescaler***
- ***uint32_t RTC_ActiveTampersTypeDef::TimeStampOnTamperDetection***
  Specifies the timeStamp on tamper detection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveOutputChangePeriod***
  Specifies the Active Tamper output change period. This parameter can be a value from 0 to 7
- ***uint32_t RTC_ActiveTampersTypeDef::Seed[RTC_ATAMP_SEED_NB_UINT32]***
  Specifies the RNG Seed value. This parameter is an array of value from 0 to 0xFFFFFFFF
- ***RTC_ATampInputTypeDef RTC_ActiveTampersTypeDef::TampInput[RTC_TAMP_NB]***
  Specifies configuration of all active tampers. The index of TampInput[RTC_TAMP_NB] can be a value of RTCEx_ActiveTamper_Sel

64.1.4 RTC_InternalTamperTypeDef

RTC_InternalTamperTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t IntTamper***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t NoErase***

Field Documentation

- ***uint32_t RTC_InternalTamperTypeDef::IntTamper***
  Specifies the Internal Tamper Pin. This parameter can be a value of ***RTCEx_Internal_Tamper_Pins***
- ***uint32_t RTC_InternalTamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_InternalTamperTypeDef::NoErase***
  Specifies the internal Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***

64.1.5 RTC_SecureStateTypeDef

RTC_SecureStateTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcSecureFull***
- ***uint32_t rtcNonSecureFeatures***
- ***uint32_t tampSecureFull***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterSecure***

Field Documentation

- ***uint32_t RTC_SecureStateTypeDef::rtcSecureFull***
  Specifies If the RTC is fully secure or not. This parameter can be a value of ***RTCEx_RTC_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::rtcNonSecureFeatures***
  Specifies the non-secure features. This parameter is only relevant if RTC is not fully secure (rtcSecureFull == RTC_SECURE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_NonSecure_Features***
- ***uint32_t RTC_SecureStateTypeDef::tampSecureFull***
  Specifies If the TAMP is fully secure or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2. Zone 1 : read secure write secure. Zone 2 : read non-secure write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 63.3.1 RTC_PrivilegeStateTypeDef***
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3. Zone 3 : read non-secure write non-secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 63.3.1 RTC_PrivilegeStateTypeDef***
- ***uint32_t RTC_SecureStateTypeDef::MonotonicCounterSecure***
  Specifies If the monotonic counter is secure or not. This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Secure***

64.1.6 RTC_PrivilegeStateTypeDef

RTC_PrivilegeStateTypeDef is defined in the stm32h5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcPrivilegeFull***
- ***uint32_t rtcPrivilegeFeatures***
- ***uint32_t tampPrivilegeFull***
- ***uint32_t backupRegisterPrivZone***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterPrivilege***

Field Documentation

- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFull***
  Specifies If the RTC is fully privileged or not. This parameter can be a value of ***RTCEx_RTC_Privilege_Full***
- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFeatures***
  Specifies the privileged features. This parameter is only relevant if RTC is not fully privileged (rtcPrivilegeFull == RTC_PRIVILEGE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_Privilege_Features***
- ***uint32_t RTC_PrivilegeStateTypeDef::tampPrivilegeFull***
  Specifies If the TAMP is fully privileged or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Privilege_Full***
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterPrivZone***
  Specifies backup register zone to be privileged. This parameter can be a combination of ***RTCEx_Backup_Reg_Privilege_zone***. Warning : this parameter is writable in secure mode or if trustzone is disabled

- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2. Zone 1 : read secure write secure. Zone 2 : read non-secure write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register . Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 63.3.1 RTC_SecureStateTypeDef***
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3. Zone 3 : read non-secure write non-secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 63.3.1 RTC_SecureStateTypeDef***
- ***uint32_t RTC_PrivilegeStateTypeDef::MonotonicCounterPrivilege***
  Specifies If the monotonic counter is privileged or not. This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Privilege***

64.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

64.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 outputs pins (RTC_OUT1 & RTC_OUT2). To configure the outputs, use the HAL_RTC_Init() function.

- RTC_OUT1 and RTC_OUT2 which select one of the following two outputs:
- CALIB: 512Hz or 1Hz clock output (with an LSE frequency of 32.768kHz). To enable the CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- TAMPALRM: This output is the OR between rtc_tamp_evt and ALARM signals. ALARM is enabled by configuring the OSEL[1:0] bits in the RTC_CR register which select the alarm A, alarm B or wakeup outputs. rtc_tamp_evt is enabled by setting the TAMPOE bit in the RTC_CR register which selects the tamper event outputs.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

RTC synchronization

- The RTC can be finely adjusted using HAL_RTCEx_SetSynchroShift() function. Writing to RTC_SHIFTR can shift (either delay or advance) the clock with a resolution of 1 ck_apre period. The shift operation consists in adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this delays the clock.

Bypass shadow registers

- Enable bypass shadow registers using the HAL_RTCEx_EnableBypassShadow(). When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter. Thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (Stop or Standby), since the shadow registers are not updated during these modes.

RTC ultra-low-power mode

- Configure the RTC ultra-low-power mode using HAL_RTCEx_SetLowPowerCalib() function. In this case, the calibration mechanism is applied on ck_apre instead of RTCCLK. The resulting accuracy is the same, but the calibration is performed during a calibration cycle of about 220 x PREDIV_A x RTCCLK pulses instead of 220 RTCCLK pulses.

RTC subsecond register underflow interruption

- Enable the RTC SSRU interruption mode using HAL_RTCEx_SetSSRU_IT() function. In this case, when the SSR rolls under 0, an SSRU interruption is triggered. Disable the RTC SSRU interruption mode using HAL_RTCEx_DeactivateSSRU() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the TAMP_CR2 register.
- With new RTC tamper configuration, you have to call HAL_RTC_Init() in order to perform TAMP base address offset calculation.
- Enable Internal tamper using HAL_RTCEx_SetInternalTamper. IT mode can be chosen using setting Interrupt field.

Backup Data Registers and Device Secrets configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup registers, use the HAL_RTCEx_BKUPRead() function.
- To reset the RTC Backup registers and erase the device secrets, use HAL_RTCEx_BKUPErase() function.
- Enable the lock of the Boot hardware Key using the HAL_RTCEx_LockBootHardwareKey() function. The backup registers from TAMP_BKP0R to TAMP_BKP7R cannot be accessed neither in read nor in write (they are read as 0 and write ignore).
- Configure the erase of the Device Secrets using HAL_RTCEx_ConfigEraseDeviceSecrets() function.
- Block the access to the RTC Backup registers and all the device secrets using HAL_RTCEx_BKUPBlock() function.

Monotonic counter

- To increment the Monotonic counter, use the HAL_RTCEx_MonotonicCounterIncrement() function.
- To get the current value of the Monotonic counter, use the HAL_RTCEx_MonotonicCounterGet() function.

RTC & TAMP secure protection modes

- Set the security level of the RTC/TAMP/Backup registers using HAL_RTCEx_SecureModeSet() function. +) Get the security level of the RTC/TAMP/Backup registers using HAL_RTCEx_SecureModeGet() function.

RTC & TAMP privilege protection modes

- Set the privilege level of the RTC/TAMP/Backup registers using HAL_RTCEx_PrivilegeModeSet() function.
  +) Get the privilege level of the RTC/TAMP/Backup registers using HAL_RTCEx_PrivilegeModeGet() function.

64.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- *HAL_RTCEx_SetTimeStamp()*
- *HAL_RTCEx_SetTimeStamp_IT()*
- *HAL_RTCEx_DeactivateTimeStamp()*
- *HAL_RTCEx_SetInternalTimeStamp()*
- *HAL_RTCEx_DeactivateInternalTimeStamp()*
- *HAL_RTCEx_GetTimeStamp()*
- *HAL_RTCEx_TimeStampIRQHandler()*
- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*

64.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

64.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Set Low Power calibration parameter.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetLowPowerCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*

- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*
- *HAL_RTCEx_MonotonicCounterIncrement()*
- *HAL_RTCEx_MonotonicCounterGet()*
- *HAL_RTCEx_SetSSRU_IT()*
- *HAL_RTCEx_DeactivateSSRU()*
- *HAL_RTCEx_SSRUIRQHandler()*
- *HAL_RTCEx_SSRUEventCallback()*

64.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

64.2.6 Tamper functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that one you can select to output tamper event on RTC pin.
- Enable the Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP, timestamp using the HAL_RTCEx_SetTamper() function. You can configure Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the TAMP_TAMPCR register.
- Enable Internal Tamper and configure it with interrupt, timestamp using the HAL_RTCEx_SetInternalTamper() function.

This section contains the following APIs:

- *HAL_RTCEx_SetTamper()*
- *HAL_RTCEx_SetTamper_IT()*
- *HAL_RTCEx_DeactivateTamper()*
- *HAL_RTCEx_SetActiveTampers()*
- *HAL_RTCEx_SetActiveSeed()*
- *HAL_RTCEx_LockBootHardwareKey()*
- *HAL_RTCEx_DeactivateActiveTampers()*
- *HAL_RTCEx_PollForTamperEvent()*
- *HAL_RTCEx_SetInternalTamper()*
- *HAL_RTCEx_SetInternalTamper_IT()*
- *HAL_RTCEx_DeactivateInternalTamper()*
- *HAL_RTCEx_PollForInternalTamperEvent()*
- *HAL_RTCEx_TamperIRQHandler()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_Tamper4EventCallback()*
- *HAL_RTCEx_Tamper5EventCallback()*

- *HAL_RTCEx_Tamper6EventCallback()*
- *HAL_RTCEx_Tamper7EventCallback()*
- *HAL_RTCEx_Tamper8EventCallback()*
- *HAL_RTCEx_InternalTamper1EventCallback()*
- *HAL_RTCEx_InternalTamper2EventCallback()*
- *HAL_RTCEx_InternalTamper3EventCallback()*
- *HAL_RTCEx_InternalTamper4EventCallback()*
- *HAL_RTCEx_InternalTamper5EventCallback()*
- *HAL_RTCEx_InternalTamper6EventCallback()*
- *HAL_RTCEx_InternalTamper7EventCallback()*
- *HAL_RTCEx_InternalTamper8EventCallback()*
- *HAL_RTCEx_InternalTamper9EventCallback()*
- *HAL_RTCEx_InternalTamper11EventCallback()*
- *HAL_RTCEx_InternalTamper12EventCallback()*
- *HAL_RTCEx_InternalTamper13EventCallback()*
- *HAL_RTCEx_InternalTamper15EventCallback()*

64.2.7 Extended RTC Backup register functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that one you can select to output tamper event on RTC pin.

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_BKUPErase()*
- *HAL_RTCEx_BKUPBlock()*
- *HAL_RTCEx_BKUPUnblock()*
- *HAL_RTCEx_ConfigEraseDeviceSecrets()*

64.2.8 Extended RTC security functions

- Before calling security function, you have to call first HAL_RTC_Init() function.

This section contains the following APIs:

- *HAL_RTCEx_SecureModeGet()*
- *HAL_RTCEx_SecureModeSet()*

64.2.9 Extended RTC privilege functions

- Before calling privilege function, you have to call first HAL_RTC_Init() function.

This section contains the following APIs:

- *HAL_RTCEx_PrivilegeModeSet()*
- *HAL_RTCEx_PrivilegeModeGet()*

64.2.10 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp with Interrupt.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.
- The application must ensure that the EXTI RTC interrupt line is enabled.

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Set Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (const RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function description

Get the RTC TimeStamp value.

Parameters

- **hrtc:** RTC handle
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTCEx_TimeStampIRQHandler

Function name

void HAL_RTCEx_TimeStampIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle TimeStamp interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle TimeStamp polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_TimeStampEventCallback

Function name

void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)

Function description

TimeStamp callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock, uint32_t WakeUpAutoClr)

Function description

Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock
- **WakeUpAutoClr:** Wake up auto clear value (look at WUTOCLR in reference manual)
  - No effect if WakeUpAutoClr is set to zero
  - This feature is meaningful in case of Low power mode to avoid any RTC software execution after Wake Up.

Return values

- **HAL:** status

Notes

- The application must ensure that the EXTI RTC interrupt line is enabled.

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivate wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (const RTC_HandleTypeDef * hrtc)

Function description

Get wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Wake Up Timer interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Wake Up Timer callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Wake Up Timer Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

Set the Smooth calibration parameters.

Parameters

- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be one of the following values :
  - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
  - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
  - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
- **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
  - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
  - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
- **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetLowPowerCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetLowPowerCalib (RTC_HandleTypeDef * hrtc, uint32_t LowPowerCalib)

Function description

Select the low power Calibration mode.

Parameters

- **hrtc:** RTC handle
- **LowPowerCalib:** Low power Calibration mode. This parameter can be one of the following values :
  - RTC_LPCAL_SET: Low power mode.
  - RTC_LPCAL_RESET: High consumption mode.

Return values

- **HAL:** status

HAL_RTCEx_SetSynchroShift

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)

Function description

Configure the Synchronization Shift Control Settings.

Parameters

- **hrtc:** RTC handle
- **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values:
  - RTC_SHIFTADD1S_SET: Add one second to the clock calendar.
  - RTC_SHIFTADD1S_RESET: No effect.
- **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.

Return values

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle
- **CalibOutput:** Select the Calibration output Selection . This parameter can be one of the following values:
  - RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.
  - RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)

Function description

Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Disable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_MonotonicCounterIncrement

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterIncrement (const RTC_HandleTypeDef * hrtc, uint32_t Instance)

Function description

Increment Monotonic counter.

Parameters

- **hrtc:** RTC handle
- **Instance:** Monotonic counter Instance This parameter can be one of the following values :
  - RTC_MONOTONIC_COUNTER_1

Return values

- **HAL:** status

HAL_RTCEx_MonotonicCounterGet

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterGet (const RTC_HandleTypeDef * hrtc, uint32_t Instance, uint32_t * pValue)

Function description

Monotonic counter.

Parameters

- **hrtc:** RTC handle
- **Instance:** Monotonic counter Instance This parameter can be one of the following values :
  - RTC_MONOTONIC_COUNTER_1
- **pValue:** Pointer to the counter monotonic counter value

Return values

- **HAL:** status

HAL_RTCEx_SetSSRU_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSSRU_IT (RTC_HandleTypeDef * hrtc)

Function description

Set SSR Underflow detection with Interrupt.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The application must ensure that the EXTI RTC interrupt line is enabled.

HAL_RTCEx_DeactivateSSRU

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateSSRU (RTC_HandleTypeDef * hrtc)

Function description

Deactivate SSR Underflow.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SSRUIRQHandler

Function name

void HAL_RTCEx_SSRUIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle SSR underflow interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SSRUEventCallback

Function name

void HAL_RTCEx_SSRUEventCallback (RTC_HandleTypeDef * hrtc)

Function description

SSR underflow callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_AlarmBEventCallback

Function name

void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm B callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm B Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper (const RTC_HandleTypeDef * hrtc, const RTC_TamperTypeDef * sTamper)

Function description

Set Tamper.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveTampers (RTC_HandleTypeDef * hrtc, const RTC_ActiveTampersTypeDef * sAllTamper)

Function description

Set all active Tampers at the same time.

Parameters

- **hrtc:** RTC handle
- **sAllTamper:** Pointer to active Tamper Structure.

Return values

- **HAL:** status

Notes

- For interrupt mode, the application must ensure that the EXTI TAMP interrupt line is enabled.

HAL_RTCEx_SetActiveSeed

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveSeed (RTC_HandleTypeDef * hrtc, const uint32_t * pSeed)

Function description

Write a new seed.

Parameters

- **hrtc:** RTC handle
- **pSeed:** Pointer to active tamper seed values.

Return values

- **HAL:** status

HAL_RTCEx_SetTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (const RTC_HandleTypeDef * hrtc, const RTC_TamperTypeDef * sTamper)

Function description

Set Tamper in IT mode.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

Notes

- The application must ensure that the EXTI TAMP interrupt line is enabled.

HAL_RTCEx_DeactivateTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (const RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function description

Deactivate Tamper.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
  - RTC_TAMPER_3
  - RTC_TAMPER_4
  - RTC_TAMPER_5
  - RTC_TAMPER_6
  - RTC_TAMPER_7
  - RTC_TAMPER_8

Return values

- **HAL:** status

HAL_RTCEx_DeactivateActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateActiveTampers (const RTC_HandleTypeDef * hrtc)

Function description

Deactivate all Active Tampers at the same time.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_PollForTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamperEvent (const RTC_HandleTypeDef * hrtc, uint32_t Tamper, uint32_t Timeout)

Function description

Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
  - RTC_TAMPER_3
  - RTC_TAMPER_4
  - RTC_TAMPER_5
  - RTC_TAMPER_6
  - RTC_TAMPER_7
  - RTC_TAMPER_8
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper (const RTC_HandleTypeDef * hrtc, const RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper_IT (const RTC_HandleTypeDef * hrtc, const RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper in interrupt mode.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

Notes

- The application must ensure that the EXTI TAMP interrupt line is enabled.

HAL_RTCEx_DeactivateInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTamper (const RTC_HandleTypeDef * hrtc, uint32_t IntTamper)

Function description

Deactivate Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** Selected internal tamper event. This parameter can be any combination of existing internal tampers.

Return values

- **HAL:** status

HAL_RTCEx_PollForInternalTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForInternalTamperEvent (const RTC_HandleTypeDef * hrtc, uint32_t IntTamper, uint32_t Timeout)

Function description

Internal Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** selected tamper. This parameter can be any combination of existing internal tampers.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_LockBootHardwareKey

Function name

HAL_StatusTypeDef HAL_RTCEx_LockBootHardwareKey (const RTC_HandleTypeDef * hrtc)

Function description

Lock the Boot hardware Key.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The backup registers from TAMP_BKP0R to TAMP_BKP7R cannot be accessed neither in read nor in write (they are read as 0 and write ignore).

HAL_RTCEx_TamperIRQHandler

Function name

void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Tamper interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper1EventCallback

Function name

void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper2EventCallback

Function name

void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper3EventCallback

Function name

void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper4EventCallback

Function name

void HAL_RTCEx_Tamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 4 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper5EventCallback

Function name

void HAL_RTCEx_Tamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 5 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper6EventCallback

Function name

void HAL_RTCEx_Tamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 6 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper7EventCallback

Function name

void HAL_RTCEx_Tamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 7 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper8EventCallback

Function name

void HAL_RTCEx_Tamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 8 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper1EventCallback

Function name

void HAL_RTCEx_InternalTamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper2EventCallback

Function name

void HAL_RTCEx_InternalTamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper3EventCallback

Function name

void HAL_RTCEx_InternalTamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper4EventCallback

Function name

void HAL_RTCEx_InternalTamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 4 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper5EventCallback

Function name

void HAL_RTCEx_InternalTamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 5 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper6EventCallback

Function name

void HAL_RTCEx_InternalTamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 6 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper7EventCallback

Function name

void HAL_RTCEx_InternalTamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 7 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper8EventCallback

Function name

void HAL_RTCEx_InternalTamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 8 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper9EventCallback

Function name

void HAL_RTCEx_InternalTamper9EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 9 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper11EventCallback

Function name

void HAL_RTCEx_InternalTamper11EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 11 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper12EventCallback

Function name

void HAL_RTCEx_InternalTamper12EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 12 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper13EventCallback

Function name

void HAL_RTCEx_InternalTamper13EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 13 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper15EventCallback

Function name

void HAL_RTCEx_InternalTamper15EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 15 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (const RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

Write a data in a specified RTC Backup data register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be RTC_BKP_DRx where x can be from 0 to RTC_BACKUP_NB
- **Data:** Data to be written in the specified Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (const RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be RTC_BKP_DRx where x can be from 0 to RTC_BACKUP_NB

Return values

- **Read:** value

HAL_RTCEx_BKUPErase

Function name

void HAL_RTCEx_BKUPErase (const RTC_HandleTypeDef * hrtc)

Function description

Reset the RTC Backup data Registers and the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPBlock

Function name

void HAL_RTCEx_BKUPBlock (const RTC_HandleTypeDef * hrtc)

Function description

Block the access to the RTC Backup data Register and all the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPUnblock

Function name

void HAL_RTCEx_BKUPUnblock (const RTC_HandleTypeDef * hrtc)

Function description

Disable the Block to the access to the RTC Backup data Register and the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_ConfigEraseDeviceSecrets

Function name

void HAL_RTCEx_ConfigEraseDeviceSecrets (const RTC_HandleTypeDef * hrtc, uint32_t DeviceSecretConf)

Function description

Enable and Disable the erase of the configurable Device Secrets.

Parameters

- **hrtc:** RTC handle
- **DeviceSecretConf:** Specifies the configuration of the Device Secrets This parameter can be a combination of the following values:
  - TAMP_DEVICESECRETS_ERASE_NONE
  - TAMP_DEVICESECRETS_ERASE_BKPSRAM

Return values

- **None:**

Notes

- This API must be called before enabling the Tamper.

HAL_RTCEx_SecureModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeGet (const RTC_HandleTypeDef * hrtc, RTC_SecureStateTypeDef * secureState)

Function description

Get the security level of the RTC/TAMP/Backup registers.

Parameters

- **hrtc:** RTC handle
- **secureState:** Secure state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_SecureModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeSet (const RTC_HandleTypeDef * hrtc, const RTC_SecureStateTypeDef * secureState)

Function description

Set the security level of the RTC/TAMP/Backup registers.

Parameters

- **hrtc:** RTC handle
- **secureState:** Secure state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_PrivilegeModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeSet (const RTC_HandleTypeDef * hrtc, const RTC_PrivilegeStateTypeDef * privilegeState)

Function description

Set the privilege level of the RTC/TAMP/Backup registers.

Parameters

- **hrtc:** RTC handle
- **privilegeState:** Privilege state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_PrivilegeModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeGet (const RTC_HandleTypeDef * hrtc, RTC_PrivilegeStateTypeDef * privilegeState)

Function description

Get the privilege level of the RTC/TAMP/Backup registers.

Parameters

- **hrtc:** RTC handle
- **privilegeState:** Privilege state

Return values

- **HAL_StatusTypeDef:**

64.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

64.3.1 RTCEx

RTCEx

RTCEx Active_Tamper_Asynchronous_Prescaler clock Definitions

RTC_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

RTC_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

RTC_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

RTC_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

RTC_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

RTC_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

RTC_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

RTC_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

RTCEx_ActiveTamper_Enable Definitions

RTC_ATAMP_ENABLE

RTC_ATAMP_DISABLE

RTCEx_ActiveTamper_Filter Definitions

RTC_ATAMP_FILTER_ENABLE

RTC_ATAMP_FILTER_DISABLE

RTCEx_ActiveTamper_Interrupt Definitions

RTC_ATAMP_INTERRUPT_ENABLE

RTC_ATAMP_INTERRUPT_DISABLE

RTCEx Active Tamper selection Definition

RTC_ATAMP_1

Tamper 1

RTC_ATAMP_2

Tamper 2

RTC_ATAMP_3

Tamper 3

RTC_ATAMP_4

Tamper 4

RTC_ATAMP_5

Tamper 5

RTC_ATAMP_6

Tamper 6

RTC_ATAMP_7

Tamper 7

RTC_ATAMP_8

Tamper 8

Seed size Definitions

RTC_ATAMP_SEED_NB_UINT32

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Alarm Sub Seconds

RTC_ALARMSUBSECONDBIN_AUTOCLR_NO

The synchronous Binary counter(SS[31:0] in RTC_SSR) is free-running

RTC_ALARMSUBSECONDBIN_AUTOCLR_YES

The synchronous Binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF whenreaching RTC_ALRMABINR -> SS[31:0].

RTC Alarm Sub Seconds with binary or mix mode

RTC_ALARMSUBSECONDBINMASK_ALL

All Alarm SS fields are masked.There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDBINMASK_SS31_1

SS[31:1] are don't care in Alarm comparison. Only SS[0] is compared

RTC_ALARMSUBSECONDBINMASK_SS31_2

SS[31:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_3

SS[31:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_4

SS[31:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_5

SS[31:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_6

SS[31:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_7

SS[31:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_8

SS[31:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_9

SS[31:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_10

SS[31:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_11

SS[31:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_12

SS[31:12] are don't care in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_13

SS[31:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_14

SS[31:14] are don't care in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_15

SS[31:15] are don't care in Alarm comparison. Only SS[14:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_16

SS[31:16] are don't care in Alarm comparison. Only SS[15:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_17

SS[31:17] are don't care in Alarm comparison. Only SS[16:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_18

SS[31:18] are don't care in Alarm comparison. Only SS[17:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_19

SS[31:19] are don't care in Alarm comparison. Only SS[18:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_20

SS[31:20] are don't care in Alarm comparison. Only SS[19:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_21

SS[31:21] are don't care in Alarm comparison. Only SS[20:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_22

SS[31:22] are don't care in Alarm comparison. Only SS[21:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_23

SS[31:23] are don't care in Alarm comparison. Only SS[22:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_24

SS[31:24] are don't care in Alarm comparison. Only SS[23:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_25

SS[31:25] are don't care in Alarm comparison. Only SS[24:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_26

SS[31:26] are don't care in Alarm comparison. Only SS[25:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_27

SS[31:27] are don't care in Alarm comparison. Only SS[26:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_28

SS[31:28] are don't care in Alarm comparison. Only SS[27:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_29

SS[31:29] are don't care in Alarm comparison. Only SS[28:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_30

SS[31:30] are don't care in Alarm comparison. Only SS[29:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31

SS[31] is don't care in Alarm comparison. Only SS[30:0] are compared

RTC_ALARMSUBSECONDBINMASK_NONE

SS[31:0] are compared and must match to activate alarm

RTCEx Backup Registers Definition

RTC_BKP_NUMBER

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTCEx Privilege Backup register privilege zone Definition

RTC_PRIVILEGE_BKUP_ZONE_NONE

RTC_PRIVILEGE_BKUP_ZONE_1

RTC_PRIVILEGE_BKUP_ZONE_2

RTC_PRIVILEGE_BKUP_ZONE_ALL

If Binary mode is RTC_BINARY_MIX, the BCD calendar second is incremented

RTC_BINARY_MIX_BCDU_0

The 1s BCD calendar increment is generated each time SS[7:0] = 0

RTC_BINARY_MIX_BCDU_1

The 1s BCD calendar increment is generated each time SS[8:0] = 0

RTC_BINARY_MIX_BCDU_2

The 1s BCD calendar increment is generated each time SS[9:0] = 0

RTC_BINARY_MIX_BCDU_3

The 1s BCD calendar increment is generated each time SS[10:0] = 0

RTC_BINARY_MIX_BCDU_4

The 1s BCD calendar increment is generated each time SS[11:0] = 0

RTC_BINARY_MIX_BCDU_5

The 1s BCD calendar increment is generated each time SS[12:0] = 0

RTC_BINARY_MIX_BCDU_6

The 1s BCD calendar increment is generated each time SS[13:0] = 0

RTC_BINARY_MIX_BCDU_7

The 1s BCD calendar increment is generated each time SS[14:0] = 0

RTC Binary Mode (32-bit free-running counter configuration).

RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled)

RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

RTC_BINARY_MIX

Free running BCD calendar and Binary modes

RTC Calibration

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operations flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- None

RTCEx Calib Output selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_CLEAR_FLAG

Description:

- Clear the specified RTC pending flag.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_CLEAR_ITSF Clear Internal Time-stamp flag
  - RTC_CLEAR_TSOVF Clear Time-stamp overflow flag
  - RTC_CLEAR_TSF Clear Time-stamp flag
  - RTC_CLEAR_WUTF Clear Wakeup timer flag
  - RTC_CLEAR_ALRBF Clear Alarm B flag
  - RTC_CLEAR_ALRAF Clear Alarm A flag

Return value:

- None

__HAL_RTC_GET_FLAG

Description:

- Check whether the specified RTC flag is set or not.

Parameters:

- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_FLAG_RECALPF Recalibration pending Flag
  - RTC_FLAG_INITF Initialization flag
  - RTC_FLAG_RSF Registers synchronization flag
  - RTC_FLAG_INITS Initialization status flag
  - RTC_FLAG_SHPF Shift operation pending flag
  - RTC_FLAG_WUTWF Wakeup timer write flag
  - RTC_FLAG_ITSF Internal Time-stamp flag
  - RTC_FLAG_TSOVF Time-stamp overflow flag
  - RTC_FLAG_TSF Time-stamp flag
  - RTC_FLAG_WUTF Wakeup timer flag
  - RTC_FLAG_ALRBF Alarm B flag
  - RTC_FLAG_ALRAF Alarm A flag

Return value:

- None

RTCEx Flags

RTC_FLAG_TAMP_1

RTC_FLAG_TAMP_2

RTC_FLAG_TAMP_3

RTC_FLAG_TAMP_4

RTC_FLAG_TAMP_5

RTC_FLAG_TAMP_6

RTC_FLAG_TAMP_7

RTC_FLAG_TAMP_8

RTC_FLAG_TAMP_ALL

RTC_FLAG_INT_TAMP_1

RTC_FLAG_INT_TAMP_2

RTC_FLAG_INT_TAMP_3

RTC_FLAG_INT_TAMP_4

RTC_FLAG_INT_TAMP_5

RTC_FLAG_INT_TAMP_6

RTC_FLAG_INT_TAMP_7

RTC_FLAG_INT_TAMP_8

RTC_FLAG_INT_TAMP_9

RTC_FLAG_INT_TAMP_11

RTC_FLAG_INT_TAMP_12

RTC_FLAG_INT_TAMP_13

RTC_FLAG_INT_TAMP_15

RTC_FLAG_INT_TAMP_ALL

RTCEx Internal Tamper Interrupt

RTC_IT_INT_TAMP_1

Tamper 1 internal Interrupt

RTC_IT_INT_TAMP_2

Tamper 2 internal Interrupt

RTC_IT_INT_TAMP_3

Tamper 3 internal Interrupt

RTC_IT_INT_TAMP_4

Tamper 4 internal Interrupt

RTC_IT_INT_TAMP_5

Tamper 5 internal Interrupt

RTC_IT_INT_TAMP_6

Tamper 6 internal Interrupt

RTC_IT_INT_TAMP_7

Tamper 7 internal Interrupt

RTC_IT_INT_TAMP_8

Tamper 8 internal Interrupt

RTC_IT_INT_TAMP_9

Tamper 9 internal Interrupt

RTC_IT_INT_TAMP_11

Tamper 11 internal Interrupt

RTC_IT_INT_TAMP_12

Tamper 12 internal Interrupt

RTC_IT_INT_TAMP_13

Tamper 13 internal Interrupt

RTC_IT_INT_TAMP_15

Tamper 15 internal Interrupt

RTC_IT_INT_TAMP_ALL

RTCEx Internal Tamper Pins Definition

RTC_INT_TAMPER_1

RTC_INT_TAMPER_2

RTC_INT_TAMPER_3

RTC_INT_TAMPER_4

RTC_INT_TAMPER_5

RTC_INT_TAMPER_6

RTC_INT_TAMPER_7

RTC_INT_TAMPER_8

RTC_INT_TAMPER_9

RTC_INT_TAMPER_11

RTC_INT_TAMPER_12

RTC_INT_TAMPER_13

RTC_INT_TAMPER_15

RTC_INT_TAMPER_ALL

Private macros to check input parameters

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_TAMPERDETECTIONOUTPUT

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_LOW_POWER_CALIB

IS_RTC_TAMPER

IS_RTC_INTERNAL_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_ATAMPER_FILTER

IS_RTC_ATAMPER_OUTPUT_CHANGE_PERIOD

IS_RTC_ATAMPER_ASYNCPRES_RTCCLK

IS_RTC_BKP

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

IS_RTC_SECURE_FULL

IS_RTC_NONSECURE_FEATURES

IS_TAMP_SECURE_FULL

IS_TAMP_MONOTONIC_CNT_SECURE

IS_RTC_PRIVILEGE_FULL

IS_RTC_PRIVILEGE_FEATURES

IS_TAMP_PRIVILEGE_FULL

IS_TAMP_MONOTONIC_CNT_PRIVILEGE

IS_RTC_PRIVILEGE_BKUP_ZONE

IS_RTC_BINARY_MODE

IS_RTC_BINARY_MIX_BCDU

IS_RTC_ALARM_SUB_SECOND_BINARY_MASK

IS_RTC_ALARMSUBSECONDBIN_AUTOCLR

RTCEx Monotonic Counter Instance Definition

RTC_MONOTONIC_COUNTER_1

Monotonic counter 1

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC_OUTPUT_TAMPER

RTCEx Secure Features Definition

RTC_NONSECURE_FEATURE_NONE

RTC_NONSECURE_FEATURE_INIT

Initialization

RTC_NONSECURE_FEATURE_CAL

Calibration

RTC_NONSECURE_FEATURE_TS

Time stamp

RTC_NONSECURE_FEATURE_WUT

Wake up timer

RTC_NONSECURE_FEATURE_ALRA

Alarm A

RTC_NONSECURE_FEATURE_ALRB

Alarm B

RTC_NONSECURE_FEATURE_ALL

RTCEx Privilege Features Definition

RTC_PRIVILEGE_FEATURE_NONE

RTC_PRIVILEGE_FEATURE_INIT

Initialization

RTC_PRIVILEGE_FEATURE_CAL

Calibration

RTC_PRIVILEGE_FEATURE_TS

Time stamp

RTC_PRIVILEGE_FEATURE_WUT

Wake up timer

RTC_PRIVILEGE_FEATURE_ALRA

Alarm A

RTC_PRIVILEGE_FEATURE_ALRB

Alarm B

RTC_PRIVILEGE_FEATURE_ALL

RTCEx Privilege Full Definition

RTC_PRIVILEGE_FULL_YES

RTC_PRIVILEGE_FULL_NO

RTCEx Secure Definition

RTC_SECURE_FULL_YES

RTC full secure

RTC_SECURE_FULL_NO

RTC is not full secure, features can be unsecure. See RTCEx_RTC_NonSecure_Features

RTCEx Smooth calib Low Power Definitions

RTC_LPCAL_SET

Calibration window is 2exp20 ck_apre, which is the required configuration for ultra-low consumption mode.

RTC_LPCAL_RESET

Calibration window is 2exp20 RTCCLK, which is a high-consumption mode. This mode should be set only when less than 32s calibration window is required.

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTC SSR Underflow

__HAL_RTC_SSRU_ENABLE_IT

Description:

- Enable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be enabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_DISABLE_IT

Description:

- Disable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be disabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_GET_IT

Description:

- Check whether the specified RTC SSRU interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_GET_FLAG

Description:

- Get the selected RTC SSRU's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag is pending or not. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- None

__HAL_RTC_SSRU_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag to clear. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- None

RTCEx tamper

__HAL_RTC_TAMPER_ENABLE

Description:

- Enable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper source to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

Return value:

- None

__HAL_RTC_TAMPER_DISABLE

Description:

- Disable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper sources to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All Internal Tamper interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_4: Internal Tamper4 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt
  - RTC_IT_INT_TAMP_12: Internal Tamper12 interrupt
  - RTC_IT_INT_TAMP_13: Internal Tamper13 interrupt
  - RTC_IT_INT_TAMP_15: Internal Tamper15 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All internal tampers interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_4: Internal Tamper4 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt
  - RTC_IT_INT_TAMP_12: Internal Tamper12 interrupt
  - RTC_IT_INT_TAMP_13: Internal Tamper13 interrupt
  - RTC_IT_INT_TAMP_15: Internal Tamper15 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tampers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_4: Internal Tamper4 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag
  - RTC_FLAG_INT_TAMP_12: Internal Tamper12 flag
  - RTC_FLAG_INT_TAMP_13: Internal Tamper13 flag
  - RTC_FLAG_INT_TAMP_15: Internal Tamper15 flag

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_ALL: All Internal Tamper flags
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_4: Internal Tamper4 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag
  - RTC_FLAG_INT_TAMP_12: Internal Tamper12 flag
  - RTC_FLAG_INT_TAMP_13: Internal Tamper13 flag
  - RTC_FLAG_INT_TAMP_15: Internal Tamper15 flag

Return value:

- None

RTCEx Tamper detection output Definitions

RTC_TAMPERDETECTIONOUTPUT_DISABLE

Tamper detection output disable on TAMPALRM

RTC_TAMPERDETECTIONOUTPUT_ENABLE

Tamper detection output enable on TAMPALRM

RTCEx Tamper EraseBackUp

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter

RTC_TAMPERFILTER_DISABLE

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

RTCEx Tamper Interrupt

RTC_IT_TAMP_1

Tamper 1 Interrupt

RTC_IT_TAMP_2

Tamper 2 Interrupt

RTC_IT_TAMP_3

Tamper 3 Interrupt

RTC_IT_TAMP_4

Tamper 4 Interrupt

RTC_IT_TAMP_5

Tamper 5 Interrupt

RTC_IT_TAMP_6

Tamper 6 Interrupt

RTC_IT_TAMP_7

Tamper 7 Interrupt

RTC_IT_TAMP_8

Tamper 8 Interrupt

RTC_IT_TAMP_ALL

RTCEx Tampers maskable number

RTC_TAMPER_MASKABLE_NB

RTCEx Tamper MaskFlag

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC_TAMPER_4

RTC_TAMPER_5

RTC_TAMPER_6

RTC_TAMPER_7

RTC_TAMPER_8

RTC_TAMPER_ALL

RTCEx Tamper Pin Precharge Duration

RTC_TAMPERPRECHARGEDURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP

RTC_TAMPER_PULLUP_ENABLE

Tamper pins are pre-charged before sampling

RTC_TAMPER_PULLUP_DISABLE

Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTCEx Tamper TimeStamp On Tamper Detection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE

TimeStamp on Tamper Detection event saved

RTCEx Tamper Trigger

RTC_TAMPERTRIGGER_RISINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_FALLINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_LOWLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_HIGHLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTCEx TAMP Device Secrets Erase Configuration Definition

TAMP_DEVICESECRETS_ERASE_NONE

TAMP_DEVICESECRETS_ERASE_BKPSRAM

Backup SRAM

RTCEx TAMP Monotonic Counter Privilege Definition

TAMP_MONOTONIC_CNT_PRIVILEGE_YES

TAMP_MONOTONIC_CNT_PRIVILEGE_NO

RTCEx TAMP Monotonic Counter Secure Definition

TAMP_MONOTONIC_CNT_SECURE_YES

TAMPER Monotonic Counter secure

TAMP_MONOTONIC_CNT_SECURE_NO

TAMPER Monotonic Counter is not secure

RTCEx TAMP security Definition

TAMP_PRIVILEGE_FULL_YES

TAMP_PRIVILEGE_FULL_NO

RTCEx TAMP Secure Definition

TAMP_SECURE_FULL_YES

TAMPER full secure

TAMP_SECURE_FULL_NO

TAMPER is not secure

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag to clear. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC Internal Time Stamps flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Internal Time Stamps pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

__HAL_RTC_TAMPTS_ENABLE

Description:

- Enable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPTS_DISABLE

Description:

- Disable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_ENABLE

Description:

- Enable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPOE_DISABLE

Description:

- Disable the RTC Tamper detection output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

Return value:

- None

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timers pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - RTC_FLAG_WUTF

Return value:

- None

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

65 HAL SAI Generic Driver

65.1 SAI Firmware driver registers structures

65.1.1 SAI_PdmInitTypeDef

SAI_PdmInitTypeDef is defined in the stm32h5xx_hal_sai.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

65.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32h5xx_hal_sai.h

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t MckOutput***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***
  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***
  Note:
  – If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::MckOutput***
  Specifies whether master clock output will be generated or not. This parameter can be a value of ***SAI_Block_MckOutput***
- ***uint32_t SAI_InitTypeDef::OutputDrive***
  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***
  Note:
  – This value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***
  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***
  Note:
  – If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values from 8 to 256.
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***
  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***
  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***
  Specifies the master clock divider. This parameter must be a number between Min_Data = 0 and Max_Data = 63.
  Note:
  – This parameter is used only if AudioFrequency is set to SAI_AUDIO_FREQUENCY_MCKDIV otherwise it is internally computed.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***
  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***
  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***
  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***
  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***
  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***
  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***
  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***
- ***uint32_t SAI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***
- ***uint32_t SAI_InitTypeDef::ClockStrobing***
  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

65.1.3 SAI_FrameInitTypeDef

SAI_FrameInitTypeDef is defined in the stm32h5xx_hal_sai.h

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FSOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***
  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
  Note:
  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***
  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

65.1.4 SAI_SlotInitTypeDef

SAI_SlotInitTypeDef is defined in the stm32h5xx_hal_sai.h

Data Fields

- ***uint32_t FirstBitOffset***
- ***uint32_t SlotSize***
- ***uint32_t SlotNumber***
- ***uint32_t SlotActive***

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***
  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24
- ***uint32_t SAI_SlotInitTypeDef::SlotSize***
  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***
- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***
  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t SAI_SlotInitTypeDef::SlotActive***
  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

65.1.5 __SAI_HandleTypeDef

__SAI_HandleTypeDef is defined in the stm32h5xx_hal_sai.h

Data Fields

- ***SAI_Block_TypeDef * Instance***
- ***SAI_InitTypeDef Init***
- ***SAI_FrameInitTypeDef FrameInit***
- ***SAI_SlotInitTypeDef SlotInit***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***

- *uint16_t XferCount*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *SAIcallback mutecallback*
- *void(* InterruptServiceRoutine*
- *HAL_LockTypeDef Lock*
- *__IO HAL_SAI_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *void(* RxCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* TxCpltCallback*
- *void(* TxHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code
- ***void(* __SAI_HandleTypeDef::RxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive complete callback
- ***void(* __SAI_HandleTypeDef::RxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive half complete callback
- ***void(* __SAI_HandleTypeDef::TxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit complete callback
- ***void(* __SAI_HandleTypeDef::TxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit half complete callback
- ***void(* __SAI_HandleTypeDef::ErrorCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI error callback

- ***void(* __SAI_HandleTypeDef::MspInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP init callback
- ***void(* __SAI_HandleTypeDef::MspDeInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP de-init callback

65.2 SAI Firmware driver API description

The following section lists the various functions of the SAI library.

65.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      - Enable the clock for the SAI GPIOs.
      - Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      - Configure the SAI interrupt priority.
      - Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx stream.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream.
      - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

- *PLLSAI1CLK output is configured or*
- *PLLSAI2CLK output is configured or*
- *PLLSAI3CLK output is configured or*
- *External clock source is configured after setting correctly the define constant EXTERNAL_SAI1_CLOCK_VALUE or EXTERNAL_SAI2_CLOCK_VALUE in the stm32h5xx_hal_conf.h file.*

Note: In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

Note: In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

Note: It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

Callback registration

The compilation define USE_HAL_SAI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_SAI_RegisterCallback() to register a user callback.

Function HAL_SAI_RegisterCallback() allows to register following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SAI_UnRegisterCallback() to reset a callback to the default weak function. HAL_SAI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

By default, after the HAL_SAI_Init and if the state is HAL_SAI_STATE_RESET all callbacks are reset to the corresponding legacy weak functions: examples HAL_SAI_RxCpltCallback(), HAL_SAI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_SAI_Init and HAL_SAI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SAI_Init and HAL_SAI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SAI_RegisterCallback before calling HAL_SAI_DeInit or HAL_SAI_Init function.

When the compilation define USE_HAL_SAI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

65.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL_SAI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SAI_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size
  - MCLK Output
  - Audio frequency
  - FIFO Threshold
  - Frame Config
  - Slot Config
  - PDM Config
- Call the function HAL_SAI_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- ***HAL_SAI_InitProtocol()***
- ***HAL_SAI_Init()***
- ***HAL_SAI_DeInit()***

- *HAL_SAI_MspInit()*
- *HAL_SAI_MspDeInit()*
- *HAL_SAI_RegisterCallback()*
- *HAL_SAI_UnRegisterCallback()*

65.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - HAL_SAI_Transmit()
  - HAL_SAI_Receive()
- Non Blocking mode functions with Interrupt are :
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
- Non Blocking mode functions with DMA are :
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()

This section contains the following APIs:

- *HAL_SAI_Transmit()*
- *HAL_SAI_Receive()*
- *HAL_SAI_Transmit_IT()*
- *HAL_SAI_Receive_IT()*
- *HAL_SAI_DMAPause()*
- *HAL_SAI_DMAResume()*
- *HAL_SAI_DMAStop()*
- *HAL_SAI_Abort()*
- *HAL_SAI_Transmit_DMA()*
- *HAL_SAI_Receive_DMA()*
- *HAL_SAI_EnableTxMuteMode()*
- *HAL_SAI_DisableTxMuteMode()*
- *HAL_SAI_EnableRxMuteMode()*
- *HAL_SAI_DisableRxMuteMode()*
- *HAL_SAI_IRQHandler()*
- *HAL_SAI_TxCpltCallback()*
- *HAL_SAI_TxHalfCpltCallback()*
- *HAL_SAI_RxCpltCallback()*
- *HAL_SAI_RxHalfCpltCallback()*
- *HAL_SAI_ErrorCallback()*

65.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_SAI_GetState()*
- *HAL_SAI_GetError()*

65.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_RegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID, pSAI_CallbackTypeDef pCallback)

Function description

Register a user SAI callback to be used instead of the weak predefined callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SAI_ERROR_CB_ID error callback ID.
  - HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_SAI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_UnRegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID)

Function description

Unregister a user SAI callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SAI_ERROR_CB_ID error callback ID.
  - HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

65.3 SAI Firmware driver defines

The following section lists the various define and macros of the module.

65.3.1 SAI

SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock Output

SAI_MCK_OUTPUT_DISABLE

SAI_MCK_OUTPUT_ENABLE

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI Error Code

HAL_SAI_ERROR_NONE

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

HAL_SAI_ERROR_INVALID_CALLBACK

Invalid callback error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

Description:

- Disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.
  - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
  - SAI_FLAG_FREQ: FIFO request flag.
  - SAI_FLAG_CNRDY: Codec not ready flag.
  - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
  - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

Description:

- Enable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_DISABLE

Description:

- Disable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

66 HAL SAI Extension Driver

66.1 SAIEx Firmware driver registers structures

66.1.1 SAIEx_PdmMicDelayParamTypeDef

SAIEx_PdmMicDelayParamTypeDef is defined in the stm32h5xx_hal_sai_ex.h

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***
  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***
  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***
  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

66.2 SAIEx Firmware driver API description

The following section lists the various functions of the SAIEx library.

66.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- ***HAL_SAIEx_ConfigPdmMicDelay()***

66.2.2 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name

HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (const SAI_HandleTypeDef * hsai, const SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)

Function description

Configure PDM microphone delays.

Parameters

- **hsai:** SAI handle.
- **pdmMicDelay:** Microphone delays configuration.

Return values

- **HAL:** status

67 HAL SDRAM Generic Driver

67.1 SDRAM Firmware driver registers structures

67.1.1 __SDRAM_HandleTypeDef

__SDRAM_HandleTypeDef is defined in the stm32h5xx_hal_sdram.h

Data Fields

- ***FMC_SDRAM_TypeDef * Instance***
- ***FMC_SDRAM_InitTypeDef Init***
- ***__IO HAL_SDRAM_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * hdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* RefreshErrorCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_SDRAM_TypeDef* __SDRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_SDRAM_InitTypeDef __SDRAM_HandleTypeDef::Init***
  SDRAM device configuration parameters
- ***__IO HAL_SDRAM_StateTypeDef __SDRAM_HandleTypeDef::State***
  SDRAM access state
- ***HAL_LockTypeDef __SDRAM_HandleTypeDef::Lock***
  SDRAM locking object
- ***DMA_HandleTypeDef* __SDRAM_HandleTypeDef::hdma***
  Pointer DMA handler
- ***void(* __SDRAM_HandleTypeDef::MspInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp Init callback
- ***void(* __SDRAM_HandleTypeDef::MspDeInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp DeInit callback
- ***void(* __SDRAM_HandleTypeDef::RefreshErrorCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Refresh Error callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferCpltCallback)(DMA_HandleTypeDef *hdma)***
  SDRAM DMA Xfer Complete callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferErrorCallback)(DMA_HandleTypeDef *hdma)***
  SDRAM DMA Xfer Error callback

67.2 SDRAM Firmware driver API description

The following section lists the various functions of the SDRAM library.

67.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM_HandleTypeDef handle structure, for example: SDRAM_HandleTypeDef hsdram
   – Fill the SDRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SDRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device

2. Declare a FMC_SDRAM_TimingTypeDef structure; for example: FMC_SDRAM_TimingTypeDef Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function HAL_SDRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SDRAM_MspInit()
   b. Control register configuration using the FMC SDRAM interface function FMC_SDRAM_Init()
   c. Timing register configuration using the FMC SDRAM interface function FMC_SDRAM_Timing_Init()
   d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SDRAM_Read()/HAL_SDRAM_Write() for polling read/write access
   – HAL_SDRAM_Read_DMA()/HAL_SDRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs HAL_SDRAM_WriteOperation_Enable()/ HAL_SDRAM_WriteOperation_Disable() to respectively enable/disable the SDRAM write operation or the function HAL_SDRAM_SendCommand() to send a specified command to the SDRAM device. The command to be sent must be configured with the FMC_SDRAM_CommandTypeDef structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function HAL_SDRAM_GetState()

Callback registration

The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SDRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SDRAM_UnRegisterCallback() to reset a callback to the default weak function. It allows to reset following callbacks:
- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SDRAM_Init and if the state is HAL_SDRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_SDRAM_Init and HAL_SDRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SDRAM_Init and HAL_SDRAM_DeInit keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SDRAM_RegisterCallback before calling HAL_SDRAM_DeInit or HAL_SDRAM_Init function. When The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

67.2.2 SDRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

This section contains the following APIs:

- ***HAL_SDRAM_Init()***
- ***HAL_SDRAM_DeInit()***
- ***HAL_SDRAM_MspInit()***
- ***HAL_SDRAM_MspDeInit()***
- ***HAL_SDRAM_IRQHandler()***
- ***HAL_SDRAM_RefreshErrorCallback()***
- ***HAL_SDRAM_DMA_XferCpltCallback()***
- ***HAL_SDRAM_DMA_XferErrorCallback()***

67.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

This section contains the following APIs:

- *HAL_SDRAM_Read_8b()*
- *HAL_SDRAM_Write_8b()*
- *HAL_SDRAM_Read_16b()*
- *HAL_SDRAM_Write_16b()*
- *HAL_SDRAM_Read_32b()*
- *HAL_SDRAM_Write_32b()*
- *HAL_SDRAM_Read_DMA()*
- *HAL_SDRAM_Write_DMA()*
- *HAL_SDRAM_RegisterCallback()*
- *HAL_SDRAM_UnRegisterCallback()*
- *HAL_SDRAM_RegisterDmaCallback()*

67.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

This section contains the following APIs:

- *HAL_SDRAM_WriteProtection_Enable()*
- *HAL_SDRAM_WriteProtection_Disable()*
- *HAL_SDRAM_SendCommand()*
- *HAL_SDRAM_ProgramRefreshRate()*
- *HAL_SDRAM_SetAutoRefreshNumber()*
- *HAL_SDRAM_GetModeStatus()*

67.2.5 SDRAM State functions

This subsection permits to get in run-time the status of the SDRAM controller and the data flow.

This section contains the following APIs:

- *HAL_SDRAM_GetState()*

67.2.6 Detailed description of functions

HAL_SDRAM_Init

Function name

HAL_StatusTypeDef HAL_SDRAM_Init (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_TimingTypeDef * Timing)

Function description

Performs the SDRAM device initialization sequence.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Timing:** Pointer to SDRAM control timing structure

Return values

- **HAL:** status

HAL_SDRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SDRAM_DeInit (SDRAM_HandleTypeDef * hsdram)

Function description

Perform the SDRAM device initialization sequence.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_MspInit

Function name

void HAL_SDRAM_MspInit (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM MSP Init.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_MspDeInit

Function name

void HAL_SDRAM_MspDeInit (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM MSP DeInit.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_IRQHandler

Function name

void HAL_SDRAM_IRQHandler (SDRAM_HandleTypeDef * hsdram)

Function description

This function handles SDRAM refresh error interrupt request.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_RefreshErrorCallback

Function name

void HAL_SDRAM_RefreshErrorCallback (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM Refresh error callback.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_DMA_XferCpltCallback

Function name

void HAL_SDRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

HAL_SDRAM_DMA_XferErrorCallback

Function name

void HAL_SDRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete error callback.

Parameters

- **hdma:** DMA handle

Return values

- **None:**

HAL_SDRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SDRAM memory using DMA transfer.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SDRAM memory using DMA transfer.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_CallbackTypeDef pCallback)

Function description

Register a User SDRAM Callback To be used instead of the weak predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SDRAM_MSP_INIT_CB_ID SDRAM MspInit callback ID
  - HAL_SDRAM_MSP_DEINIT_CB_ID SDRAM MspDeInit callback ID
  - HAL_SDRAM_REFRESH_ERR_CB_ID SDRAM Refresh Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SDRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_UnRegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId)

Function description

Unregister a User SDRAM Callback SDRAM Callback is redirected to the weak predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SDRAM_MSP_INIT_CB_ID SDRAM MspInit callback ID
  - HAL_SDRAM_MSP_DEINIT_CB_ID SDRAM MspDeInit callback ID
  - HAL_SDRAM_REFRESH_ERR_CB_ID SDRAM Refresh Error callback ID
  - HAL_SDRAM_DMA_XFER_CPLT_CB_ID SDRAM DMA Xfer Complete callback ID
  - HAL_SDRAM_DMA_XFER_ERR_CB_ID SDRAM DMA Xfer Error callback ID

Return values

- **status:**

HAL_SDRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterDmaCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_DmaCallbackTypeDef pCallback)

Function description

Register a User SDRAM Callback for DMA transfers To be used instead of the weak predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SDRAM_DMA_XFER_CPLT_CB_ID SDRAM DMA Xfer Complete callback ID
  - HAL_SDRAM_DMA_XFER_ERR_CB_ID SDRAM DMA Xfer Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SDRAM_WriteProtection_Enable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (SDRAM_HandleTypeDef * hsdram)

Function description

Enables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_WriteProtection_Disable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (SDRAM_HandleTypeDef * hsdram)

Function description

Disables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_SendCommand

Function name

HAL_StatusTypeDef HAL_SDRAM_SendCommand (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_CommandTypeDef * Command, uint32_t Timeout)

Function description

Sends Command to the SDRAM bank.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Command:** SDRAM command structure
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SDRAM_ProgramRefreshRate

Function name

HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (SDRAM_HandleTypeDef * hsdram, uint32_t RefreshRate)

Function description

Programs the SDRAM Memory Refresh rate.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **RefreshRate:** The SDRAM refresh rate value

Return values

- **HAL:** status

HAL_SDRAM_SetAutoRefreshNumber

Function name

HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (SDRAM_HandleTypeDef * hsdram, uint32_t AutoRefreshNumber)

Function description

Sets the Number of consecutive SDRAM Memory auto Refresh commands.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **AutoRefreshNumber:** The SDRAM auto Refresh number

Return values

- **HAL:** status

HAL_SDRAM_GetModeStatus

Function name

uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM memory current mode.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **The:** SDRAM memory mode.

HAL_SDRAM_GetState

Function name

HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM state.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** state

67.3 SDRAM Firmware driver defines

The following section lists the various define and macros of the module.

67.3.1 SDRAM

SDRAM

SDRAM Exported Macros

__HAL_SDRAM_RESET_HANDLE_STATE

Description:

- Reset SDRAM handle state.

Parameters:

- __HANDLE__: specifies the SDRAM handle.

Return value:

- None

68 HAL SD Generic Driver

68.1 SD Firmware driver registers structures

68.1.1 HAL_SD_CardInfoTypeDef

HAL_SD_CardInfoTypeDef is defined in the stm32h5xx_hal_sd.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t CardVersion***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***
- ***uint32_t CardSpeed***

Field Documentation

- ***uint32_t HAL_SD_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_SD_CardInfoTypeDef::CardVersion***
  Specifies the card version
- ***uint32_t HAL_SD_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::CardSpeed***
  Specifies the card Speed

68.1.2 __SD_HandleTypeDef

__SD_HandleTypeDef is defined in the stm32h5xx_hal_sd.h

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_SD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_SD_CardInfoTypeDef SdCard***

- *uint32_t CSD*
- *uint32_t CID*
- *void(* TxCpltCallback*
- *void(* RxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* Read_DMALnkLstBufCpltCallback*
- *void(* Write_DMALnkLstBufCpltCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SD_TypeDef* __SD_HandleTypeDef::Instance***
  SD registers base address
- ***SD_InitTypeDef __SD_HandleTypeDef::Init***
  SD required parameters
- ***HAL_LockTypeDef __SD_HandleTypeDef::Lock***
  SD locking object
- ***const uint8_t* __SD_HandleTypeDef::pTxBuffPtr***
  Pointer to SD Tx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::TxXferSize***
  SD Tx Transfer size
- ***uint8_t* __SD_HandleTypeDef::pRxBuffPtr***
  Pointer to SD Rx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::RxXferSize***
  SD Rx Transfer size
- ***__IO uint32_t __SD_HandleTypeDef::Context***
  SD transfer context
- ***__IO HAL_SD_StateTypeDef __SD_HandleTypeDef::State***
  SD card State
- ***__IO uint32_t __SD_HandleTypeDef::ErrorCode***
  SD Card Error codes
- ***HAL_SD_CardInfoTypeDef __SD_HandleTypeDef::SdCard***
  SD Card information
- ***uint32_t __SD_HandleTypeDef::CSD[4]***
  SD card specific data table
- ***uint32_t __SD_HandleTypeDef::CID[4]***
  SD card identification number table
- ***void(* __SD_HandleTypeDef::TxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::RxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::ErrorCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::AbortCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspInitCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspDeInitCallback)(struct __SD_HandleTypeDef *hsd)***

68.1.3 HAL_SD_CardCSDTypeDef

HAL_SD_CardCSDTypeDef is defined in the stm32h5xx_hal_sd.h

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***
- ***__IO uint8_t TAAC***

- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_SD_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved4***
  Always 1

68.1.4 HAL_SD_CardCIDTypeDef

HAL_SD_CardCIDTypeDef is defined in the stm32h5xx_hal_sd.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2***
  Always 1

68.1.5 HAL_SD_CardStatusTypeDef

HAL_SD_CardStatusTypeDef is defined in the stm32h5xx_hal_sd.h

Data Fields

- ***__IO uint8_t DataBusWidth***
- ***__IO uint8_t SecuredMode***
- ***__IO uint16_t CardType***
- ***__IO uint32_t ProtectedAreaSize***
- ***__IO uint8_t SpeedClass***
- ***__IO uint8_t PerformanceMove***
- ***__IO uint8_t AllocationUnitSize***
- ***__IO uint16_t EraseSize***
- ***__IO uint8_t EraseTimeout***
- ***__IO uint8_t EraseOffset***
- ***__IO uint8_t UhsSpeedGrade***
- ***__IO uint8_t UhsAllocationUnitSize***
- ***__IO uint8_t VideoSpeedClass***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::DataBusWidth***
  Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SecuredMode***
  Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::CardType***
  Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypeDef::ProtectedAreaSize***
  Carries information about the capacity of protected area
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SpeedClass***
  Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::PerformanceMove***
  Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::AllocationUnitSize***
  Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::EraseSize***
  Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseTimeout***
  Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseOffset***
  Carries information about the erase offset
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsSpeedGrade***
  Carries information about the speed grade of UHS card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsAllocationUnitSize***
  Carries information about the UHS card's allocation unit size
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::VideoSpeedClass***
  Carries information about the Video Speed Class of UHS card

68.2 SD Firmware driver API description

The following section lists the various functions of the SD library.

68.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_SD_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implementing the HAL_SD_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for SD card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_SD_ReadBlocks_IT() and HAL_SD_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the HAL_SD_Init() function. It Initializes SDMMC Peripheral(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the SD card. The API used is HAL_SD_InitCard(). This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (HAL_SD_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL_SD_ReadBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL_SD_WriteBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL_SD_GetCardStatus().

SD card information

- To get SD card information, you can use the function HAL_SD_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

SD card CSD register

SD card CID register

SD HAL driver macros list

Note: You can refer to the SD HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_SD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SD_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : SD MspInit.
- MspDeInitCallback : SD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. For specific callbacks TransceiverCallback use dedicated register callbacks: respectively HAL_SD_RegisterTransceiverCallback(). Use function HAL_SD_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : SD MspInit.

- MspDeInitCallback : SD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. For specific callbacks TransceiverCallback use dedicated unregister callbacks: respectively HAL_SD_UnRegisterTransceiverCallback(). By default, after the HAL_SD_Init and if the state is HAL_SD_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SD_Init and HAL_SD_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SD_Init and HAL_SD_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/ unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SD_RegisterCallback before calling HAL_SD_DeInit or HAL_SD_Init function. When The compilation define USE_HAL_SD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

68.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- ***HAL_SD_Init()***
- ***HAL_SD_InitCard()***
- ***HAL_SD_DeInit()***
- ***HAL_SD_MspInit()***
- ***HAL_SD_MspDeInit()***

68.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- ***HAL_SD_ReadBlocks()***
- ***HAL_SD_WriteBlocks()***
- ***HAL_SD_ReadBlocks_IT()***
- ***HAL_SD_WriteBlocks_IT()***
- ***HAL_SD_ReadBlocks_DMA()***
- ***HAL_SD_WriteBlocks_DMA()***
- ***HAL_SD_Erase()***
- ***HAL_SD_IRQHandler()***
- ***HAL_SD_GetState()***
- ***HAL_SD_GetError()***
- ***HAL_SD_TxCpltCallback()***
- ***HAL_SD_RxCpltCallback()***
- ***HAL_SD_ErrorCallback()***
- ***HAL_SD_AbortCallback()***
- ***HAL_SD_RegisterCallback()***
- ***HAL_SD_UnRegisterCallback()***

68.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- ***HAL_SD_GetCardCID()***
- ***HAL_SD_GetCardCSD()***
- ***HAL_SD_GetCardStatus()***
- ***HAL_SD_GetCardInfo()***

- *HAL_SD_ConfigWideBusOperation()*
- *HAL_SD_ConfigSpeedBusOperation()*
- *HAL_SD_GetCardState()*
- *HAL_SD_Abort()*
- *HAL_SD_Abort_IT()*

68.2.5 Detailed description of functions

HAL_SD_Init

Function name

HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)

Function description

Initializes the SD according to the specified parameters in the SD_HandleTypeDef and create the associated handle.

Parameters

- **hsd:** Pointer to the SD handle

Return values

- **HAL:** status

HAL_SD_InitCard

Function name

HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)

Function description

Initializes the SD Card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

Notes

- This function initializes the SD card. It could be used when a card re-initialization is needed.

HAL_SD_DeInit

Function name

HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)

Function description

De-Initializes the SD card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

HAL_SD_MspInit

Function name

void HAL_SD_MspInit (SD_HandleTypeDef * hsd)

Function description

Initializes the SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_MspDeInit

Function name

void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)

Function description

De-Initialize SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of SD blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of SD blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_Erase

Function name

HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given SD card.

Parameters

- **hsd:** Pointer to SD handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

HAL_SD_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Tx interrupt event.

HAL_SD_IRQHandler

Function name

void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)

Function description

This function handles SD card interrupt request.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_TxCpltCallback

Function name

void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Tx Transfer completed callbacks.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_RxCpltCallback

Function name

void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Rx Transfer completed callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_ErrorCallback

Function name

void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)

Function description

SD error callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_AbortCallback

Function name

void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)

Function description

SD Abort callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_RegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID, pSD_CallbackTypeDef pCallback)

Function description

Register a User SD Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  - HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  - HAL_SD_ERROR_CB_ID SD Error Callback ID
  - HAL_SD_ABORT_CB_ID SD Abort Callback ID
  - HAL_SD_READ_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Rx Linked List Node buffer Callback ID
  - HAL_SD_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Tx Linked List Node buffer Callback ID
  - HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  - HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_SD_RegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_UnRegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID)

Function description

Unregister a User SD Callback SD Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  - HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  - HAL_SD_ERROR_CB_ID SD Error Callback ID
  - HAL_SD_ABORT_CB_ID SD Abort Callback ID
  - HAL_SD_READ_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Rx Linked List Node buffer Callback ID
  - HAL_SD_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Tx Linked List Node buffer Callback ID
  - HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  - HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_SD_UnRegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hsd:** Pointer to SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:
  - SDMMC_BUS_WIDE_8B: 8-bit data transfer
  - SDMMC_BUS_WIDE_4B: 4-bit data transfer
  - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_SD_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigSpeedBusOperation (SD_HandleTypeDef * hsd, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hsd:** Pointer to the SD handle
- **SpeedMode:** Specifies the SD card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed/SDR12 mode
  - SDMMC_SPEED_MODE_HIGH: High Speed/SDR25 mode
  - SDMMC_SPEED_MODE_ULTRA: Ultra high speed mode

Return values

- **HAL:** status

HAL_SD_GetCardState

Function name

HAL_SD_CardStateTypeDef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)

Function description

Gets the current sd card data state.

Parameters

- **hsd:** pointer to SD handle

Return values

- **Card:** state

HAL_SD_GetCardCID

Function name

HAL_StatusTypeDef HAL_SD_GetCardCID (SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hsd:** Pointer to SD handle
- **pCID:** Pointer to a HAL_SD_CardCIDTypeDef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_SD_GetCardCSD

Function name

HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hsd:** Pointer to SD handle
- **pCSD:** Pointer to a HAL_SD_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_SD_GetCardStatus

Function name

HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)

Function description

Gets the SD status info.

Parameters

- **hsd:** Pointer to SD handle
- **pStatus:** Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetCardInfo

Function name

HAL_StatusTypeDef HAL_SD_GetCardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)

Function description

Gets the SD card info.

Parameters

- **hsd:** Pointer to SD handle
- **pCardInfo:** Pointer to the HAL_SD_CardInfoTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetState

Function name

HAL_SD_StateTypeDef HAL_SD_GetState (const SD_HandleTypeDef * hsd)

Function description

return the SD state

Parameters

- **hsd:** Pointer to sd handle

Return values

- **HAL:** state

HAL_SD_GetError

Function name

uint32_t HAL_SD_GetError (const SD_HandleTypeDef * hsd)

Function description

Return the SD error code.

Parameters

- **hsd:** : Pointer to a SD_HandleTypeDef structure that contains the configuration information.

Return values

- **SD:** Error Code

HAL_SD_Abort

Function name

HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

HAL_SD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD (IT mode).

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

68.3 SD Firmware driver defines

The following section lists the various define and macros of the module.

68.3.1 SD

SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE

No error

HAL_SD_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_SD_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SD_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_SD_ERROR_DATA_TIMEOUT

Data timeout

HAL_SD_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SD_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SD_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_SD_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_SD_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_SD_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_SD_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_SD_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_SD_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_SD_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_SD_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_SD_ERROR_CC_ERR

Internal card controller error

HAL_SD_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_SD_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_SD_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_SD_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_SD_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_SD_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_SD_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_SD_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_SD_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_SD_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_SD_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_SD_ERROR_PARAM

the used parameter is not valid

HAL_SD_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_SD_ERROR_BUSY

Error when transfer process is busy

HAL_SD_ERROR_DMA

Error while DMA transfer

HAL_SD_ERROR_TIMEOUT

Timeout error

HAL_SD_ERROR_INVALID_CALLBACK

Invalid callback error

SD context enumeration

SD_CONTEXT_NONE

None

SD_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

SD_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

SD_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

SD_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

SD_CONTEXT_IT

Process in Interrupt mode

SD_CONTEXT_DMA

Process in DMA mode

SD Supported Memory Cards

CARD_NORMAL_SPEED

Normal Speed Card <12.5Mo/s , Spec Version 1.01

CARD_HIGH_SPEED

High Speed Card <25Mo/s , Spec version 2.00

CARD_ULTRA_HIGH_SPEED

UHS-I SD Card <50Mo/s for SDR50, DDR5 Cards and <104Mo/s for SDR104, Spec version 3.01

CARD_SDSC

SD Standard Capacity <2Go

CARD_SDHC_SDXC

SD High Capacity <32Go, SD Extended Capacity <2To

CARD_SECURED

SD Supported Version

CARD_V1_X

CARD_V2_X

SD Exported Constants

BLOCKSIZE

Block size is 512 bytes

SD Exported Macros

__HAL_SD_RESET_HANDLE_STATE

Description:

- Reset SD handle state.

Parameters:

- __HANDLE__: SD Handle.

Return value:

- None

__HAL_SD_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_GET_FLAG

Description:

- Check whether the specified SD flag is set or not.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of SD FLAG (SET or RESET).

__HAL_SD_CLEAR_FLAG

Description:

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_SD_GET_IT

Description:

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of SD IT (SET or RESET).

__HAL_SD_CLEAR_IT

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

SD Card State enumeration structure

HAL_SD_CARD_READY

Card state is ready

HAL_SD_CARD_IDENTIFICATION

Card is in identification state

HAL_SD_CARD_STANDBY

Card is in standby state

HAL_SD_CARD_TRANSFER

Card is in transfer state

HAL_SD_CARD_SENDING

Card is sending an operation

HAL_SD_CARD_RECEIVING

Card is receiving operation information

HAL_SD_CARD_PROGRAMMING

Card is in programming state

HAL_SD_CARD_DISCONNECTED

Card is disconnected

HAL_SD_CARD_ERROR

Card response Error

SD Handle Structure definition

SD_InitTypeDef

SD_TypeDef

69 HAL SD Extension Driver

69.1 SDEx Firmware driver API description

The following section lists the various functions of the SDEx library.

69.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_SDEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_SDEx_ReadBlocksDMAMultiBuffer() and HAL_SDEx_WriteBlocksDMAMultiBuffer() functions.

69.1.2 Linked List management functions

This subsection provides a set of functions allowing to manage the needed functions.

This section contains the following APIs:

- *HAL_SDEx_DMALinkedList_BuildNode()*
- *HAL_SDEx_DMALinkedList_InsertNode()*
- *HAL_SDEx_DMALinkedList_RemoveNode()*
- *HAL_SDEx_DMALinkedList_LockNode()*
- *HAL_SDEx_DMALinkedList_UnlockNode()*
- *HAL_SDEx_DMALinkedList_EnableCircularMode()*
- *HAL_SDEx_DMALinkedList_DisableCircularMode()*
- *HAL_SDEx_DMALinkedList_ReadBlocks()*
- *HAL_SDEx_DMALinkedList_WriteBlocks()*
- *HAL_SDEx_DMALinkedList_ReadBlocks()*
- *HAL_SDEx_DMALinkedList_WriteBlocks()*
- *HAL_SDEx_Read_DMALnkLstBufCpltCallback()*
- *HAL_SDEx_Write_DMALnkLstBufCpltCallback()*

69.1.3 Detailed description of functions

HAL_SDEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_ReadBlocks (SD_HandleTypeDef * hsd, SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_WriteBlocks (SD_HandleTypeDef * hsd, SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_BuildNode (SD_DMALinkNodeTypeDef * pNode, SD_DMALinkNodeConfTypeDef * pNodeConf)

Function description

Build Linked List node.

Parameters

- **pNode:** Pointer to new node to add.
- **pNodeConf:** Pointer to configuration parameters for new node to add.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_InsertNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pPrevNode, SD_DMALinkNodeTypeDef * pNewNode)

Function description

Insert new Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pPrevNode:** Pointer to previous node.
- **pNewNode:** Pointer to new node to insert.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_RemoveNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pNode)

Function description

Remove Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_LockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

Lock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_UnlockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

Unlock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_EnableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

Enable Circular mode for DMA Linked List.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_DisableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

Disable Circular mode for DMA Linked List.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes

Return values

- **HAL:** status

HAL_SDEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Read_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Write_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

69.2 SDEx Firmware driver defines

The following section lists the various define and macros of the module.

69.2.1 SDEx

SDEx

Linked List Wrapper

SD_DMALinkNodeTypeDef

SD_DMALinkNodeConfTypeDef

SD_DMALinkedListTypeDef

HAL_SDEx_DMALinkedList_WriteCpltCallback

HAL_SDEx_DMALinkedList_ReadCpltCallback

70 HAL SMARTCARD Generic Driver

70.1 SMARTCARD Firmware driver registers structures

70.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32h5xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***
  Note:
  – The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***
- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***

- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***
  Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***
  Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***
  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***
  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***
  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***
  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***
  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

70.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32h5xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of ***SMARTCARDEx_Transmission_Completion_Indication***.

70.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32h5xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***const uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code
- ***void(* __SMARTCARD_HandleTypeDef::TxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::ErrorCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Error Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Complete Callback

- ***void(* __SMARTCARD_HandleTypeDef::AbortTransmitCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Transmit Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortReceiveCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Receive Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxFifoFullCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Fifo Full Callback
- ***void(* __SMARTCARD_HandleTypeDef::TxFifoEmptyCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Fifo Empty Callback
- ***void(* __SMARTCARD_HandleTypeDef::MspInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp Init callback
- ***void(* __SMARTCARD_HandleTypeDef::MspDeInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp DeInit callback

70.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

70.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note: The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

70.2.2 Callback registration

The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_SMARTCARD_RegisterCallback() to register a user callback. Function HAL_SMARTCARD_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.

- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_SMARTCARD_UnRegisterCallback() to reset a callback to the default weak function. HAL_SMARTCARD_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit.

By default, after the HAL_SMARTCARD_Init() and when the state is HAL_SMARTCARD_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_SMARTCARD_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_SMARTCARD_STATE_READY or HAL_SMARTCARD_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SMARTCARD_RegisterCallback() before calling HAL_SMARTCARD_DeInit() or HAL_SMARTCARD_Init() function.

When The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

70.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***
- ***HAL_SMARTCARD_RegisterCallback()***
- ***HAL_SMARTCARD_UnRegisterCallback()***

70.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
   c. The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are :
   a. HAL_SMARTCARD_Transmit()
   b. HAL_SMARTCARD_Receive()
3. Non Blocking mode APIs with Interrupt are :
   a. HAL_SMARTCARD_Transmit_IT()
   b. HAL_SMARTCARD_Receive_IT()
   c. HAL_SMARTCARD_IRQHandler()
4. Non Blocking mode functions with DMA are :
   a. HAL_SMARTCARD_Transmit_DMA()
   b. HAL_SMARTCARD_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   a. HAL_SMARTCARD_TxCpltCallback()
   b. HAL_SMARTCARD_RxCpltCallback()
   c. HAL_SMARTCARD_ErrorCallback()
1. Non-Blocking mode transfers could be aborted using Abort API's :
   a. HAL_SMARTCARD_Abort()
   b. HAL_SMARTCARD_AbortTransmit()
   c. HAL_SMARTCARD_AbortReceive()
   d. HAL_SMARTCARD_Abort_IT()
   e. HAL_SMARTCARD_AbortTransmit_IT()
   f. HAL_SMARTCARD_AbortReceive_IT()
2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   a. HAL_SMARTCARD_AbortCpltCallback()
   b. HAL_SMARTCARD_AbortTransmitCpltCallback()
   c. HAL_SMARTCARD_AbortReceiveCpltCallback()

3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   a. Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
   b. Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode transmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- *HAL_SMARTCARD_Transmit()*
- *HAL_SMARTCARD_Receive()*
- *HAL_SMARTCARD_Transmit_IT()*
- *HAL_SMARTCARD_Receive_IT()*
- *HAL_SMARTCARD_Transmit_DMA()*
- *HAL_SMARTCARD_Receive_DMA()*
- *HAL_SMARTCARD_Abort()*
- *HAL_SMARTCARD_AbortTransmit()*
- *HAL_SMARTCARD_AbortReceive()*
- *HAL_SMARTCARD_Abort_IT()*
- *HAL_SMARTCARD_AbortTransmit_IT()*
- *HAL_SMARTCARD_AbortReceive_IT()*
- *HAL_SMARTCARD_IRQHandler()*
- *HAL_SMARTCARD_TxCpltCallback()*
- *HAL_SMARTCARD_RxCpltCallback()*
- *HAL_SMARTCARD_ErrorCallback()*
- *HAL_SMARTCARD_AbortCpltCallback()*
- *HAL_SMARTCARD_AbortTransmitCpltCallback()*
- *HAL_SMARTCARD_AbortReceiveCpltCallback()*

70.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_SMARTCARD_GetState()*
- *HAL_SMARTCARD_GetError()*

70.2.6 Detailed description of functions

HAL_SMARTCARD_Init

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_DeInit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD peripheral.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_MspInit

Function name

void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_RegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID, pSMARTCARD_CallbackTypeDef pCallback)

Function description

Register a User SMARTCARD Callback To be used to override the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_RegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_UnRegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMARTCARD callback SMARTCARD callback is redirected to the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_UnRegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to un-register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_SMARTCARD_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_TDR register is empty, i.e one interrupt per data to transmit.
- When FIFO mode is enabled, USART interrupt is generated whenever TXFIFO threshold reached. In that case the interrupt rate depends on TXFIFO threshold configuration.
- This function sets the hsmartcard->TxIsr function pointer according to the FIFO mode (data transmission processing depends on FIFO mode).

HAL_SMARTCARD_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_RDR register can be read, i.e one interrupt per data to receive.
- When FIFO mode is enabled, USART interrupt is generated whenever RXFIFO threshold reached. In that case the interrupt rate depends on RXFIFO threshold configuration.
- This function sets the hsmartcard->RxIsr function pointer according to the FIFO mode (data reception processing depends on FIFO mode).

HAL_SMARTCARD_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name

void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Handle SMARTCARD interrupt requests.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Tx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Rx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD error callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Receive Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle state.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle error code.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle Error Code

70.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

70.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Prescaler

SMARTCARD_PRESCALER_DIV1

fclk_pres = fclk

SMARTCARD_PRESCALER_DIV2

fclk_pres = fclk/2

SMARTCARD_PRESCALER_DIV4

fclk_pres = fclk/4

SMARTCARD_PRESCALER_DIV6

fclk_pres = fclk/6

SMARTCARD_PRESCALER_DIV8

fclk_pres = fclk/8

SMARTCARD_PRESCALER_DIV10

fclk_pres = fclk/10

SMARTCARD_PRESCALER_DIV12

fclk_pres = fclk/12

SMARTCARD_PRESCALER_DIV16

fclk_pres = fclk/16

SMARTCARD_PRESCALER_DIV32

fclk_pres = fclk/32

SMARTCARD_PRESCALER_DIV64

fclk_pres = fclk/64

SMARTCARD_PRESCALER_DIV128

fclk_pres = fclk/128

SMARTCARD_PRESCALER_DIV256

fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Error Code Definition

HAL_SMARTCARD_ERROR_NONE

No error

HAL_SMARTCARD_ERROR_PE

Parity error

HAL_SMARTCARD_ERROR_NE

Noise error

HAL_SMARTCARD_ERROR_FE

frame error

HAL_SMARTCARD_ERROR_ORE

Overrun error

HAL_SMARTCARD_ERROR_DMA

DMA transfer error

HAL_SMARTCARD_ERROR_RTO

Receiver TimeOut error

HAL_SMARTCARD_ERROR_INVALID_CALLBACK

Invalid Callback error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available)
  - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  - SMARTCARD_FLAG_BUSY Busy flag
  - SMARTCARD_FLAG_EOBF End of block flag
  - SMARTCARD_FLAG_RTOF Receiver timeout flag
  - SMARTCARD_FLAG_TXE Transmit data register empty flag
  - SMARTCARD_FLAG_TC Transmission complete flag
  - SMARTCARD_FLAG_RXNE Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE Idle line detection flag
  - SMARTCARD_FLAG_ORE Overrun error flag
  - SMARTCARD_FLAG_NE Noise error flag
  - SMARTCARD_FLAG_FE Framing error flag
  - SMARTCARD_FLAG_PE Parity error flag
  - SMARTCARD_FLAG_TXFNF TXFIFO not full flag
  - SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
  - SMARTCARD_FLAG_TXFE TXFIFO Empty flag
  - SMARTCARD_FLAG_RXFF RXFIFO Full flag
  - SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
  - SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK

SMARTCARD interruptions flags mask

SMARTCARD_CR_MASK

SMARTCARD control register mask

SMARTCARD_CR_POS

SMARTCARD control register position

SMARTCARD_ISR_MASK

SMARTCARD ISR register mask

SMARTCARD_ISR_POS

SMARTCARD ISR register position

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD State Code Definition

HAL_SMARTCARD_STATE_RESET

Peripheral is not initialized. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_READY

Peripheral Initialized and ready for use. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_SMARTCARD_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_SMARTCARD_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_SMARTCARD_STATE_ERROR

Error Value is allowed for gState only

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

SMARTCARD frame length

71 HAL SMARTCARD Extension Driver

71.1 SMARTCARDEx Firmware driver API description

The following section lists the various functions of the SMARTCARDEx library.

71.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
2. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: When SMARTCARD operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.

71.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature

This section contains the following APIs:

- ***HAL_SMARTCARDEx_BlockLength_Config()***
- ***HAL_SMARTCARDEx_TimeOut_Config()***
- ***HAL_SMARTCARDEx_EnableReceiverTimeOut()***
- ***HAL_SMARTCARDEx_DisableReceiverTimeOut()***

71.1.3 IO operation functions

This subsection provides a set of FIFO mode related callback functions.

1. TX/RX Fifos Callbacks:
   - HAL_SMARTCARDEx_RxFifoFullCallback()
   - HAL_SMARTCARDEx_TxFifoEmptyCallback()

This section contains the following APIs:

- ***HAL_SMARTCARDEx_RxFifoFullCallback()***
- ***HAL_SMARTCARDEx_TxFifoEmptyCallback()***

71.1.4 Peripheral FIFO Control functions

This subsection provides a set of functions allowing to control the SMARTCARD FIFO feature.

- HAL_SMARTCARDEx_EnableFifoMode() API enables the FIFO mode
- HAL_SMARTCARDEx_DisableFifoMode() API disables the FIFO mode
- HAL_SMARTCARDEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_SMARTCARDEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_SMARTCARDEx_EnableFifoMode()***
- ***HAL_SMARTCARDEx_DisableFifoMode()***
- ***HAL_SMARTCARDEx_SetTxFifoThreshold()***
- ***HAL_SMARTCARDEx_SetRxFifoThreshold()***

71.1.5 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name

void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)

Function description

Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **BlockLength:** SMARTCARD block length (8-bit long at most)

Return values

- **None:**

HAL_SMARTCARDEx_TimeOut_Config

Function name

void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD RX Fifo full callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD TX Fifo empty callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_TXFIFO_THRESHOLD_1_8
  - SMARTCARD_TXFIFO_THRESHOLD_1_4
  - SMARTCARD_TXFIFO_THRESHOLD_1_2
  - SMARTCARD_TXFIFO_THRESHOLD_3_4
  - SMARTCARD_TXFIFO_THRESHOLD_7_8
  - SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_RXFIFO_THRESHOLD_1_8
  - SMARTCARD_RXFIFO_THRESHOLD_1_4
  - SMARTCARD_RXFIFO_THRESHOLD_1_2
  - SMARTCARD_RXFIFO_THRESHOLD_3_4
  - SMARTCARD_RXFIFO_THRESHOLD_7_8
  - SMARTCARD_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

71.2 SMARTCARDEx Firmware driver defines

The following section lists the various define and macros of the module.

71.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before of after guard time

SMARTCARD FIFO mode

SMARTCARD_FIFOMODE_DISABLE

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY

SMARTCARD busy flag

SMARTCARD_FLAG_EOBF

SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF

SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE

SMARTCARD transmit data register empty

SMARTCARD_FLAG_TXFNF

SMARTCARD TXFIFO not full

SMARTCARD_FLAG_TC

SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE

SMARTCARD read data register not empty

SMARTCARD_FLAG_RXFNE

SMARTCARD RXFIFO not empty

SMARTCARD_FLAG_IDLE

SMARTCARD idle line detection

SMARTCARD_FLAG_ORE

SMARTCARD overrun error

SMARTCARD_FLAG_NE

SMARTCARD noise error

SMARTCARD_FLAG_FE

SMARTCARD frame error

SMARTCARD_FLAG_PE

SMARTCARD parity error

SMARTCARD_FLAG_TXFE

SMARTCARD TXFIFO Empty flag

SMARTCARD_FLAG_RXFF

SMARTCARD RXFIFO Full flag

SMARTCARD_FLAG_RXFT

SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_TXFT

SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD parity error interruption

SMARTCARD_IT_TXE

SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TXFNF

SMARTCARD TX FIFO not full interruption

SMARTCARD_IT_TC

SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE

SMARTCARD read data register not empty interruption

SMARTCARD_IT_RXFNE

SMARTCARD RXFIFO not empty interruption

SMARTCARD_IT_IDLE

SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR

SMARTCARD error interruption

SMARTCARD_IT_ORE

SMARTCARD overrun error interruption

SMARTCARD_IT_NE

SMARTCARD noise error interruption

SMARTCARD_IT_FE

SMARTCARD frame error interruption

SMARTCARD_IT_EOB

SMARTCARD end of block interruption

SMARTCARD_IT_RTO

SMARTCARD receiver timeout interruption

SMARTCARD_IT_TCBGT

SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF

SMARTCARD RXFIFO full interruption

SMARTCARD_IT_TXFE

SMARTCARD TXFIFO empty interruption

SMARTCARD_IT_RXFT

SMARTCARD RXFIFO threshold reached interruption

SMARTCARD_IT_TXFT

SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF

SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF

SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF

SMARTCARD noise error detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TXFECF

TXFIFO empty Clear Flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARD TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

72 HAL SMBUS Generic Driver

72.1 SMBUS Firmware driver registers structures

72.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32h5xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
  Specifies addressing mode selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***
  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***
- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

72.1.2 __SMBUS_HandleTypeDef

__SMBUS_HandleTypeDef is defined in the stm32h5xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef __SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* __SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t __SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t __SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t __SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t __SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef __SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t __SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t __SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code
- ***void(* __SMBUS_HandleTypeDef::MasterTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::MasterRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Rx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Rx Transfer completed callback

- ***void(* __SMBUS_HandleTypeDef::ListenCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Listen Complete callback
- ***void(* __SMBUS_HandleTypeDef::ErrorCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Error callback
- ***void(* __SMBUS_HandleTypeDef::AddrCallback)(struct __SMBUS_HandleTypeDef *hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  SMBUS Slave Address Match callback
- ***void(* __SMBUS_HandleTypeDef::MspInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp Init callback
- ***void(* __SMBUS_HandleTypeDef::MspDeInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp DeInit callback

72.2 SMBUS Firmware driver API description

The following section lists the various functions of the SMBUS library.

72.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      - Enable the clock for the SMBUS GPIOs
      - Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the SMBUSx interrupt priority
      - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
  – At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
  – The associated previous transfer callback is called at the end of abort process
  – mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  – mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive

- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
  - When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  - At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  - At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() or HAL_SMBUS_DisableAlert_IT()
  - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

Callback registration

The compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SMBUS_RegisterCallback() or HAL_SMBUS_RegisterAddrCallback() to register an interrupt callback.

Function HAL_SMBUS_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_SMBUS_RegisterAddrCallback.

Use function HAL_SMBUS_UnRegisterCallback to reset a callback to the default weak function. HAL_SMBUS_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.

- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_SMBUS_UnRegisterAddrCallback.

By default, after the HAL_SMBUS_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMBUS_MasterTxCpltCallback(), HAL_SMBUS_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SMBUS_RegisterCallback() before calling HAL_SMBUS_DeInit() or HAL_SMBUS_Init() function.

When the compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the SMBUS HAL driver header file for more useful macros

72.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- Enable/Disable Analog/Digital filters with HAL_SMBUS_ConfigAnalogFilter() and HAL_SMBUS_ConfigDigitalFilter().

This section contains the following APIs:

- ***HAL_SMBUS_Init()***
- ***HAL_SMBUS_DeInit()***
- ***HAL_SMBUS_MspInit()***
- ***HAL_SMBUS_MspDeInit()***
- ***HAL_SMBUS_ConfigAnalogFilter()***
- ***HAL_SMBUS_ConfigDigitalFilter()***
- ***HAL_SMBUS_RegisterCallback()***

- *HAL_SMBUS_UnRegisterCallback()*
- *HAL_SMBUS_RegisterAddrCallback()*
- *HAL_SMBUS_UnRegisterAddrCallback()*

72.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :
   - HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   - Non-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are :
   - HAL_SMBUS_Master_Transmit_IT()
   - HAL_SMBUS_Master_Receive_IT()
   - HAL_SMBUS_Slave_Transmit_IT()
   - HAL_SMBUS_Slave_Receive_IT()
   - HAL_SMBUS_EnableListen_IT() or alias HAL_SMBUS_EnableListen_IT()
   - HAL_SMBUS_DisableListen_IT()
   - HAL_SMBUS_EnableAlert_IT()
   - HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in non-Blocking mode:
   - HAL_SMBUS_MasterTxCpltCallback()
   - HAL_SMBUS_MasterRxCpltCallback()
   - HAL_SMBUS_SlaveTxCpltCallback()
   - HAL_SMBUS_SlaveRxCpltCallback()
   - HAL_SMBUS_AddrCallback()
   - HAL_SMBUS_ListenCpltCallback()
   - HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- *HAL_SMBUS_Master_Transmit_IT()*
- *HAL_SMBUS_Master_Receive_IT()*
- *HAL_SMBUS_Master_Abort_IT()*
- *HAL_SMBUS_Slave_Transmit_IT()*
- *HAL_SMBUS_Slave_Receive_IT()*
- *HAL_SMBUS_EnableListen_IT()*
- *HAL_SMBUS_DisableListen_IT()*
- *HAL_SMBUS_EnableAlert_IT()*
- *HAL_SMBUS_DisableAlert_IT()*
- *HAL_SMBUS_IsDeviceReady()*

72.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_SMBUS_GetState()*
- *HAL_SMBUS_GetError()*

72.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DeInit

Function name

HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS peripheral.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_MspInit

Function name

void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MspDeInit

Function name

void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)

Function description

Configure Analog noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **AnalogFilter:** This parameter can be one of the following values:
  - SMBUS_ANALOGFILTER_ENABLE
  - SMBUS_ANALOGFILTER_DISABLE

Return values

- **HAL:** status

HAL_SMBUS_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)

Function description

Configure Digital noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_SMBUS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID, pSMBUS_CallbackTypeDef pCallback)

Function description

Register a User SMBUS Callback To be used instead of the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_SMBUS_ERROR_CB_ID Error callback ID
  - HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  - HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_RegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID.

HAL_SMBUS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMBUS Callback SMBUS callback is redirected to the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_SMBUS_ERROR_CB_ID Error callback ID
  - HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  - HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_UnRegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to un-register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID

HAL_SMBUS_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus, pSMBUS_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match SMBUS Callback To be used instead of the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_SMBUS_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

UnRegister the Slave Address Match SMBUS Callback Info Ready SMBUS Callback is redirected to the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Check if target device is ready for communication.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SMBUS_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)

Function description

Abort a master/host SMBUS process communication with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

Notes

- This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS event interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS error interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **TransferDirection:** Master request Transfer Direction (Write/Read)
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_SMBUS_ListenCpltCallback

Function name

void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Listen Complete callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ErrorCallback

Function name

void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

SMBUS error callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_GetState

Function name

uint32_t HAL_SMBUS_GetState (const SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS handle state.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** state

HAL_SMBUS_GetError

Function name

uint32_t HAL_SMBUS_GetError (const SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS error code.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

72.3 SMBUS Firmware driver defines

The following section lists the various define and macros of the module.

72.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

HAL_SMBUS_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_SMBUS_ERROR_INVALID_PARAM

Invalid Parameters error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (SET or RESET).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert
  - SMBUS_FLAG_BUSY Bus busy
  - SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_FRAME_WITH_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

73 HAL SMBUS Extension Driver

73.1 SMBUSEx Firmware driver API description

The following section lists the various functions of the SMBUSEx library.

73.1.1 SMBUS peripheral Extended features

Comparing to other previous devices, the SMBUS interface for STM32H5xx devices contains the following additional features

- Disable or enable wakeup from Stop mode(s)

73.1.2 How to use this driver

73.1.3 WakeUp Mode Functions

This section provides functions allowing to:

- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_SMBUSEx_EnableWakeUp()***
- ***HAL_SMBUSEx_DisableWakeUp()***

73.1.4 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_SMBUSEx_ConfigFastModePlus()***

73.1.5 Detailed description of functions

HAL_SMBUSEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_EnableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_DisableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_SMBUSEx_ConfigFastModePlus (SMBUS_HandleTypeDef * hsmbus, uint32_t FastModePlus)

Function description

Configure SMBUS Fast Mode Plus.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **FastModePlus:** New state of the Fast Mode Plus.

Return values

- **HAL:** status

73.2 SMBUSEx Firmware driver defines

The following section lists the various define and macros of the module.

73.2.1 SMBUSEx

SMBUSEx

SMBUS Extended Fast Mode Plus

SMBUS_FASTMODEPLUS_ENABLE

Enable Fast Mode Plus

SMBUS_FASTMODEPLUS_DISABLE

Disable Fast Mode Plus

74 HAL SPI Generic Driver

74.1 SPI Firmware driver registers structures

74.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32h5xx_hal_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***
- ***uint32_t NSSPolarity***
- ***uint32_t FifoThreshold***
- ***uint32_t TxCRCInitializationPattern***
- ***uint32_t RxCRCInitializationPattern***
- ***uint32_t MasterSSIdleness***
- ***uint32_t MasterInterDataIdleness***
- ***uint32_t MasterReceiverAutoSusp***
- ***uint32_t MasterKeepIOState***
- ***uint32_t IOSwap***
- ***uint32_t ReadyMasterManagement***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_Management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_Transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_Mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 0 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode***
  This mode is activated by the SSOM bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0).
- ***uint32_t SPI_InitTypeDef::NSSPolarity***
  Specifies which level of SS input/output external signal (present on SS pin) is considered as active one. This parameter can be a value of ***SPI_NSS_Polarity***
- ***uint32_t SPI_InitTypeDef::FifoThreshold***
  Specifies the FIFO threshold level. This parameter can be a value of ***SPI_Fifo_Threshold***
- ***uint32_t SPI_InitTypeDef::TxCRCInitializationPattern***
  Specifies the transmitter CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::RxCRCInitializationPattern***
  Specifies the receiver CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::MasterSSIdleness***
  Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS and first data transaction start in master mode. This parameter can be a value of ***SPI_Master_SS_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterInterDataIdleness***
  Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in master mode. This parameter can be a value of ***SPI_Master_InterData_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterReceiverAutoSusp***
  Control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition. This parameter can be a value of ***SPI_Master_RX_AutoSuspend***
- ***uint32_t SPI_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***SPI_Master_Keep_IO_State***
- ***uint32_t SPI_InitTypeDef::IOSwap***
  Invert MISO/MOSI alternate functions This parameter can be a value of ***SPI_IO_Swap***
- ***uint32_t SPI_InitTypeDef::ReadyMasterManagement***
  Specifies if RDY Signal is managed internally or not. This parameter can be a value of ***SPI_RDY_Master_Management***
- ***uint32_t SPI_InitTypeDef::ReadyPolarity***
  Specifies which level of RDY Signal input (present on RDY pin) is considered as active one. This parameter can be a value of ***SPI_RDY_Polarity***

74.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32h5xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***

- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxRxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* SuspendCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***const uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object

- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code
- ***void(* __SPI_HandleTypeDef::TxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Completed callback
- ***void(* __SPI_HandleTypeDef::RxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Completed callback
- ***void(* __SPI_HandleTypeDef::TxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Half Completed callback
- ***void(* __SPI_HandleTypeDef::RxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Half Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Half Completed callback
- ***void(* __SPI_HandleTypeDef::ErrorCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Error callback
- ***void(* __SPI_HandleTypeDef::AbortCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Abort callback
- ***void(* __SPI_HandleTypeDef::SuspendCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Suspend callback
- ***void(* __SPI_HandleTypeDef::MspInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp Init callback
- ***void(* __SPI_HandleTypeDef::MspDeInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp DeInit callback

74.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

74.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      - Enable the clock for the SPI GPIOs
      - Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process or DMA process
      - Configure the SPIx interrupt priority
      - Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      - Enable the DMAx clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx Stream/Channel
      - Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.

4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/ unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/ DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Callback registration: (#) The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback
- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback
- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback

- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback
- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Circular mode restriction:

- The DMA circular mode cannot be used when the SPI is configured in these modes:
  - Master 2Lines RxOnly
  - Master 1Line Rx
- The CRC feature is not managed when the DMA circular mode is enabled
- The functions HAL_SPI_DMAPause()/ HAL_SPI_DMAResume() are not supported. Return always HAL_ERROR with ErrorCode set to HAL_SPI_ERROR_NOT_SUPPORTED. Those functions are maintained for backward compatibility reasons.

74.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
  - FIFO transmission threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init()***

- *HAL_SPI_DeInit()*
- *HAL_SPI_MspInit()*
- *HAL_SPI_MspDeInit()*
- *HAL_SPI_RegisterCallback()*
- *HAL_SPI_UnRegisterCallback()*

74.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode :

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- *HAL_SPI_Transmit()*
- *HAL_SPI_Receive()*
- *HAL_SPI_TransmitReceive()*
- *HAL_SPI_Transmit_IT()*
- *HAL_SPI_Receive_IT()*
- *HAL_SPI_TransmitReceive_IT()*
- *HAL_SPI_Transmit_DMA()*
- *HAL_SPI_Receive_DMA()*
- *HAL_SPI_TransmitReceive_DMA()*
- *HAL_SPI_Abort()*
- *HAL_SPI_Abort_IT()*
- *HAL_SPI_DMAPause()*
- *HAL_SPI_DMAResume()*
- *HAL_SPI_DMAStop()*
- *HAL_SPI_IRQHandler()*
- *HAL_SPI_TxCpltCallback()*
- *HAL_SPI_RxCpltCallback()*
- *HAL_SPI_TxRxCpltCallback()*
- *HAL_SPI_TxHalfCpltCallback()*
- *HAL_SPI_RxHalfCpltCallback()*
- *HAL_SPI_TxRxHalfCpltCallback()*
- *HAL_SPI_ErrorCallback()*
- *HAL_SPI_AbortCpltCallback()*
- *HAL_SPI_SuspendCallback()*

74.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- HAL_SPI_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL_SPI_GetError() check in run-time Errors occurring during communication

This section contains the following APIs:

- *HAL_SPI_GetState()*
- *HAL_SPI_GetError()*

74.2.5 Detailed description of functions

HAL_SPI_Init

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_DeInit

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI peripheral.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_MspInit

Function name

void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_MspDeInit

Function name

void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_RegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID, pSPI_CallbackTypeDef pCallback)

Function description

Register a User SPI Callback To be used instead of the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_SPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_UnRegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID)

Function description

Unregister an SPI Callback SPI callback is redirected to the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

HAL_SPI_Transmit

Function name

HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Receive

Function name

HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received

Return values

- **HAL:** status

HAL_SPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name

HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)

Function description

Pause the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAResume

Function name

HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)

Function description

Resume the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAStop

Function name

HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)

Function description

Stop the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_Abort

Function name

HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (blocking mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode) + Set handle State to READY.
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (Interrupt mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode) + Set handle State to READY + At abort completion, call user abort complete callback.
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

Handle SPI interrupt request.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **None:**

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Half Transfer callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

SPI error callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Abort Complete callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_SuspendCallback

Function name

void HAL_SPI_SuspendCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Suspend callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_GetState

Function name

HAL_SPI_StateTypeDef HAL_SPI_GetState (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI handle state.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** state

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI error code.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** error code in bitmap format

74.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

74.3.1 SPI

SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_BYPASS

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Calculation Initialization Pattern

SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN

SPI_CRC_INITIALIZATION_ALL_ONE_PATTERN

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_4BIT

SPI_CRC_LENGTH_5BIT

SPI_CRC_LENGTH_6BIT

SPI_CRC_LENGTH_7BIT

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_9BIT

SPI_CRC_LENGTH_10BIT

SPI_CRC_LENGTH_11BIT

SPI_CRC_LENGTH_12BIT

SPI_CRC_LENGTH_13BIT

SPI_CRC_LENGTH_14BIT

SPI_CRC_LENGTH_15BIT

SPI_CRC_LENGTH_16BIT

SPI_CRC_LENGTH_17BIT

SPI_CRC_LENGTH_18BIT

SPI_CRC_LENGTH_19BIT

SPI_CRC_LENGTH_20BIT

SPI_CRC_LENGTH_21BIT

SPI_CRC_LENGTH_22BIT

SPI_CRC_LENGTH_23BIT

SPI_CRC_LENGTH_24BIT

SPI_CRC_LENGTH_25BIT

SPI_CRC_LENGTH_26BIT

SPI_CRC_LENGTH_27BIT

SPI_CRC_LENGTH_28BIT

SPI_CRC_LENGTH_29BIT

SPI_CRC_LENGTH_30BIT

SPI_CRC_LENGTH_31BIT

SPI_CRC_LENGTH_32BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI_DATASIZE_17BIT

SPI_DATASIZE_18BIT

SPI_DATASIZE_19BIT

SPI_DATASIZE_20BIT

SPI_DATASIZE_21BIT

SPI_DATASIZE_22BIT

SPI_DATASIZE_23BIT

SPI_DATASIZE_24BIT

SPI_DATASIZE_25BIT

SPI_DATASIZE_26BIT

SPI_DATASIZE_27BIT

SPI_DATASIZE_28BIT

SPI_DATASIZE_29BIT

SPI_DATASIZE_30BIT

SPI_DATASIZE_31BIT

SPI_DATASIZE_32BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_TXONLY

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Codes

HAL_SPI_ERROR_NONE
: No error

HAL_SPI_ERROR_MODF
: MODF error

HAL_SPI_ERROR_CRC
: CRC error

HAL_SPI_ERROR_OVR
: OVR error

HAL_SPI_ERROR_FRE
: FRE error

HAL_SPI_ERROR_DMA
: DMA transfer error

HAL_SPI_ERROR_FLAG

Error on RXP/TXP/DXP/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

HAL_SPI_ERROR_UDR

Underrun error

HAL_SPI_ERROR_TIMEOUT

Timeout error

HAL_SPI_ERROR_UNKNOW

Unknown error

HAL_SPI_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_SPI_ERROR_INVALID_CALLBACK

Invalid Callback error

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXP : Rx-Packet available flag
  - SPI_FLAG_TXP : Tx-Packet space available flag
  - SPI_FLAG_DXP : Duplex Packet flag
  - SPI_FLAG_EOT : End of transfer flag
  - SPI_FLAG_TXTF : Transmission Transfer Filled flag
  - SPI_FLAG_UDR : Underrun flag
  - SPI_FLAG_OVR : Overrun flag
  - SPI_FLAG_CRCERR : CRC error flag
  - SPI_FLAG_FRE : TI mode frame format error flag
  - SPI_FLAG_MODF : Mode fault flag
  - SPI_FLAG_SUSP : Transfer suspend complete flag
  - SPI_FLAG_TXC : TxFIFO transmission complete flag
  - SPI_FLAG_FRLVL : Fifo reception level flag
  - SPI_FLAG_RXWNE : RxFIFO word not empty flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_UDRFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_EOTFLAG

Description:

- Clear the SPI EOT pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_TXTFFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_SUSPFLAG

Description:

- Clear the SPI SUSP pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

SPI Fifo Threshold

SPI_FIFO_THRESHOLD_01DATA

SPI_FIFO_THRESHOLD_02DATA

SPI_FIFO_THRESHOLD_03DATA

SPI_FIFO_THRESHOLD_04DATA

SPI_FIFO_THRESHOLD_05DATA

SPI_FIFO_THRESHOLD_06DATA

SPI_FIFO_THRESHOLD_07DATA

SPI_FIFO_THRESHOLD_08DATA

SPI_FIFO_THRESHOLD_09DATA

SPI_FIFO_THRESHOLD_10DATA

SPI_FIFO_THRESHOLD_11DATA

SPI_FIFO_THRESHOLD_12DATA

SPI_FIFO_THRESHOLD_13DATA

SPI_FIFO_THRESHOLD_14DATA

SPI_FIFO_THRESHOLD_15DATA

SPI_FIFO_THRESHOLD_16DATA

SPI FIFO Type

SPI_LOWEND_FIFO_SIZE

SPI_HIGHEND_FIFO_SIZE

SPI Flags Definition

SPI_FLAG_RXP

SPI_FLAG_TXP

SPI_FLAG_DXP

SPI_FLAG_EOT

SPI_FLAG_TXTF

SPI_FLAG_UDR

SPI_FLAG_OVR

SPI_FLAG_CRCERR

SPI_FLAG_FRE

SPI_FLAG_MODF

SPI_FLAG_SUSP

SPI_FLAG_TXC

SPI_FLAG_FRLVL

SPI_FLAG_RXWNE

SPI Interrupt Definition

SPI_IT_RXP

SPI_IT_TXP

SPI_IT_DXP

SPI_IT_EOT

SPI_IT_TXTF

SPI_IT_UDR

SPI_IT_OVR

SPI_IT_CRCERR

SPI_IT_FRE

SPI_IT_MODF

SPI_IT_ERR

Control SPI IO Swap

SPI_IO_SWAP_DISABLE

SPI_IO_SWAP_ENABLE

SPI Master Inter-Data Idleness

SPI_MASTER_INTERDATA_IDLENESS_00CYCLE

SPI_MASTER_INTERDATA_IDLENESS_01CYCLE

SPI_MASTER_INTERDATA_IDLENESS_02CYCLE

SPI_MASTER_INTERDATA_IDLENESS_03CYCLE

SPI_MASTER_INTERDATA_IDLENESS_04CYCLE

SPI_MASTER_INTERDATA_IDLENESS_05CYCLE

SPI_MASTER_INTERDATA_IDLENESS_06CYCLE

SPI_MASTER_INTERDATA_IDLENESS_07CYCLE

SPI_MASTER_INTERDATA_IDLENESS_08CYCLE

SPI_MASTER_INTERDATA_IDLENESS_09CYCLE

SPI_MASTER_INTERDATA_IDLENESS_10CYCLE

SPI_MASTER_INTERDATA_IDLENESS_11CYCLE

SPI_MASTER_INTERDATA_IDLENESS_12CYCLE

SPI_MASTER_INTERDATA_IDLENESS_13CYCLE

SPI_MASTER_INTERDATA_IDLENESS_14CYCLE

SPI_MASTER_INTERDATA_IDLENESS_15CYCLE

Keep IO State

SPI_MASTER_KEEP_IO_STATE_DISABLE

SPI_MASTER_KEEP_IO_STATE_ENABLE

SPI Master Receiver AutoSuspend

SPI_MASTER_RX_AUTOSUSP_DISABLE

SPI_MASTER_RX_AUTOSUSP_ENABLE

SPI Master SS Idleness

SPI_MASTER_SS_IDLENESS_00CYCLE

SPI_MASTER_SS_IDLENESS_01CYCLE

SPI_MASTER_SS_IDLENESS_02CYCLE

SPI_MASTER_SS_IDLENESS_03CYCLE

SPI_MASTER_SS_IDLENESS_04CYCLE

SPI_MASTER_SS_IDLENESS_05CYCLE

SPI_MASTER_SS_IDLENESS_06CYCLE

SPI_MASTER_SS_IDLENESS_07CYCLE

SPI_MASTER_SS_IDLENESS_08CYCLE

SPI_MASTER_SS_IDLENESS_09CYCLE

SPI_MASTER_SS_IDLENESS_10CYCLE

SPI_MASTER_SS_IDLENESS_11CYCLE

SPI_MASTER_SS_IDLENESS_12CYCLE

SPI_MASTER_SS_IDLENESS_13CYCLE

SPI_MASTER_SS_IDLENESS_14CYCLE

SPI_MASTER_SS_IDLENESS_15CYCLE

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_DISABLE

SPI_NSS_PULSE_ENABLE

SPI NSS Polarity

SPI_NSS_POLARITY_LOW

SPI_NSS_POLARITY_HIGH

SPI RDY Signal Input Master Management

SPI_RDY_MASTER_MANAGEMENT_INTERNALLY

SPI_RDY_MASTER_MANAGEMENT_EXTERNALLY

SPI RDY Signal Input/Output Polarity

SPI_RDY_POLARITY_HIGH

SPI_RDY_POLARITY_LOW

SPI Reception FIFO Status Level

SPI_RX_FIFO_0PACKET

SPI_RX_FIFO_1PACKET

SPI_RX_FIFO_2PACKET

SPI_RX_FIFO_3PACKET

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Underrun Behavior

SPI_UNDERRUN_BEHAV_REGISTER_PATTERN

SPI_UNDERRUN_BEHAV_LAST_RECEIVED

75 HAL SPI Extension Driver

75.1 SPIEx Firmware driver API description

The following section lists the various functions of the SPIEx library.

75.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. SPIEx function:
   - HAL_SPIEx_FlushRxFifo()
   - HAL_SPIEx_FlushRxFifo()
   - HAL_SPIEx_EnableLockConfiguration()
   - HAL_SPIEx_ConfigureUnderrun()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo()***
- ***HAL_SPIEx_EnableLockConfiguration()***
- ***HAL_SPIEx_ConfigureUnderrun()***

75.1.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (const SPI_HandleTypeDef * hspi)

Function description

Flush the RX fifo.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPIEx_EnableLockConfiguration

Function name

HAL_StatusTypeDef HAL_SPIEx_EnableLockConfiguration (SPI_HandleTypeDef * hspi)

Function description

Enable the Lock for the AF configuration of associated IOs and write protect the Content of Configuration register 2 when SPI is enabled.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPIEx_ConfigureUnderrun

Function name

HAL_StatusTypeDef HAL_SPIEx_ConfigureUnderrun (SPI_HandleTypeDef * hspi, uint32_t UnderrunDetection, uint32_t UnderrunBehaviour)

Function description

Configure the UNDERRUN condition and behavior of slave transmitter.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **UnderrunDetection:** : Detection of underrun condition at slave transmitter This parameter is not supported in this SPI version. It is kept in order to not break the compatibility.
- **UnderrunBehaviour:** : Behavior of slave transmitter at underrun condition This parameter can be a value of SPI Underrun Behavior.

Return values

- **None:**

76 HAL SRAM Generic Driver

76.1 SRAM Firmware driver registers structures

76.1.1 __SRAM_HandleTypeDef

__SRAM_HandleTypeDef is defined in the stm32h5xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef __SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef __SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* __SRAM_HandleTypeDef::hdma***
  Pointer DMA handler
- ***void(* __SRAM_HandleTypeDef::MspInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp Init callback
- ***void(* __SRAM_HandleTypeDef::MspDeInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp DeInit callback
- ***void(* __SRAM_HandleTypeDef::DmaXferCpltCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Complete callback
- ***void(* __SRAM_HandleTypeDef::DmaXferErrorCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Error callback

76.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

76.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

Callback registration

The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SRAM_UnRegisterCallback() to reset a callback to the default weak function. It allows to reset following callbacks:
- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SRAM_Init and if the state is HAL_SRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_SRAM_Init and HAL_SRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SRAM_Init and HAL_SRAM_DeInit keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SRAM_RegisterCallback before calling HAL_SRAM_DeInit or HAL_SRAM_Init function. When The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

76.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***
- ***HAL_SRAM_MspInit()***

- *HAL_SRAM_MspDeInit()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

76.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- *HAL_SRAM_Read_8b()*
- *HAL_SRAM_Write_8b()*
- *HAL_SRAM_Read_16b()*
- *HAL_SRAM_Write_16b()*
- *HAL_SRAM_Read_32b()*
- *HAL_SRAM_Write_32b()*
- *HAL_SRAM_Read_DMA()*
- *HAL_SRAM_Write_DMA()*
- *HAL_SRAM_RegisterCallback()*
- *HAL_SRAM_UnRegisterCallback()*
- *HAL_SRAM_RegisterDmaCallback()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

76.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- *HAL_SRAM_WriteOperation_Enable()*
- *HAL_SRAM_WriteOperation_Disable()*

76.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- *HAL_SRAM_GetState()*

76.2.6 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Performs the SRAM device initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **Timing:** Pointer to SRAM control timing structure
- **ExtTiming:** Pointer to SRAM extended mode timing structure

Return values

- **HAL:** status

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

Performs the SRAM device De-initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP Init.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_MspDeInit

Function name

void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP DeInit.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete error callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_CallbackTypeDef pCallback)

Function description

Register a User SRAM Callback To be used instead of the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_UnRegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId)

Function description

Unregister a User SRAM Callback SRAM Callback is redirected to the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID

Return values

- **status:**

HAL_SRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterDmaCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_DmaCallbackTypeDef pCallback)

Function description

Register a User SRAM Callback for DMA transfers To be used instead of the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

Enables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

Disables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)

Function description

Returns the SRAM controller state.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** state

76.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

76.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- __HANDLE__: SRAM handle

Return value:

- None

77 HAL TIM Generic Driver

77.1 TIM Firmware driver registers structures

77.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF Macro **__HAL_TIM_CALC_PSC()** can be used to calculate prescaler value
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated)Macros **__HAL_TIM_CALC_PERIOD()**, **__HAL_TIM_CALC_PERIOD_DITHER()**,**__HAL_TIM_CALC_PERIOD_BY_DELAY()**, **__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY()**can be used to calculate Period value
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***
  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

77.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note:
  – This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.

77.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2Selection***
- ***uint32_t IC2Prescaler***

- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***
  Note:
  – When the Master/slave mode is enabled, the effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is not mandatory in case of timer synchronization mode.

77.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

77.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***

- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode, This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode, This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level, This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time, This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State, This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity, This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input.This parameter can be a value of ***TIM_Break_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State, This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity, This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input.This parameter can be a value of ***TIM_Break2_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state, This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

77.1.11 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32h5xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelState***

- *__IO HAL_TIM_ChannelStateTypeDef ChannelNState*
- *__IO HAL_TIM_DMABurstStateTypeDef DMABurstState*
- *void(* Base_MspInitCallback*
- *void(* Base_MspDeInitCallback*
- *void(* IC_MspInitCallback*
- *void(* IC_MspDeInitCallback*
- *void(* OC_MspInitCallback*
- *void(* OC_MspDeInitCallback*
- *void(* PWM_MspInitCallback*
- *void(* PWM_MspDeInitCallback*
- *void(* OnePulse_MspInitCallback*
- *void(* OnePulse_MspDeInitCallback*
- *void(* Encoder_MspInitCallback*
- *void(* Encoder_MspDeInitCallback*
- *void(* HallSensor_MspInitCallback*
- *void(* HallSensor_MspDeInitCallback*
- *void(* PeriodElapsedCallback*
- *void(* PeriodElapsedHalfCpltCallback*
- *void(* TriggerCallback*
- *void(* TriggerHalfCpltCallback*
- *void(* IC_CaptureCallback*
- *void(* IC_CaptureHalfCpltCallback*
- *void(* OC_DelayElapsedCallback*
- *void(* PWM_PulseFinishedCallback*
- *void(* PWM_PulseFinishedHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* CommutationCallback*
- *void(* CommutationHalfCpltCallback*
- *void(* BreakCallback*
- *void(* Break2Callback*
- *void(* EncoderIndexCallback*
- *void(* DirectionChangeCallback*
- *void(* IndexErrorCallback*
- *void(* TransitionErrorCallback*

Field Documentation

- ***TIM_TypeDef* __TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef __TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel __TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a *DMA_Handle_index*
- ***HAL_LockTypeDef __TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef __TIM_HandleTypeDef::State***
  TIM operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelState[6]***
  TIM channel operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelNState[4]***
  TIM complementary channel operation state

- ***__IO HAL_TIM_DMABurstStateTypeDef __TIM_HandleTypeDef::DMABurstState***
  DMA burst operation state
- ***void(* __TIM_HandleTypeDef::Base_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Base_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::IC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::IC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp Init Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp Init Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed half complete Callback
- ***void(* __TIM_HandleTypeDef::TriggerCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger Callback
- ***void(* __TIM_HandleTypeDef::TriggerHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger half complete Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture half complete Callback
- ***void(* __TIM_HandleTypeDef::OC_DelayElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Output Compare Delay Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished half complete Callback
- ***void(* __TIM_HandleTypeDef::ErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Error Callback
- ***void(* __TIM_HandleTypeDef::CommutationCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation Callback
- ***void(* __TIM_HandleTypeDef::CommutationHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation half complete Callback

- ***void(* __TIM_HandleTypeDef::BreakCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break Callback
- ***void(* __TIM_HandleTypeDef::Break2Callback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break2 Callback
- ***void(* __TIM_HandleTypeDef::EncoderIndexCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Index Callback
- ***void(* __TIM_HandleTypeDef::DirectionChangeCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Direction Change Callback
- ***void(* __TIM_HandleTypeDef::IndexErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Index Error Callback
- ***void(* __TIM_HandleTypeDef::TransitionErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Transition Error Callback

77.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

77.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental encoder for positioning purposes

77.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Time Base : HAL_TIM_Base_MspInit()
   – Input Capture : HAL_TIM_IC_MspInit()
   – Output Compare : HAL_TIM_OC_MspInit()
   – PWM generation : HAL_TIM_PWM_MspInit()
   – One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   – Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.

4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   - HAL_TIM_Base_Init: to use the Timer to generate a simple time base
   - HAL_TIM_OC_Init, HAL_TIM_OC_ConfigChannel and optionally HAL_TIMEx_OC_ConfigPulseOnCompare: to use the Timer to generate an Output Compare signal.
   - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
   - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
   - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
   - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
   - Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
   - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
   - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
   - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
   - One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
   - Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

Callback registration

The compilation define USE_HAL_TIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_TIM_RegisterCallback() to register a callback. HAL_TIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_TIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_TIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- Base_MspInitCallback : TIM Base Msp Init Callback.
- Base_MspDeInitCallback : TIM Base Msp DeInit Callback.
- IC_MspInitCallback : TIM IC Msp Init Callback.
- IC_MspDeInitCallback : TIM IC Msp DeInit Callback.
- OC_MspInitCallback : TIM OC Msp Init Callback.
- OC_MspDeInitCallback : TIM OC Msp DeInit Callback.
- PWM_MspInitCallback : TIM PWM Msp Init Callback.
- PWM_MspDeInitCallback : TIM PWM Msp DeInit Callback.
- OnePulse_MspInitCallback : TIM One Pulse Msp Init Callback.
- OnePulse_MspDeInitCallback : TIM One Pulse Msp DeInit Callback.
- Encoder_MspInitCallback : TIM Encoder Msp Init Callback.
- Encoder_MspDeInitCallback : TIM Encoder Msp DeInit Callback.
- HallSensor_MspInitCallback : TIM Hall Sensor Msp Init Callback.
- HallSensor_MspDeInitCallback : TIM Hall Sensor Msp DeInit Callback.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger Callback.
- TriggerHalfCpltCallback : TIM Trigger half complete Callback.
- IC_CaptureCallback : TIM Input Capture Callback.
- IC_CaptureHalfCpltCallback : TIM Input Capture half complete Callback.
- OC_DelayElapsedCallback : TIM Output Compare Delay Elapsed Callback.
- PWM_PulseFinishedCallback : TIM PWM Pulse Finished Callback.
- PWM_PulseFinishedHalfCpltCallback : TIM PWM Pulse Finished half complete Callback.

- ErrorCallback : TIM Error Callback.
- CommutationCallback : TIM Commutation Callback.
- CommutationHalfCpltCallback : TIM Commutation half complete Callback.
- BreakCallback : TIM Break Callback.
- Break2Callback : TIM Break2 Callback.
- EncoderIndexCallback : TIM Encoder Index Callback.
- DirectionChangeCallback : TIM Direction Change Callback
- IndexErrorCallback : TIM Index Error Callback.
- TransitionErrorCallback : TIM Transition Error Callback

By default, after the Init and when the state is HAL_TIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_TIM_TriggerCallback(), HAL_TIM_ErrorCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init / DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init / DeInit keep and use the user MspInit / MspDeInit callbacks(registered beforehand)

Callbacks can be registered / unregistered in HAL_TIM_STATE_READY state only. Exception done MspInit / MspDeInit that can be registered / unregistered in HAL_TIM_STATE_READY or HAL_TIM_STATE_RESET state, thus registered(user) MspInit / DeInit callbacks can be used during the Init / DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_TIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_TIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

77.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

77.2.4 TIM Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the TIM Output Compare.
- Stop the TIM Output Compare.
- Start the TIM Output Compare and enable interrupt.
- Stop the TIM Output Compare and disable interrupt.

- Start the TIM Output Compare and enable DMA transfer.
- Stop the TIM Output Compare and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_OC_Init()***
- ***HAL_TIM_OC_DeInit()***
- ***HAL_TIM_OC_MspInit()***
- ***HAL_TIM_OC_MspDeInit()***
- ***HAL_TIM_OC_Start()***
- ***HAL_TIM_OC_Stop()***
- ***HAL_TIM_OC_Start_IT()***
- ***HAL_TIM_OC_Stop_IT()***
- ***HAL_TIM_OC_Start_DMA()***
- ***HAL_TIM_OC_Stop_DMA()***

77.2.5 TIM PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the TIM PWM.
- Stop the TIM PWM.
- Start the TIM PWM and enable interrupt.
- Stop the TIM PWM and disable interrupt.
- Start the TIM PWM and enable DMA transfer.
- Stop the TIM PWM and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_PWM_Init()***
- ***HAL_TIM_PWM_DeInit()***
- ***HAL_TIM_PWM_MspInit()***
- ***HAL_TIM_PWM_MspDeInit()***
- ***HAL_TIM_PWM_Start()***
- ***HAL_TIM_PWM_Stop()***
- ***HAL_TIM_PWM_Start_IT()***
- ***HAL_TIM_PWM_Stop_IT()***
- ***HAL_TIM_PWM_Start_DMA()***
- ***HAL_TIM_PWM_Stop_DMA()***

77.2.6 TIM Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the TIM Input Capture.
- Stop the TIM Input Capture.
- Start the TIM Input Capture and enable interrupt.
- Stop the TIM Input Capture and disable interrupt.
- Start the TIM Input Capture and enable DMA transfer.
- Stop the TIM Input Capture and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_IC_Init()***
- ***HAL_TIM_IC_DeInit()***

- *HAL_TIM_IC_MspInit()*
- *HAL_TIM_IC_MspDeInit()*
- *HAL_TIM_IC_Start()*
- *HAL_TIM_IC_Stop()*
- *HAL_TIM_IC_Start_IT()*
- *HAL_TIM_IC_Stop_IT()*
- *HAL_TIM_IC_Start_DMA()*
- *HAL_TIM_IC_Stop_DMA()*

77.2.7 TIM One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the TIM One Pulse.
- Stop the TIM One Pulse.
- Start the TIM One Pulse and enable interrupt.
- Stop the TIM One Pulse and disable interrupt.
- Start the TIM One Pulse and enable DMA transfer.
- Stop the TIM One Pulse and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OnePulse_Init()*
- *HAL_TIM_OnePulse_DeInit()*
- *HAL_TIM_OnePulse_MspInit()*
- *HAL_TIM_OnePulse_MspDeInit()*
- *HAL_TIM_OnePulse_Start()*
- *HAL_TIM_OnePulse_Stop()*
- *HAL_TIM_OnePulse_Start_IT()*
- *HAL_TIM_OnePulse_Stop_IT()*

77.2.8 TIM Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the TIM Encoder.
- Stop the TIM Encoder.
- Start the TIM Encoder and enable interrupt.
- Stop the TIM Encoder and disable interrupt.
- Start the TIM Encoder and enable DMA transfer.
- Stop the TIM Encoder and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_Encoder_Init()*
- *HAL_TIM_Encoder_DeInit()*
- *HAL_TIM_Encoder_MspInit()*
- *HAL_TIM_Encoder_MspDeInit()*
- *HAL_TIM_Encoder_Start()*
- *HAL_TIM_Encoder_Stop()*
- *HAL_TIM_Encoder_Start_IT()*
- *HAL_TIM_Encoder_Stop_IT()*
- *HAL_TIM_Encoder_Start_DMA()*

- *HAL_TIM_Encoder_Stop_DMA()*

77.2.9 TIM Callbacks functions

This section provides TIM callback functions:

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback
- TIM Index callback
- TIM Direction change callback
- TIM Index error callback
- TIM Transition error callback

This section contains the following APIs:

- *HAL_TIM_PeriodElapsedCallback()*
- *HAL_TIM_PeriodElapsedHalfCpltCallback()*
- *HAL_TIM_OC_DelayElapsedCallback()*
- *HAL_TIM_IC_CaptureCallback()*
- *HAL_TIM_IC_CaptureHalfCpltCallback()*
- *HAL_TIM_PWM_PulseFinishedCallback()*
- *HAL_TIM_PWM_PulseFinishedHalfCpltCallback()*
- *HAL_TIM_TriggerCallback()*
- *HAL_TIM_TriggerHalfCpltCallback()*
- *HAL_TIM_ErrorCallback()*
- *HAL_TIM_RegisterCallback()*
- *HAL_TIM_UnRegisterCallback()*

77.2.10 Detailed description of functions

HAL_TIM_Base_Init

Function name

HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Base_DeInit() before HAL_TIM_Base_Init()

HAL_TIM_Base_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Base peripheral.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_MspInit

Function name

void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_MspDeInit

Function name

void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_Start

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OC_DeInit() before HAL_TIM_OC_Init()

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

HAL_TIM_OC_MspInit

Function name

void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_MspDeInit

Function name

void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_Start

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_PWM_DeInit() before HAL_TIM_PWM_Init()

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

HAL_TIM_PWM_MspInit

Function name

void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_MspDeInit

Function name

void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_IC_DeInit() before HAL_TIM_IC_Init()

HAL_TIM_IC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

HAL_TIM_IC_MspInit

Function name

void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture MSP.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **None:**

HAL_TIM_IC_MspDeInit

Function name

void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Input Capture MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_IC_Start

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Init

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)

Function description

Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM One Pulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OnePulse_DeInit() before HAL_TIM_OnePulse_Init()
- When the timer instance is initialized in One Pulse mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, const TIM_Encoder_InitTypeDef * sConfig)

Function description

Initializes the TIM Encoder Interface and initialize the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Encoder Interface configuration structure

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Encoder_DeInit() before HAL_TIM_Encoder_Init()
- Encoder mode and External clock mode 2 are not compatible and must not be selected together Ex: A call for HAL_TIM_Encoder_Init will erase the settings of HAL_TIM_ConfigClockSource using TIM_CLOCKSOURCE_ETRMODE2 and vice versa
- When the timer instance is initialized in Encoder mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_Encoder_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Encoder interface.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_MspDeInit

Function name

void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

This function handles TIM interrupts requests.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM Output Compare handle
- **sConfig:** TIM Output Compare configuration structure
- **Channel:** TIM Channels to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM PWM handle
- **sConfig:** TIM PWM configuration structure
- **Channel:** TIM Channels to be configured This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_IC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim:** TIM IC handle
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** TIM Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function description

Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** TIM One Pulse handle
- **sConfig:** TIM One Pulse configuration structure
- **OutputChannel:** TIM output channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** TIM input Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- To output a waveform with a minimum delay user can enable the fast mode by calling the __HAL_TIM_ENABLE_OCxFAST macro. Then CCx output is forced in response to the edge detection on TIx input, without taking in account the comparison.

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, const TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **HAL:** status

HAL_TIM_ConfigClockSource

Function name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, const TIM_ClockConfigTypeDef * sClockSourceConfig)

Function description

Configures the clock source to be used.

Parameters

- **htim:** TIM handle
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** TIM handle.
- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  - TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  - TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro_IT (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiWriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer multiple Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stops the TIM DMA Burst mode.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stop the DMA burst reading.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values

- **HAL:** status

HAL_TIM_GenerateEvent

Function name

HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description

Generate a software event.

Parameters

- **htim:** TIM handle
- **EventSource:** specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  - TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  - TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  - TIM_EVENTSOURCE_COM: Timer COM event source
  - TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  - TIM_EVENTSOURCE_BREAK: Timer Break event source
  - TIM_EVENTSOURCE_BREAK2: Timer Break2 event source

Return values

- **HAL:** status

Notes

- Basic timers can only generate an update event.
- TIM_EVENTSOURCE_COM is relevant only with advanced timer instances.
- TIM_EVENTSOURCE_BREAK and TIM_EVENTSOURCE_BREAK2 are relevant only for timer instances supporting break input(s).

HAL_TIM_ReadCapturedValue

Function name

uint32_t HAL_TIM_ReadCapturedValue (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Read the captured value from Capture Compare unit.

Parameters

- **htim:** TIM handle.
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name

void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Output Compare callback in non-blocking mode.

Parameters

- **htim:** TIM OC handle

Return values

- **None:**

HAL_TIM_IC_CaptureCallback

Function name

void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_IC_CaptureHalfCpltCallback

Function name

void HAL_TIM_IC_CaptureHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedHalfCpltCallback

Function name

void HAL_TIM_PWM_PulseFinishedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerHalfCpltCallback

Function name

void HAL_TIM_TriggerHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

Timer error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_RegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID, pTIM_CallbackTypeDef pCallback)

Function description

Register a User TIM callback to be used instead of the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  - HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  - HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  - HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  - HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  - HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  - HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  - HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  - HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  - HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  - HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  - HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  - HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  - HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  - HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  - HAL_TIM_ERROR_CB_ID Error Callback ID
  - HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  - HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  - HAL_TIM_BREAK_CB_ID Break Callback ID
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID
  - HAL_TIM_ENCODER_INDEX_CB_ID Encoder Index Callback ID
  - HAL_TIM_DIRECTION_CHANGE_CB_ID Direction Change Callback ID
  - HAL_TIM_INDEX_ERROR_CB_ID Index Error Callback ID
  - HAL_TIM_TRANSITION_ERROR_CB_ID Transition Error Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_TIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_UnRegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a TIM callback TIM callback is redirected to the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  - HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  - HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  - HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  - HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  - HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  - HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  - HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  - HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  - HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  - HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  - HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  - HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  - HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  - HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  - HAL_TIM_ERROR_CB_ID Error Callback ID
  - HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  - HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  - HAL_TIM_BREAK_CB_ID Break Callback ID
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID
  - HAL_TIM_ENCODER_INDEX_CB_ID Encoder Index Callback ID
  - HAL_TIM_DIRECTION_CHANGE_CB_ID Direction Change Callback ID
  - HAL_TIM_INDEX_ERROR_CB_ID Index Error Callback ID
  - HAL_TIM_TRANSITION_ERROR_CB_ID Transition Error Callback ID

Return values

- **status:**

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Base handle state.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** state

HAL_TIM_OC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM OC handle state.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** state

HAL_TIM_PWM_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM PWM handle state.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** state

HAL_TIM_IC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Input Capture handle state.

Parameters

- **htim:** TIM IC handle

Return values

- **HAL:** state

HAL_TIM_OnePulse_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM One Pulse Mode handle state.

Parameters

- **htim:** TIM OPM handle

Return values

- **HAL:** state

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** state

HAL_TIM_GetActiveChannel

Function name

HAL_TIM_ActiveChannel HAL_TIM_GetActiveChannel (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM handle

Return values

- **Active:** channel

HAL_TIM_GetChannelState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIM_GetChannelState (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Return actual state of the TIM channel.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **TIM:** Channel state

HAL_TIM_DMABurstState

Function name

HAL_TIM_DMABurstStateTypeDef HAL_TIM_DMABurstState (const TIM_HandleTypeDef * htim)

Function description

Return actual state of a DMA burst operation.

Parameters

- **htim:** TIM handle

Return values

- **DMA:** burst state

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, const TIM_Base_InitTypeDef * Structure)

Function description

Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
  - TIM_ICPOLARITY_RISING
  - TIM_ICPOLARITY_FALLING
  - TIM_ICPOLARITY_BOTHEDGE
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
  - TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  - TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  - TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, const TIM_OC_InitTypeDef * OC_Config)

Function description

Timer Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The output configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
  - TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF.
  - TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2.
  - TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4.
  - TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
  - TIM_ETRPOLARITY_INVERTED: active low or falling edge active.
  - TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Delay Pulse half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_DISABLE.

Return values

- **None:**

TIM_DMA_Start_IT

Function name

HAL_StatusTypeDef TIM_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t src, uint32_t dst, uint32_t length)

Function description

Start the DMA data transfer.

Parameters

- **hdma:** DMA handle
- **src:** : The source memory Buffer address.
- **dst:** : The destination memory Buffer address.
- **length:** : The size of a source block transfer in byte.

Return values

- **HAL:** status

TIM_ResetCallback

Function name

void TIM_ResetCallback (TIM_HandleTypeDef * htim)

Function description

Reset interrupt callbacks to the legacy weak callbacks.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

77.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

77.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break2 Input Alternate Function Mode

TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

Break input BRK2 is disabled

TIM_BREAK2_ENABLE

Break input BRK2 is enabled

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

Break input BRK2 is active low

TIM_BREAK2POLARITY_HIGH

Break input BRK2 is active high

TIM Break Input Alternate Function Mode

TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

TIM Break Input Enable

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM Break System

TIM_BREAK_SYSTEM_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17/20 Break Input and also the PVDE and PLS bits of the Power Control Interface

TIM_BREAK_SYSTEM_SRAM_PARITY_ERROR

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_LOCKUP

Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/8/15/16/17/20

CCx DMA request selection

TIM_CCDMAREQUEST_CC

CCx DMA request sent when capture or compare match event occurs

TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

TIM Channel

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_5

Compare channel 5 identifier

TIM_CHANNEL_6

Compare channel 6 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM_CLEARINPUTSOURCE_OCREFCLR

OCREF_CLR is connected to OCREF_CLR_INT

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2

Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4

Clock division: tDTS=4*tCK_INT

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_ITR4

External clock source mode 1 (ITR4)

TIM_CLOCKSOURCE_ITR5

External clock source mode 1 (ITR5)

TIM_CLOCKSOURCE_ITR6

External clock source mode 1 (ITR6)

TIM_CLOCKSOURCE_ITR7

External clock source mode 1 (ITR7)

TIM_CLOCKSOURCE_ITR8

External clock source mode 1 (ITR8)

TIM_CLOCKSOURCE_ITR9

External clock source mode 1 (ITR9)

TIM_CLOCKSOURCE_ITR10

External clock source mode 1 (ITR10)

TIM_CLOCKSOURCE_ITR11

External clock source mode 1 (ITR11)

TIM_CLOCKSOURCE_ITR12

External clock source mode 1 (ITR12)

TIM Commutation Source

TIM_COMMUTATION_TRGI

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit or when an rising edge occurs on trigger input

TIM_COMMUTATION_SOFTWARE

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit

TIM Counter Mode

TIM_COUNTERMODE_UP

Counter used as up-counter

TIM_COUNTERMODE_DOWN

Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned mode 3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_CCMR3

TIM_DMABASE_DTR2

TIM_DMABASE_ECR

TIM_DMABASE_TISEL

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

The transfer is done to 1 register starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_19TRANSFERS

The transfer is done to 19 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_20TRANSFERS

The transfer is done to 20 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_21TRANSFERS

The transfer is done to 21 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_22TRANSFERS

The transfer is done to 22 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_23TRANSFERS

The transfer is done to 23 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_24TRANSFERS

The transfer is done to 24 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_25TRANSFERS

The transfer is done to 25 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_26TRANSFERS

The transfer is done to 26 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction, x2 mode

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_UIFREMAP_ENABLE

Description:

- Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

Notes:

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

__HAL_TIM_UIFREMAP_DISABLE

Description:

- Disable update interrupt flag (UIF) remapping.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

__HAL_TIM_GET_UIFCPY

Description:

- Get update interrupt flag (UIF) copy status.

Parameters:

- __COUNTER__: Counter value.

Return value:

- The: state of UIFCPY (TRUE or FALSE). mode.

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

__HAL_TIM_SET_COUNTER

Description:

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value
  - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_ENABLE_OCxFAST

Description:

- Enable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles. Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST

Description:

- Disable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

__HAL_TIM_SELECT_CCDMAREQUEST

Description:

- Select the Capture/compare DMA request source.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __CCDMA__: specifies Capture/compare DMA request source This parameter can be one of the following values:
  - TIM_CCDMAREQUEST_CC: CCx DMA request generated on Capture/Compare event
  - TIM_CCDMAREQUEST_UPDATE: CCx DMA request generated on Update event

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_CC5

Capture/Compare 5 interrupt flag

TIM_FLAG_CC6

Capture/Compare 6 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_BREAK2

Break 2 interrupt flag

TIM_FLAG_SYSTEM_BREAK

System Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM_FLAG_IDX

Encoder index flag

TIM_FLAG_DIR

Direction change flag

TIM_FLAG_IERR

Index error flag

TIM_FLAG_TERR

Transition error flag

TIM Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM Input Capture Prescaler

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2

Capture performed once every 2 events

TIM_ICPSC_DIV4

Capture performed once every 4 events

TIM_ICPSC_DIV8

Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM_IT_IDX

Index interrupt

TIM_IT_DIR

Direction change interrupt

TIM_IT_IERR

Index error interrupt

TIM_IT_TERR

Transition error interrupt

TIM Lock level

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM Master Mode Selection

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM_TRGO_ENCODER_CLK

Encoder clock is used as trigger output(TRGO)

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO2)

TIM_TRGO2_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO2)

TIM_TRGO2_UPDATE

Update event is used as trigger output (TRGO2)

TIM_TRGO2_OC1

Capture or a compare match 1 is used as trigger output (TRGO2)

TIM_TRGO2_OC1REF

OC1REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC2REF

OC2REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC3REF

OC3REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF

OC4REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC5REF

OC5REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC6REF

OC6REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF_RISINGFALLING

OC4REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC6REF_RISINGFALLING

OC6REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

OC4REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

OC4REF rising or OC6REF falling edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM One Pulse Mode

TIM_OPMODE_SINGLE

Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE

Counter is not stopped at update event

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

Frozen

TIM_OCMODE_ACTIVE

Set channel to active level on match

TIM_OCMODE_INACTIVE

Set channel to inactive level on match

TIM_OCMODE_TOGGLE

Toggle

TIM_OCMODE_PWM1

PWM mode 1

TIM_OCMODE_PWM2

PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

Force active level

TIM_OCMODE_FORCED_INACTIVE

Force inactive level

TIM_OCMODE_RETRIGERRABLE_OPM1

Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

TIM_OCMODE_PULSE_ON_COMPARE

Pulse on compare (CH3&CH4 only)

TIM_OCMODE_DIRECTION_OUTPUT

Direction output (CH3&CH4 only)

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE

Capture/Compare 1 output enabled

TIM Output Fast State

TIM_OCFAST_DISABLE

Output Compare fast disable

TIM_OCFAST_ENABLE

Output Compare fast enable

TIM Slave mode

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED

Gated Mode

TIM_SLAVEMODE_TRIGGER

Trigger Mode

TIM_SLAVEMODE_EXTERNAL1

External Clock Mode 1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode

TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

Internal Trigger 0 (ITR0)

TIM_TS_ITR1

Internal Trigger 1 (ITR1)

TIM_TS_ITR2

Internal Trigger 2 (ITR2)

TIM_TS_ITR3

Internal Trigger 3 (ITR3)

TIM_TS_ITR4

Internal Trigger 4 (ITR4)

TIM_TS_ITR5

Internal Trigger 5 (ITR5)

TIM_TS_ITR6

Internal Trigger 6 (ITR6)

TIM_TS_ITR7

Internal Trigger 7 (ITR7)

TIM_TS_ITR8

Internal Trigger 8 (ITR8)

TIM_TS_ITR9

Internal Trigger 9 (ITR9)

TIM_TS_ITR10

Internal Trigger 10 (ITR10)

TIM_TS_ITR11

Internal Trigger 11 (ITR11)

TIM_TS_ITR12

Internal Trigger 12 (ITR12)

TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_NONE

No trigger selected

TIM Update Interrupt Flag Remap

TIM_UIFREMAP_DISABLE

Update interrupt flag remap disabled

TIM_UIFREMAP_ENABLE

Update interrupt flag remap enabled

78 HAL TIM Extension Driver

78.1 TIMEx Firmware driver registers structures

78.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32h5xx_hal_tim_ex.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

78.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32h5xx_hal_tim_ex.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity***

78.1.3 TIMEx_EncoderIndexConfigTypeDef

TIMEx_EncoderIndexConfigTypeDef is defined in the stm32h5xx_hal_tim_ex.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t Prescaler***
- ***uint32_t Filter***
- ***uint32_t Blanking***
- ***FunctionalState FirstIndexEnable***
- ***uint32_t Position***
- ***uint32_t Direction***

Field Documentation

- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Polarity***
  TIM Encoder index polarity.This parameter can be a value of ***TIMEx_Encoder_Index_Polarity***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Prescaler***
  TIM Encoder index prescaler.This parameter can be a value of ***TIMEx_Encoder_Index_Prescaler***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Filter***
  TIM Encoder index filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Blanking***
  Specifies whether or not the encoder index event is conditioned by TI3 or TI4 input.This parameter can be a value of ***TIMEx_Encoder_Index_Blanking***
- ***FunctionalState TIMEx_EncoderIndexConfigTypeDef::FirstIndexEnable***
  Specifies whether or not the encoder first index is enabled.This parameter value can be ENABLE or DISABLE.
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Position***
  Specifies in which AB input configuration the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Position***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Direction***
  Specifies in which counter direction the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Direction***

78.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

78.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
5. In case of Pulse on compare, configure pulse length and delay
6. Encoder index configuration

78.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Hall Sensor output : HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   – HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. In case of Pulse On Compare:
   – HAL_TIMEx_OC_ConfigPulseOnCompare(): to configure pulse width and prescaler
6. Activate the TIM peripheral using one of the start functions:
   – Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

78.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_HallSensor_Init()***
- ***HAL_TIMEx_HallSensor_DeInit()***
- ***HAL_TIMEx_HallSensor_MspInit()***
- ***HAL_TIMEx_HallSensor_MspDeInit()***
- ***HAL_TIMEx_HallSensor_Start()***
- ***HAL_TIMEx_HallSensor_Stop()***
- ***HAL_TIMEx_HallSensor_Start_IT()***
- ***HAL_TIMEx_HallSensor_Stop_IT()***
- ***HAL_TIMEx_HallSensor_Start_DMA()***
- ***HAL_TIMEx_HallSensor_Stop_DMA()***

78.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_OCN_Start()***
- ***HAL_TIMEx_OCN_Stop()***
- ***HAL_TIMEx_OCN_Start_IT()***
- ***HAL_TIMEx_OCN_Stop_IT()***
- ***HAL_TIMEx_OCN_Start_DMA()***
- ***HAL_TIMEx_OCN_Stop_DMA()***

78.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- ***HAL_TIMEx_PWMN_Start()***
- ***HAL_TIMEx_PWMN_Stop()***
- ***HAL_TIMEx_PWMN_Start_IT()***
- ***HAL_TIMEx_PWMN_Stop_IT()***
- ***HAL_TIMEx_PWMN_Start_DMA()***
- ***HAL_TIMEx_PWMN_Stop_DMA()***

78.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- ***HAL_TIMEx_OnePulseN_Start()***
- ***HAL_TIMEx_OnePulseN_Stop()***
- ***HAL_TIMEx_OnePulseN_Start_IT()***
- ***HAL_TIMEx_OnePulseN_Stop_IT()***

78.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Select timer input source.
- Enable or disable channel grouping.
- Configure Pulse on compare.
- Configure Encoder index.

This section contains the following APIs:

- *HAL_TIMEx_ConfigCommutEvent()*
- *HAL_TIMEx_ConfigCommutEvent_IT()*
- *HAL_TIMEx_ConfigCommutEvent_DMA()*
- *HAL_TIMEx_MasterConfigSynchronization()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_ConfigBreakInput()*
- *HAL_TIMEx_RemapConfig()*
- *HAL_TIMEx_TISelection()*
- *HAL_TIMEx_GroupChannel5()*
- *HAL_TIMEx_DisarmBreakInput()*
- *HAL_TIMEx_ReArmBreakInput()*
- *HAL_TIMEx_DitheringEnable()*
- *HAL_TIMEx_DitheringDisable()*
- *HAL_TIMEx_OC_ConfigPulseOnCompare()*
- *HAL_TIMEx_ConfigSlaveModePreload()*
- *HAL_TIMEx_EnableSlaveModePreload()*
- *HAL_TIMEx_DisableSlaveModePreload()*
- *HAL_TIMEx_EnableDeadTimePreload()*
- *HAL_TIMEx_DisableDeadTimePreload()*
- *HAL_TIMEx_ConfigDeadTime()*
- *HAL_TIMEx_ConfigAsymmetricalDeadTime()*
- *HAL_TIMEx_EnableAsymmetricalDeadTime()*
- *HAL_TIMEx_DisableAsymmetricalDeadTime()*
- *HAL_TIMEx_ConfigEncoderIndex()*
- *HAL_TIMEx_EnableEncoderIndex()*
- *HAL_TIMEx_DisableEncoderIndex()*
- *HAL_TIMEx_EnableEncoderFirstIndex()*
- *HAL_TIMEx_DisableEncoderFirstIndex()*

78.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- *HAL_TIMEx_CommutCallback()*
- *HAL_TIMEx_CommutHalfCpltCallback()*
- *HAL_TIMEx_BreakCallback()*
- *HAL_TIMEx_Break2Callback()*
- *HAL_TIMEx_EncoderIndexCallback()*
- *HAL_TIMEx_DirectionChangeCallback()*
- *HAL_TIMEx_IndexErrorCallback()*
- *HAL_TIMEx_TransitionErrorCallback()*

78.2.9 Extended Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_TIMEx_HallSensor_GetState()*
- *HAL_TIMEx_GetChannelNState()*

78.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, const TIM_HallSensor_InitTypeDef * sConfig)

Function description

Initializes the TIM Hall Sensor Interface and initialize the associated handle.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **sConfig:** TIM Hall Sensor configuration structure

Return values

- **HAL:** status

Notes

- When the timer instance is initialized in Hall Sensor Interface mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIMEx_HallSensor_DeInit

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name

void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name

void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM OC handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_ConfigCommutEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_ITR12: Internal trigger 12 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_ITR12: Internal trigger 12 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR9: Internal trigger 9 selected
  - TIM_TS_ITR10: Internal trigger 10 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_ITR12: Internal trigger 12 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, const TIM_MasterConfigTypeDef * sMasterConfig)

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, const TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

Notes

- Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

HAL_TIMEx_ConfigBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, const TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)

Function description

Configures the break input source.

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to configure This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input
- **sBreakInputConfig:** Break input source configuration

Return values

- **HAL:** status

HAL_TIMEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- **HAL:** status

HAL_TIMEx_RemapConfig

Function name

HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)

Function description

Configures the TIMx Remapping input capabilities.

Parameters

- **htim:** TIM handle.
- **Remap:** specifies the TIM remapping source. For TIM1, the parameter can take one of the following values:
  - TIM_TIM1_ETR_GPIO TIM1 ETR is connected to GPIO
  - TIM_TIM1_ETR_COMP1 TIM1 ETR is connected to COMP1 output (*)
  - TIM_TIM1_ETR_ADC1_AWD1 TIM1 ETR is connected to ADC1 AWD1
  - TIM_TIM1_ETR_ADC1_AWD2 TIM1 ETR is connected to ADC1 AWD2
  - TIM_TIM1_ETR_ADC1_AWD3 TIM1 ETR is connected to ADC1 AWD3

  For TIM2, the parameter can take one of the following values:
  - TIM_TIM2_ETR_GPIO TIM2 ETR is connected to GPIO
  - TIM_TIM2_ETR_COMP1 TIM2 ETR is connected to COMP1 output (*)
  - TIM_TIM2_ETR_LSE TIM2 ETR is connected to LSE
  - TIM_TIM2_ETR_SAI1_FSA TIM2 ETR is connected to SAI1 FSA (*)
  - TIM_TIM2_ETR_SAI1_FSB TIM2 ETR is connected to SAI1 FSB (*)
  - TIM_TIM2_ETR_TIM3_ETR TIM2 ETR is connected to TIM3 ETR pin
  - TIM_TIM2_ETR_TIM4_ETR TIM2 ETR is connected to TIM4 ETR pin (*)
  - TIM_TIM2_ETR_TIM5_ETR TIM2 ETR is connected to TIM5 ETR pin (*)
  - TIM_TIM2_ETR_ETH_PPS TIM2 ETR is connected to ETH PPS (*)

  For TIM3, the parameter can take one of the following values:
  - TIM_TIM3_ETR_GPIO TIM3 ETR is connected to GPIO
  - TIM_TIM3_ETR_COMP1 TIM3 ETR is connected to COMP1 output (*)
  - TIM_TIM3_ETR_TIM2_ETR TIM3 ETR is connected to TIM2 ETR pin
  - TIM_TIM3_ETR_TIM4_ETR TIM3 ETR is connected to TIM4 ETR pin (*)
  - TIM_TIM3_ETR_TIM5_ETR TIM3 ETR is connected to TIM5 ETR pin (*)
  - TIM_TIM3_ETR_ETH_PPS TIM3 ETR is connected to ETH PPS (*)

  For TIM4, the parameter can take one of the following values: (**)
  - TIM_TIM4_ETR_GPIO TIM4 ETR is connected to GPIO
  - TIM_TIM4_ETR_TIM2_ETR TIM4 ETR is connected to TIM2 ETR pin
  - TIM_TIM4_ETR_TIM3_ETR TIM4 ETR is connected to TIM3 ETR pin
  - TIM_TIM4_ETR_TIM5_ETR TIM4 ETR is connected to TIM5 ETR pin

  For TIM5, the parameter can take one of the following values: (**)
  - TIM_TIM5_ETR_GPIO TIM5 ETR is connected to GPIO
  - TIM_TIM2_ETR_SAI2_FSA TIM2 ETR is connected to SAI2 FSA
  - TIM_TIM2_ETR_SAI2_FSB TIM2 ETR is connected to SAI2 FSB
  - TIM_TIM5_ETR_TIM2_ETR TIM5 ETR is connected to TIM2 ETR pin
  - TIM_TIM5_ETR_TIM3_ETR TIM5 ETR is connected to TIM3 ETR pin
  - TIM_TIM5_ETR_TIM4_ETR TIM5 ETR is connected to TIM4 ETR pin

  For TIM8, the parameter can take one of the following values: (**)
  - TIM_TIM8_ETR_GPIO TIM8 ETR is connected to GPIO
  - TIM_TIM8_ETR_ADC2_AWD1 TIM8 ETR is connected to ADC2 AWD1
  - TIM_TIM8_ETR_ADC2_AWD2 TIM8 ETR is connected to ADC2 AWD2
  - TIM_TIM8_ETR_ADC2_AWD3 TIM8 ETR is connected to ADC2 AWD3

  (*) Value not defined in all devices. (**) Timer instance not available on all devices.
-

Return values

- **HAL:** status

HAL_TIMEx_TISelection

Function name

HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef * htim, uint32_t TISelection, uint32_t Channel)

Function description

Select the timer input source.

Parameters

- **htim:** TIM handle.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TI1 input channel
  - TIM_CHANNEL_2: TI2 input channel
  - TIM_CHANNEL_4: TI4 input channel

- **TISelection:** parameter of the TIM_TISelectionStruct structure is detailed as follows: For TIM1, the parameter is one of the following values:
  - TIM_TIM1_TI1_GPIO: TIM1 TI1 is connected to GPIO
  - TIM_TIM1_TI1_COMP1: TIM1 TI1 is connected to COMP1 output (*)
  - TIM_TIM1_TI2_GPIO: TIM1 TI2 is connected to GPIO
  - TIM_TIM1_TI3_GPIO: TIM1 TI3 is connected to GPIO
  - TIM_TIM1_TI4_GPIO: TIM1 TI4 is connected to GPIO

  For TIM2, the parameter is one of the following values:
  - TIM_TIM2_TI1_GPIO: TIM2 TI1 is connected to GPIO
  - TIM_TIM2_TI1_LSI: TIM2 TI1 is connected to LSI (*)
  - TIM_TIM2_TI1_LSE: TIM2 TI1 is connected to LSE (*)
  - TIM_TIM2_TI1_ETH_PPS TIM2 TI1 is connected to ETH PPS (*)
  - TIM_TIM2_TI1_RTC_WKUP: TIM2 TI2 is connected to RTC_WKUP (*)
  - TIM_TIM2_TI1_TIM3_TI1: TIM2 TI2 is connected to TIM3_TI1 (*)
  - TIM_TIM2_TI2_GPIO: TIM2 TI2 is connected to GPIO
  - TIM_TIM2_TI2_HSI_1024: TIM2 TI2 is connected to HSI/1024 (*)
  - TIM_TIM2_TI2_CSI_128: TIM2 TI2 is connected to CSI/128 (*)
  - TIM_TIM2_TI2_MCO2: TIM2 TI2 is connected to MCO1 (*)
  - TIM_TIM2_TI2_MCO1: TIM2 TI2 is connected to MCO1 (*)
  - TIM_TIM2_TI3_GPIO: TIM2 TI3 is connected to GPIO
  - TIM_TIM2_TI4_GPIO: TIM2 TI4 is connected to GPIO
  - TIM_TIM2_TI4_COMP1: TIM2 TI4 is connected to COMP1 output (*)

  For TIM3, the parameter is one of the following values:
  - TIM_TIM3_TI1_GPIO: TIM3 TI1 is connected to GPIO
  - TIM_TIM3_TI1_COMP1: TIM3 TI1 is connected to COMP1 output (*)
  - TIM_TIM3_TI1_MCO1: TIM3 TI2 is connected to MCO1 (*)
  - TIM_TIM3_TI1_TIM2_TI1: TIM3 TI2 is connected to TIM2 TI1 (*)
  - TIM_TIM3_TI1_HSE_1MHZ: TIM3 TI2 is connected to HSE_1MHZ (*)
  - TIM_TIM3_TI1_ETH_PPS TIM3 TI1 is connected to ETH PPS (*)
  - TIM_TIM3_TI2_GPIO: TIM3 TI2 is connected to GPIO
  - TIM_TIM3_TI2_CSI_128: TIM3 TI2 is connected to CSI_128 (*)
  - TIM_TIM3_TI2_MCO2: TIM3 TI2 is connected to MCO2 (*)
  - TIM_TIM3_TI2_HSI_1024: TIM3 TI2 is connected to HSI_1024 (*)
  - TIM_TIM3_TI3_GPIO: TIM3 TI2 is connected to GPIO
  - TIM_TIM3_TI4_GPIO: TIM3 TI2 is connected to GPIO

  For TIM4, the parameter is one of the following values: (**)
  - TIM_TIM4_TI1_GPIO: TIM4 TI1 is connected to GPIO
  - TIM_TIM4_TI2_GPIO: TIM4 TI2 is connected to GPIO
  - TIM_TIM4_TI3_GPIO: TIM4 TI3 is connected to GPIO
  - TIM_TIM4_TI4_GPIO: TIM4 TI4 is connected to GPIO

  For TIM5, the parameter is one of the following values: (**)
  - TIM_TIM5_TI1_GPIO: TIM5 TI1 is connected to GPIO
  - TIM_TIM5_TI2_GPIO: TIM5 TI2 is connected to GPIO
  - TIM_TIM5_TI3_GPIO: TIM5 TI3 is connected to GPIO
  - TIM_TIM5_TI4_GPIO: TIM5 TI4 is connected to GPIO

  For TIM8, the parameter is one of the following values: (**)
  - TIM_TIM8_TI1_GPIO: TIM8 TI1 is connected to GPIO
  - TIM_TIM8_TI2_GPIO: TIM8 TI2 is connected to GPIO
  - TIM_TIM8_TI3_GPIO: TIM8 TI3 is connected to GPIO
  - TIM_TIM8_TI4_GPIO: TIM8 TI4 is connected to GPIO

For TIM12, the parameter is one of the following values: (**)
– TIM_TIM12_TI1_GPIO: TIM12 TI1 is connected to GPIO
– TIM_TIM12_TI1_HSI_1024: TIM12 TI1 is connected to HSI/1024
– TIM_TIM12_TI1_CSI_128: TIM12 TI1 is connected to CSI/128
For TIM13, the parameter is one of the following values: (**)
– TIM_TIM12_TI1_GPIO: TIM13 TI1 is connected to GPIO
For TIM14, the parameter is one of the following values: (**)
– TIM_TIM14_TI1_GPIO: TIM14 TI1 is connected to GPIO
For TIM15, the parameter can have the following values: (**)
– TIM_TIM15_TI1_GPIO: TIM15 TI1 is connected to GPIO
– TIM_TIM15_TI1_TIM2: TIM15 TI1 is connected to TIM2
– TIM_TIM15_TI1_TIM3: TIM15 TI1 is connected to TIM3
– TIM_TIM15_TI1_TIM4: TIM15 TI1 is connected to TIM4
– TIM_TIM15_TI1_LSE: TIM15 TI1 is connected to LSE
– TIM_TIM15_TI1_CSI_128: TIM15 TI1 is connected to CSI/128
– TIM_TIM15_TI1_MCO: TIM15 TI1 is connected to MCO
– TIM_TIM15_TI2_GPIO: TIM15 TI1 is connected to GPIO
– TIM_TIM15_TI2_TIM2: TIM15 TI1 is connected to TIM2
– TIM_TIM15_TI2_TIM3: TIM15 TI1 is connected to TIM3
– TIM_TIM15_TI2_TIM4: TIM15 TI1 is connected to TIM4
For TIM16, the parameter is one of the following values: (**)
– TIM_TIM16_TI1_GPIO: TIM16 TI1 is connected to GPIO
– TIM_TIM16_TI1_LSI: TIM16 TI1 is connected to LSI
– TIM_TIM16_TI1_LSE: TIM16 TI1 is connected to LSE
– TIM_TIM16_TI1_RTC_WKUP: TIM16 TI1 is connected to RTCWKUP
For TIM17, the parameter can have the following values: (**)
– TIM_TIM17_TI1_GPIO: TIM17 TI1 is connected to GPIO
– TIM_TIM17_TI1_HSE_1MHZ: TIM17 TI1 is connected to HSE_1MHZ
– TIM_TIM17_TI1_MCO: TIM17 TI1 is connected to MCO
(*) Value not defined in all devices.

- (**) Timer instance not available on all devices.
-

Return values

- **HAL:** status

HAL_TIMEx_DisarmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_DisarmBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Disarm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to disarm This parameter can be one of the following values:
  – TIM_BREAKINPUT_BRK: Timer break input
  – TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

HAL_TIMEx_ReArmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ReArmBreakInput (const TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Arm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to arm This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- Arming is possible at anytime, even if fault is present.
- Break input is automatically armed as soon as MOE bit is set.

HAL_TIMEx_DitheringEnable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringEnable (TIM_HandleTypeDef * htim)

Function description

Enable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- Main usage is PWM mode
- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode for 16b timers (corresponds to 4094 for the integer part and 15 for the dithered part).
- Macros __HAL_TIM_CALC_PERIOD_DITHER() __HAL_TIM_CALC_DELAY_DITHER() __HAL_TIM_CALC_PULSE_DITHER() can be used to calculate period (ARR) and delay (CCRx) value.
- Enabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part.
- Enabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_DitheringDisable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringDisable (TIM_HandleTypeDef * htim)

Function description

Disable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode (corresponds to 4094 for the integer part and 15 for the dithered part).
- Disabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_OC_ConfigPulseOnCompare

Function name

HAL_StatusTypeDef HAL_TIMEx_OC_ConfigPulseOnCompare (TIM_HandleTypeDef * htim, uint32_t PulseWidthPrescaler, uint32_t PulseWidth)

Function description

Initializes the pulse on compare pulse width and pulse prescaler.

Parameters

- **htim:** TIM Output Compare handle
- **PulseWidthPrescaler:** Pulse width prescaler This parameter can be a number between Min_Data = 0x0 and Max_Data = 0x7
- **PulseWidth:** Pulse width This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

Return values

- **HAL:** status

HAL_TIMEx_ConfigSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigSlaveModePreload (TIM_HandleTypeDef * htim, uint32_t Source)

Function description

Configure preload source of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle
- **Source:** Source of slave mode selection preload This parameter can be one of the following values:
  - TIM_SMS_PRELOAD_SOURCE_UPDATE: Timer update event is used as source of Slave Mode Selection preload
  - TIM_SMS_PRELOAD_SOURCE_INDEX: Timer index event is used as source of Slave Mode Selection preload

Return values

- **HAL:** status

HAL_TIMEx_EnableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Enable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Disable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Enable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Disable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigDeadTime (TIM_HandleTypeDef * htim, uint32_t Deadtime)

Function description

Configure deadtime.

Parameters

- **htim:** TIM handle
- **Deadtime:** Deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_ConfigAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigAsymmetricalDeadTime (TIM_HandleTypeDef * htim, uint32_t FallingDeadtime)

Function description

Configure asymmetrical deadtime.

Parameters

- **htim:** TIM handle
- **FallingDeadtime:** Falling edge deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_EnableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Enable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Disable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigEncoderIndex (TIM_HandleTypeDef * htim, TIMEx_EncoderIndexConfigTypeDef * sEncoderIndexConfig)

Function description

Configures the encoder index.

Parameters

- **htim:** TIM handle.
- **sEncoderIndexConfig:** Encoder index configuration

Return values

- **HAL:** status

Notes

- warning in case of encoder mode clock plus direction TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1 or TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2 Direction must be set to TIM_ENCODERINDEX_DIRECTION_UP_DOWN

HAL_TIMEx_EnableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_CommutHalfCpltCallback

Function name

void HAL_TIMEx_CommutHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_BreakCallback

Function name

void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)

Function description

Hall Break detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_Break2Callback

Function name

void HAL_TIMEx_Break2Callback (TIM_HandleTypeDef * htim)

Function description

Hall Break2 detection callback in non blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_EncoderIndexCallback

Function name

void HAL_TIMEx_EncoderIndexCallback (TIM_HandleTypeDef * htim)

Function description

Encoder index callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_DirectionChangeCallback

Function name

void HAL_TIMEx_DirectionChangeCallback (TIM_HandleTypeDef * htim)

Function description

Direction change callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_IndexErrorCallback

Function name

void HAL_TIMEx_IndexErrorCallback (TIM_HandleTypeDef * htim)

Function description

Index error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_TransitionErrorCallback

Function name

void HAL_TIMEx_TransitionErrorCallback (TIM_HandleTypeDef * htim)

Function description

Transition error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Hall Sensor interface handle state.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** state

HAL_TIMEx_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (const TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

Return actual state of the TIM complementary channel.

Parameters

- **htim:** TIM handle
- **ChannelN:** TIM Complementary channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

Return values

- **TIM:** Complementary channel state

TIMEx_DMACommutationCplt

Function name

void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIMEx_DMACommutationHalfCplt

Function name

void TIMEx_DMACommutationHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

78.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

78.3.1 TIMEx

TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

Timer break input

TIM_BREAKINPUT_BRK2

Timer break2 input

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

An external source (GPIO) is connected to the BKIN pin

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

Break input source is disabled

TIM_BREAKINPUTSOURCE_ENABLE

Break input source is enabled

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

Break input source is active low

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

Break input source is active_high

TIM Extended Encoder index blanking

TIM_ENCODERINDEX_BLANKING_DISABLE

Encoder index blanking is disabled

TIM_ENCODERINDEX_BLANKING_TI3

Encoder index blanking is enabled on TI3

TIM_ENCODERINDEX_BLANKING_TI4

Encoder index blanking is enabled on TI4

TIM Extended Encoder index direction

TIM_ENCODERINDEX_DIRECTION_UP_DOWN

Index resets the counter whatever the direction

TIM_ENCODERINDEX_DIRECTION_UP

Index resets the counter when up-counting only

TIM_ENCODERINDEX_DIRECTION_DOWN

Index resets the counter when down-counting only

TIM Extended Encoder index polarity

TIM_ENCODERINDEX_POLARITY_INVERTED

Polarity for ETRx pin

TIM_ENCODERINDEX_POLARITY_NONINVERTED

Polarity for ETRx pin

TIM Extended Encoder index position

TIM_ENCODERINDEX_POSITION_00

Encoder index position is AB=00

TIM_ENCODERINDEX_POSITION_01

Encoder index position is AB=01

TIM_ENCODERINDEX_POSITION_10

Encoder index position is AB=10

TIM_ENCODERINDEX_POSITION_11

Encoder index position is AB=11

TIM_ENCODERINDEX_POSITION_0

In directional clock mode or clock plus direction mode, index resets the counter when clock is 0

TIM_ENCODERINDEX_POSITION_1

In directional clock mode or clock plus direction mode, index resets the counter when clock is 1

TIM Extended Encodder index prescaler

TIM_ENCODERINDEX_PRESCALER_DIV1

No prescaler is used

TIM_ENCODERINDEX_PRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_ENCODERINDEX_PRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_ENCODERINDEX_PRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Extended Exported Macros

__HAL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PSC(80000000, 1000000);

__HAL_TIM_CALC_PERIOD

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD(1000000, 0, 10000);

__HAL_TIM_CALC_PERIOD_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER(1000000, 0, 10000); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PULSE(1000000, 0, 10);

__HAL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PULSE_DITHER(1000000, 0, 10); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PERIOD_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_BY_DELAY(1000000, 0, 10, 20);

__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY(1000000, 0, 10, 20); This macro should be used only if dithering is already enabled

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM1_ETR is not connected to I/O

TIM_TIM1_ETR_ADC1_AWD1

TIM1_ETR is connected to ADC1 AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM1_ETR is connected to ADC1 AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM1_ETR is connected to ADC1 AWD3

TIM_TIM2_ETR_GPIO

TIM2_ETR is not connected to I/O

TIM_TIM2_ETR_LSE

TIM2_ETR is connected to LSE

TIM_TIM2_ETR_SAI1_FSA

TIM2_ETR is connected to SAI1 FS_A

TIM_TIM2_ETR_SAI1_FSB

TIM2_ETR is connected to SAI1

TIM_TIM2_ETR_TIM3_ETR

TIM2_ETR is connected to TIM3 ETR

TIM_TIM2_ETR_TIM4_ETR

TIM2_ETR is connected to TIM4 ETR

TIM_TIM2_ETR_TIM5_ETR

TIM2_ETR is connected to TIM5 ETR

TIM_TIM2_ETR_ETH_PPS

TIM2_ETR is connected to ETH PPS

TIM_TIM3_ETR_GPIO

TIM3_ETR is not connected to I/O

TIM_TIM3_ETR_TIM2_ETR

TIM3_ETR is connected to TIM2 ETR

TIM_TIM3_ETR_TIM4_ETR

TIM3_ETR is connected to TIM4 ETR

TIM_TIM3_ETR_TIM5_ETR

TIM3_ETR is connected to TIM5 ETR

TIM_TIM3_ETR_ETH_PPS

TIM3_ETR is connected to ETH PPS

TIM_TIM4_ETR_GPIO

TIM4_ETR is not connected to I/O

TIM_TIM4_ETR_TIM2_ETR

TIM4_ETR is connected to TIM2 ETR

TIM_TIM4_ETR_TIM3_ETR

TIM4_ETR is connected to TIM3 ETR

TIM_TIM4_ETR_TIM5_ETR

TIM4_ETR is connected to TIM5 ETR

TIM_TIM5_ETR_GPIO

TIM5_ETR is not connected to I/O

TIM_TIM5_ETR_SAI2_FSA

TIM5_ETR is connected to SAI2

TIM_TIM5_ETR_SAI2_FSB

TIM5_ETR is connected to SAI2

TIM_TIM5_ETR_TIM2_ETR

TIM5_ETR is connected to TIM2 ETR

TIM_TIM5_ETR_TIM3_ETR

TIM5_ETR is connected to TIM3 ETR

TIM_TIM5_ETR_TIM4_ETR

TIM5_ETR is connected to TIM4 ETR

TIM_TIM8_ETR_GPIO

TIM8_ETR is not connected to I/O

TIM_TIM8_ETR_ADC2_AWD1

TIM8_ETR is connected to ADC1 AWD1

TIM_TIM8_ETR_ADC2_AWD2

TIM8_ETR is connected to ADC1 AWD2

TIM_TIM8_ETR_ADC2_AWD3

TIM8_ETR is connected to ADC1 AWD3

TIM Extended Bitfield SMS preload enabling

TIM_SMS_PRELOAD_SOURCE_UPDATE

Prelaod of SMS bitfield is disabled

TIM_SMS_PRELOAD_SOURCE_INDEX

Preload of SMS bitfield is enabled

TIM Extended Timer input selection

TIM_TIM1_TI1_GPIO

TIM1_TI1 is connected to GPIO

TIM_TIM1_TI2_GPIO

TIM1_TI2 is connected to GPIO

TIM_TIM1_TI3_GPIO

TIM1_TI3 is connected to GPIO

TIM_TIM1_TI4_GPIO

TIM1_TI4 is connected to GPIO

TIM_TIM2_TI1_GPIO

TIM2_TI1 is connected to GPIO

TIM_TIM2_TI1_ETH_PPS

TIM2_TI1 is connected to ETH PPS

TIM_TIM2_TI2_GPIO

TIM2_TI2 is connected to GPIO

TIM_TIM2_TI3_GPIO

TIM2_TI3 is connected to GPIO

TIM_TIM2_TI4_GPIO

TIM2_TI4 is connected to GPIO

TIM_TIM3_TI1_GPIO

TIM3_TI1 is connected to GPIO

TIM_TIM3_TI1_ETH_PPS

TIM3_TI1 is connected to ETH PPS

TIM_TIM3_TI2_GPIO

TIM3_TI2 is connected to GPIO

TIM_TIM3_TI3_GPIO

TIM3_TI3 is connected to GPIO

TIM_TIM3_TI4_GPIO

TIM3_TI4 is connected to GPIO

TIM_TIM4_TI1_GPIO

TIM4_TI1 is connected to GPIO

TIM_TIM4_TI2_GPIO

TIM4_TI2 is connected to GPIO

TIM_TIM4_TI3_GPIO

TIM4_TI3 is connected to GPIO

TIM_TIM4_TI4_GPIO

TIM4_TI4 is connected to GPIO

TIM_TIM5_TI1_GPIO

TIM5_TI1 is connected to GPIO

TIM_TIM5_TI2_GPIO

TIM5_TI2 is connected to GPIO

TIM_TIM5_TI3_GPIO

TIM5_TI3 is connected to GPIO

TIM_TIM5_TI4_GPIO

TIM5_TI4 is connected to GPIO

TIM_TIM8_TI1_GPIO

TIM8_TI1 is connected to GPIO

TIM_TIM8_TI2_GPIO

TIM8_TI2 is connected to GPIO

TIM_TIM8_TI3_GPIO

TIM8_TI3 is connected to GPIO

TIM_TIM8_TI4_GPIO

TIM8_TI4 is connected to GPIO

TIM_TIM12_TI1_GPIO

TIM12_TI1 is connected to GPIO

TIM_TIM12_TI1_HSI_1024

TIM12_TI1 is connected to HSI 1024

TIM_TIM12_TI1_CSI_128

TIM12_TI1 is connected to CSI 128

TIM_TIM13_TI1_GPIO

TIM13_TI1 is connected to GPIO

TIM_TIM14_TI1_GPIO

TIM14_TI1 is connected to GPIO

TIM_TIM15_TI1_GPIO

TIM15_TI1 is connected to GPIO

TIM_TIM15_TI1_TIM2

TIM15_TI1 is connected to TIM2

TIM_TIM15_TI1_TIM3

TIM15_TI1 is connected to TIM3

TIM_TIM15_TI1_TIM4

TIM15_TI1 is connected to TIM4

TIM_TIM15_TI1_LSE

TIM15_TI1 is connected to LSE

TIM_TIM15_TI1_CSI_128

TIM15_TI1 is connected to CSI 128

TIM_TIM15_TI1_MCO2

TIM15_TI1 is connected to MCO2

TIM_TIM15_TI2_GPIO

TIM15_TI1 is connected to GPIO

TIM_TIM15_TI2_TIM2

TIM15_TI2 is connected to TIM2

TIM_TIM15_TI2_TIM3

TIM15_TI2 is connected to TIM3

TIM_TIM15_TI2_TIM4

TIM15_TI2 is connected to TIM4

TIM_TIM16_TI1_GPIO

TIM16_TI1 is connected to GPIO

TIM_TIM16_TI1_LSI

TIM16_TI1 is connected to LSI

TIM_TIM16_TI1_LSE

TIM16_TI1 is connected to LSE

TIM_TIM16_TI1_RTC_WKUP

TIM16_TI1 is connected to RTC

TIM_TIM17_TI1_GPIO

TIM17_TI1 is connected to GPIO

TIM_TIM17_TI1_HSE_1MHZ

TIM17_TI1 is connected to HSE 1MHZ

TIM_TIM17_TI1_MCO1

TIM17_TI1 is connected to MCO1

79 HAL UART Generic Driver

79.1 UART Firmware driver registers structures

79.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32h5xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: LPUART: ======= Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler UART: =====
  – If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))
  – If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate))) [15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.
- ***uint32_t UART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

79.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32h5xx_hal_uart.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

79.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32h5xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***

- ***uint16_t Mask***
- ***uint32_t FifoMode***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***__IO HAL_UART_RxTypeTypeDef ReceptionType***
- ***__IO HAL_UART_RxEventTypeTypeDef RxEventType***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* WakeupCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* RxEventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***const uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.

- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***__IO HAL_UART_RxTypeTypeDef __UART_HandleTypeDef::ReceptionType***
  Type of ongoing reception
- ***__IO HAL_UART_RxEventTypeTypeDef __UART_HandleTypeDef::RxEventType***
  Type of Rx Event
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code
- ***void(* __UART_HandleTypeDef::TxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Half Complete Callback
- ***void(* __UART_HandleTypeDef::TxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Complete Callback
- ***void(* __UART_HandleTypeDef::RxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Half Complete Callback
- ***void(* __UART_HandleTypeDef::RxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Complete Callback
- ***void(* __UART_HandleTypeDef::ErrorCallback)(struct __UART_HandleTypeDef *huart)***
  UART Error Callback
- ***void(* __UART_HandleTypeDef::AbortCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Complete Callback
- ***void(* __UART_HandleTypeDef::AbortTransmitCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Transmit Complete Callback
- ***void(* __UART_HandleTypeDef::AbortReceiveCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Receive Complete Callback
- ***void(* __UART_HandleTypeDef::WakeupCallback)(struct __UART_HandleTypeDef *huart)***
  UART Wakeup Callback
- ***void(* __UART_HandleTypeDef::RxFifoFullCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Fifo Full Callback
- ***void(* __UART_HandleTypeDef::TxFifoEmptyCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Fifo Empty Callback
- ***void(* __UART_HandleTypeDef::RxEventCallback)(struct __UART_HandleTypeDef *huart, uint16_t Pos)***
  UART Reception Event Callback
- ***void(* __UART_HandleTypeDef::MspInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp Init callback
- ***void(* __UART_HandleTypeDef::MspDeInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp DeInit callback

79.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

79.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   - Enable the USARTx interface clock.
   - UART pins configuration:
     - Enable the clock for the UART GPIOs.
     - Configure these UART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - UART interrupts handling:

Note: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.

   - DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

79.2.2 Callback registration

The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_UART_RegisterCallback() to register a user callback. Function HAL_UART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.

- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_UART_UnRegisterCallback() to reset a callback to the default weak function. HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

For specific callback RxEventCallback, use dedicated registration/reset functions: respectively HAL_UART_RegisterRxEventCallback() , HAL_UART_UnRegisterRxEventCallback().

By default, after the HAL_UART_Init() and when the state is HAL_UART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_UART_TxCpltCallback(), HAL_UART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_UART_Init() and HAL_UART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_UART_Init() and HAL_UART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_UART_RegisterCallback() before calling HAL_UART_DeInit() or HAL_UART_Init() function.

When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

79.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***
- ***HAL_UART_MspDeInit()***
- ***HAL_UART_RegisterCallback()***
- ***HAL_UART_UnRegisterCallback()***
- ***HAL_UART_RegisterRxEventCallback()***
- ***HAL_UART_UnRegisterRxEventCallback()***

79.2.4 IO operation functions

This section contains the following APIs:

- ***HAL_UART_Transmit()***
- ***HAL_UART_Receive()***
- ***HAL_UART_Transmit_IT()***
- ***HAL_UART_Receive_IT()***
- ***HAL_UART_Transmit_DMA()***
- ***HAL_UART_Receive_DMA()***
- ***HAL_UART_DMAPause()***
- ***HAL_UART_DMAResume()***
- ***HAL_UART_DMAStop()***
- ***HAL_UART_Abort()***
- ***HAL_UART_AbortTransmit()***
- ***HAL_UART_AbortReceive()***
- ***HAL_UART_Abort_IT()***
- ***HAL_UART_AbortTransmit_IT()***

- *HAL_UART_AbortReceive_IT()*
- *HAL_UART_IRQHandler()*
- *HAL_UART_TxCpltCallback()*
- *HAL_UART_TxHalfCpltCallback()*
- *HAL_UART_RxCpltCallback()*
- *HAL_UART_RxHalfCpltCallback()*
- *HAL_UART_ErrorCallback()*
- *HAL_UART_AbortCpltCallback()*
- *HAL_UART_AbortTransmitCpltCallback()*
- *HAL_UART_AbortReceiveCpltCallback()*
- *HAL_UARTEx_RxEventCallback()*

79.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.

- HAL_UART_ReceiverTimeout_Config() API allows to configure the receiver timeout value on the fly
- HAL_UART_EnableReceiverTimeout() API enables the receiver timeout feature
- HAL_UART_DisableReceiverTimeout() API disables the receiver timeout feature
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- *HAL_UART_ReceiverTimeout_Config()*
- *HAL_UART_EnableReceiverTimeout()*
- *HAL_UART_DisableReceiverTimeout()*
- *HAL_MultiProcessor_EnableMuteMode()*
- *HAL_MultiProcessor_DisableMuteMode()*
- *HAL_MultiProcessor_EnterMuteMode()*
- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*
- *HAL_LIN_SendBreak()*

79.2.6 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- *HAL_UART_GetState()*
- *HAL_UART_GetError()*

79.2.7 Detailed description of functions

HAL_UART_Init

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_Init

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_LIN_Init

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values:
  - UART_LINBREAKDETECTLENGTH_10B 10-bit break detection
  - UART_LINBREAKDETECTLENGTH_11B 11-bit break detection

Return values

- **HAL:** status

HAL_MultiProcessor_Init

Function name

HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)

Function description

Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.
- **Address:** UART node address (4-, 6-, 7- or 8-bit long).
- **WakeUpMethod:** Specifies the UART wakeup method. This parameter can be one of the following values:
  - UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection
  - UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark

Return values

- **HAL:** status

Notes

- If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.
- If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name

HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

Initialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_MspDeInit

Function name

void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID, pUART_CallbackTypeDef pCallback)

Function description

Register a User UART Callback To be used to override the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_UART_RegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID)

Function description

Unregister an UART Callback UART callaback is redirected to the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_UART_UnRegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to un-register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_RegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterRxEventCallback (UART_HandleTypeDef * huart, pUART_RxEventCallbackTypeDef pCallback)

Function description

Register a User UART Rx Event Callback To be used instead of the weak predefined callback.

Parameters

- **huart:** Uart handle
- **pCallback:** Pointer to the Rx Event Callback function

Return values

- **HAL:** status

HAL_UART_UnRegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterRxEventCallback (UART_HandleTypeDef * huart)

Function description

UnRegister the UART Rx Event Callback UART Rx Event Callback is redirected to the weak HAL_UARTEx_RxEventCallback() predefined callback.

Parameters

- **huart:** Uart handle

Return values

- **HAL:** status

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.
- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

Pause the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

Resume the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

Stop the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_Abort

Function name

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name

void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)

Function description

Handle UART interrupt request.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxHalfCpltCallback

Function name

void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxCpltCallback

Function name

void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

UART error callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Receive Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxEventCallback

Function name

void HAL_UARTEx_RxEventCallback (UART_HandleTypeDef * huart, uint16_t Size)

Function description

Reception Event Callback (Rx event notification called after use of advanced reception service).

Parameters

- **huart:** UART handle
- **Size:** Number of data available in application reception buffer (indicates a position in reception buffer until which, data are available)

Return values

- **None:**

HAL_UART_ReceiverTimeout_Config

Function name

void HAL_UART_ReceiverTimeout_Config (UART_HandleTypeDef * huart, uint32_t TimeoutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **TimeoutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFF.

Return values

- **None:**

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Disable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (const UART_HandleTypeDef * huart)

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (const UART_HandleTypeDef * huart)

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_InitCallbacksToDefault

Function name

void UART_InitCallbacksToDefault (UART_HandleTypeDef * huart)

Function description

Initialize the callbacks to their default values.

Parameters

- **huart:** UART handle.

Return values

- **none:**

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

This function handles UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** The actual Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

UART_Start_Receive_IT

Function name

HAL_StatusTypeDef UART_Start_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in Interrupt mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

UART_Start_Receive_DMA

Function name

HAL_StatusTypeDef UART_Start_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in DMA mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

79.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

79.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1

fclk_pres = fclk

UART_PRESCALER_DIV2

fclk_pres = fclk/2

UART_PRESCALER_DIV4

fclk_pres = fclk/4

UART_PRESCALER_DIV6

fclk_pres = fclk/6

UART_PRESCALER_DIV8

fclk_pres = fclk/8

UART_PRESCALER_DIV10

fclk_pres = fclk/10

UART_PRESCALER_DIV12

fclk_pres = fclk/12

UART_PRESCALER_DIV16

fclk_pres = fclk/16

UART_PRESCALER_DIV32

fclk_pres = fclk/32

UART_PRESCALER_DIV64

fclk_pres = fclk/64

UART_PRESCALER_DIV128

fclk_pres = fclk/128

UART_PRESCALER_DIV256

fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Error Definition

HAL_UART_ERROR_NONE

No error

HAL_UART_ERROR_PE

Parity error

HAL_UART_ERROR_NE

Noise error

HAL_UART_ERROR_FE

Frame error

HAL_UART_ERROR_ORE

Overrun error

HAL_UART_ERROR_DMA

DMA transfer error

HAL_UART_ERROR_RTO

Receiver Timeout error

HAL_UART_ERROR_INVALID_CALLBACK

Invalid Callback error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_RTOF Receiver Timeout clear flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_TXFECF

Description:

- Clear the UART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_WUF Wake up from stop mode flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_RTOF Receiver Timeout flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_RTOF Receiver timeout clear flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST Send Break Request
  - UART_MUTE_MODE_REQUEST Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_REACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_WUF

UART wake-up from stop mode flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK

UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TXFNF

UART TX FIFO not full interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_RXFNE

UART RXFIFO not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_WUF

UART wake-up from stop mode interruption

UART_IT_RXFF

UART RXFIFO full interruption

UART_IT_TXFE

UART TXFIFO empty interruption

UART_IT_RXFT

UART RXFIFO threshold reached interruption

UART_IT_TXFT

UART TXFIFO threshold reached interruption

UART_IT_RTO

UART receiver timeout interruption

UART_IT_ERR

UART error interruption

UART_IT_ORE

UART overrun error interruption

UART_IT_NE

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise Error detected Clear Flag

UART_CLEAR_OREF

Overrun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_WUF

Wake Up from stop mode Clear Flag

UART_CLEAR_RTOF

UART receiver timeout clear flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

Local Interconnect Network disable

UART_LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX

RX mode

UART_MODE_TX

TX mode

UART_MODE_TX_RX

RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity

UART_PARITY_NONE

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Receiver Timeout

UART_RECEIVER_TIMEOUT_DISABLE

UART Receiver Timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

UART Receiver Timeout enable

UART Reception type values

HAL_UART_RECEPTION_STANDARD

Standard reception

HAL_UART_RECEPTION_TOIDLE

Reception till completion or IDLE event

HAL_UART_RECEPTION_TORTO

Reception till completion or RTO event

HAL_UART_RECEPTION_TOCHARMATCH

Reception till completion or CM event

UART Request Parameters

UART_AUTOBAUD_REQUEST

Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

Send Break Request

UART_MUTE_MODE_REQUEST

Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART RxEvent type values

HAL_UART_RXEVENT_TC

RxEvent linked to Transfer Complete event

HAL_UART_RXEVENT_HT

RxEvent linked to Half Transfer event

HAL_UART_RXEVENT_IDLE

RxEvent linked to IDLE event

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State

UART_STATE_DISABLE

UART disabled

UART_STATE_ENABLE

UART enabled

UART State Code Definition

HAL_UART_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_UART_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState.Value is result of combination (Or) between gState and RxState values

HAL_UART_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_UART_STATE_ERROR

Error Value is allowed for gState only

UART Number of Stop Bits

UART_STOPBITS_0_5

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART wake-up on address

UART_WAKEUP_ON_STARTBIT

UART wake-up on start bit

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

80 HAL UART Extension Driver

80.1 UARTEx Firmware driver registers structures

80.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32h5xx_hal_uart_ex.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

80.2 UARTEx Firmware driver API description

The following section lists the various functions of the UARTEx library.

80.2.1 UART peripheral extended features

80.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- *HAL_RS485Ex_Init()*

80.2.3 IO operation functions

This section contains the following APIs:

- *HAL_UARTEx_WakeupCallback()*
- *HAL_UARTEx_RxFifoFullCallback()*
- *HAL_UARTEx_TxFifoEmptyCallback()*

80.2.4 Peripheral Control functions

This section provides the following functions:

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_UARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_UARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_UARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown).

1. Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller : (+) Detection of inactivity period (RX line has not been active for a given period).
   - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
   - RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte. (+) Detection that a specific character has been received.
2. There are two mode of transfer: (+) Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer. (+) Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected.
3. Blocking mode API: (+) HAL_UARTEx_ReceiveToIdle()
4. Non-Blocking mode API with Interrupt: (+) HAL_UARTEx_ReceiveToIdle_IT()
5. Non-Blocking mode API with DMA: (+) HAL_UARTEx_ReceiveToIdle_DMA()

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown). (#) Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller :

- Detection of inactivity period (RX line has not been active for a given period).
  - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
  - RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte.
- Detection that a specific character has been received. (#) There are two mode of transfer:

- Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer.
- Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected. (#) Blocking mode API:
- HAL_UARTEx_ReceiveToIdle() (#) Non-Blocking mode API with Interrupt:
- HAL_UARTEx_ReceiveToIdle_IT() (#) Non-Blocking mode API with DMA:
- HAL_UARTEx_ReceiveToIdle_DMA()

This section contains the following APIs:

- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_EnableFifoMode()***
- ***HAL_UARTEx_DisableFifoMode()***
- ***HAL_UARTEx_SetTxFifoThreshold()***
- ***HAL_UARTEx_SetRxFifoThreshold()***
- ***HAL_UARTEx_ReceiveToIdle()***
- ***HAL_UARTEx_ReceiveToIdle_IT()***
- ***HAL_UARTEx_ReceiveToIdle_DMA()***
- ***HAL_UARTEx_GetRxEventType()***

80.2.5 Detailed description of functions

HAL_RS485Ex_Init

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **Polarity:** Select the driver enable polarity. This parameter can be one of the following values:
  - UART_DE_POLARITY_HIGH DE signal is active high
  - UART_DE_POLARITY_LOW DE signal is active low
- **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)
- **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

Return values

- **HAL:** status

HAL_UARTEx_WakeupCallback

Function name

void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)

Function description

UART wakeup from Stop mode callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxFifoFullCallback

Function name

void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)

Function description

UART RX Fifo full callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_TxFifoEmptyCallback

Function name

void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)

Function description

UART TX Fifo empty callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)

Function description

Set Wakeup from Stop mode interrupt flag selection.

Parameters

- **huart:** UART handle.
- **WakeUpSelection:** Address match, Start Bit detection or RXNE/RXFNE bit status. This parameter can be one of the following values:
  - UART_WAKEUP_ON_ADDRESS
  - UART_WAKEUP_ON_STARTBIT
  - UART_WAKEUP_ON_READDATA_NONEMPTY

Return values

- **HAL:** status

Notes

- It is the application responsibility to enable the interrupt used as usart_wkup interrupt source before entering low-power mode.

HAL_UARTEx_EnableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)

Function description

Enable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)

Function description

Disable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name

HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)

Function description

By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).

Parameters

- **huart:** UART handle.
- **AddressLength:** This parameter can be one of the following values:
  - UART_ADDRESS_DETECT_4B 4-bit long address
  - UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address

Return values

- **HAL:** status

Notes

- Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)

Function description

Enable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)

Function description

Disable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - UART_TXFIFO_THRESHOLD_1_8
  - UART_TXFIFO_THRESHOLD_1_4
  - UART_TXFIFO_THRESHOLD_1_2
  - UART_TXFIFO_THRESHOLD_3_4
  - UART_TXFIFO_THRESHOLD_7_8
  - UART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - UART_RXFIFO_THRESHOLD_1_8
  - UART_RXFIFO_THRESHOLD_1_4
  - UART_RXFIFO_THRESHOLD_1_2
  - UART_RXFIFO_THRESHOLD_3_4
  - UART_RXFIFO_THRESHOLD_7_8
  - UART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_ReceiveToIdle

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint16_t * RxLen, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.
- **RxLen:** Number of data elements finally received (could be lower than Size, in case reception ends on IDLE event)
- **Timeout:** Timeout duration expressed in ms (covers the whole reception sequence).

Return values

- **HAL:** status

Notes

- HAL_OK is returned if reception is completed (expected number of data has been received) or if reception is stopped after IDLE event (less than the expected number of data has been received) In this case, RxLen output parameter indicates number of data available in reception buffer.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UARTEx_ReceiveToIdle_IT

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to UART interrupts raised by RXNE and IDLE events. Callback is called at end of reception indicating number of received data elements.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_ReceiveToIdle_DMA

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to DMA services, transferring automatically received data elements in user reception buffer and calling registered callbacks at half/end of reception. UART IDLE events are also used to consider reception phase as ended. In all cases, callback execution will indicate number of received data elements.
- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_GetRxEventType

Function name

HAL_UART_RxEventTypeTypeDef HAL_UARTEx_GetRxEventType (const UART_HandleTypeDef * huart)

Function description

Provide Rx Event type that has lead to RxEvent callback execution.

Parameters

- **huart:** UART handle.

Return values

- **Rx:** Event Type (return vale will be a value of UART RxEvent type values)

Notes

- When HAL_UARTEx_ReceiveToIdle_IT() or HAL_UARTEx_ReceiveToIdle_DMA() API are called, progress of reception process is provided to application through calls of Rx Event callback (either default one HAL_UARTEx_RxEventCallback() or user registered one). As several types of events could occur (IDLE event, Half Transfer, or Transfer Complete), this function allows to retrieve the Rx Event type that has lead to Rx Event callback execution.
- This function is expected to be called within the user implementation of Rx Event Callback, in order to provide the accurate value : In Interrupt Mode : HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one) In DMA Mode :HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_HT : when half of expected nb of data has been receivedHAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one). In DMA mode, RxEvent callback could be called several times; When DMA is configured in Normal Mode, HT event does not stop Reception process; When DMA is configured in Circular Mode, HT, TC or IDLE events don't stop Reception process;

80.3 UARTEx Firmware driver defines

The following section lists the various define and macros of the module.

80.3.1 UARTEx

UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8

RX FIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4

RX FIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2

RX FIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4

RX FIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8

RX FIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8

RX FIFO becomes full

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8

TX FIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4

TX FIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TX FIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TX FIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TX FIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TX FIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

81 HAL USART Generic Driver

81.1 USART Firmware driver registers structures

81.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32h5xx_hal_usart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.
  Note:
  – Oversampling by 8 is systematically applied to achieve high baud rates.
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.
- ***uint32_t USART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

81.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32h5xx_hal_usart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t SlaveMode***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***const uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution

- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Enable/Disable USART SPI Slave Mode. This parameter can be a value of ***USARTEx_Slave_Mode***
- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code
- ***void(* __USART_HandleTypeDef::TxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Half Complete Callback
- ***void(* __USART_HandleTypeDef::TxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Complete Callback
- ***void(* __USART_HandleTypeDef::RxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Half Complete Callback
- ***void(* __USART_HandleTypeDef::RxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Complete Callback
- ***void(* __USART_HandleTypeDef::TxRxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Rx Complete Callback
- ***void(* __USART_HandleTypeDef::ErrorCallback)(struct __USART_HandleTypeDef *husart)***
  USART Error Callback
- ***void(* __USART_HandleTypeDef::AbortCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Abort Complete Callback
- ***void(* __USART_HandleTypeDef::RxFifoFullCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Fifo Full Callback
- ***void(* __USART_HandleTypeDef::TxFifoEmptyCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Fifo Empty Callback
- ***void(* __USART_HandleTypeDef::MspInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp Init callback
- ***void(* __USART_HandleTypeDef::MspDeInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp DeInit callback

81.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

81.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).

2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure these USART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - USART interrupts handling:

Note: The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.

   - DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

Note: To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.

81.2.2 Callback registration

The compilation define USE_HAL_USART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_USART_RegisterCallback() to register a user callback. Function HAL_USART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_USART_UnRegisterCallback() to reset a callback to the default weak function. HAL_USART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.

- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit.

By default, after the HAL_USART_Init() and when the state is HAL_USART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_USART_TxCpltCallback(), HAL_USART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_USART_Init() and HAL_USART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_USART_Init() and HAL_USART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_USART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_USART_STATE_READY or HAL_USART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_USART_RegisterCallback() before calling HAL_USART_DeInit() or HAL_USART_Init() function.

When The compilation define USE_HAL_USART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

81.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***
- ***HAL_USART_RegisterCallback()***
- ***HAL_USART_UnRegisterCallback()***

81.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected
2. Blocking mode API's are :
   - HAL_USART_Transmit() in simplex mode
   - HAL_USART_Receive() in full duplex receive only
   - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   - HAL_USART_Transmit_IT() in simplex mode
   - HAL_USART_Receive_IT() in full duplex receive only
   - HAL_USART_TransmitReceive_IT() in full duplex mode
   - HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
   - HAL_USART_Transmit_DMA() in simplex mode
   - HAL_USART_Receive_DMA() in full duplex receive only
   - HAL_USART_TransmitReceive_DMA() in full duplex mode
   - HAL_USART_DMAPause()
   - HAL_USART_DMAResume()
   - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   - HAL_USART_TxCpltCallback()
   - HAL_USART_RxCpltCallback()
   - HAL_USART_TxHalfCpltCallback()
   - HAL_USART_RxHalfCpltCallback()
   - HAL_USART_ErrorCallback()
   - HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_USART_Abort()
   - HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
   - HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   - Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_USART_Transmit()***
- ***HAL_USART_Receive()***
- ***HAL_USART_TransmitReceive()***
- ***HAL_USART_Transmit_IT()***
- ***HAL_USART_Receive_IT()***

- *HAL_USART_TransmitReceive_IT()*
- *HAL_USART_Transmit_DMA()*
- *HAL_USART_Receive_DMA()*
- *HAL_USART_TransmitReceive_DMA()*
- *HAL_USART_DMAPause()*
- *HAL_USART_DMAResume()*
- *HAL_USART_DMAStop()*
- *HAL_USART_Abort()*
- *HAL_USART_Abort_IT()*
- *HAL_USART_IRQHandler()*
- *HAL_USART_TxCpltCallback()*
- *HAL_USART_TxHalfCpltCallback()*
- *HAL_USART_RxCpltCallback()*
- *HAL_USART_RxHalfCpltCallback()*
- *HAL_USART_TxRxCpltCallback()*
- *HAL_USART_ErrorCallback()*
- *HAL_USART_AbortCpltCallback()*

81.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- *HAL_USART_GetState()*
- *HAL_USART_GetError()*

81.2.6 Detailed description of functions

HAL_USART_Init

Function name

HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)

Function description

Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DeInit

Function name

HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART peripheral.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_MspInit

Function name

void HAL_USART_MspInit (USART_HandleTypeDef * husart)

Function description

Initialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_MspDeInit

Function name

void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_RegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID, pUSART_CallbackTypeDef pCallback)

Function description

Register a User USART Callback To be used to override the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status +

Notes

- The HAL_USART_RegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_UnRegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID)

Function description

Unregister an USART Callback USART callaback is redirected to the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_USART_UnRegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to un-register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_Transmit

Function name

HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function description

Simplex send an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive

Function name

HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pRxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Send and Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_IT

Function name

HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Send and Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received/sent.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_DMAPause

Function name

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

Pause the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAResume

Function name

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

Resume the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAStop

Function name

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

Stop the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_Abort

Function name

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

Handle USART interrupt request.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

USART error callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

USART Abort Complete callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (const USART_HandleTypeDef * husart)

Function description

Return the USART handle state.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle state

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (const USART_HandleTypeDef * husart)

Function description

Return the USART error code.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle Error Code

81.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

81.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1

fclk_pres = fclk

USART_PRESCALER_DIV2

fclk_pres = fclk/2

USART_PRESCALER_DIV4

fclk_pres = fclk/4

USART_PRESCALER_DIV6

fclk_pres = fclk/6

USART_PRESCALER_DIV8

fclk_pres = fclk/8

USART_PRESCALER_DIV10

fclk_pres = fclk/10

USART_PRESCALER_DIV12

fclk_pres = fclk/12

USART_PRESCALER_DIV16

fclk_pres = fclk/16

USART_PRESCALER_DIV32

fclk_pres = fclk/32

USART_PRESCALER_DIV64

fclk_pres = fclk/64

USART_PRESCALER_DIV128

fclk_pres = fclk/128

USART_PRESCALER_DIV256

fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE

USART frame phase on first clock transition

USART_PHASE_2EDGE

USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW

Driver enable signal is active high

USART_POLARITY_HIGH

Driver enable signal is active low

USART Error Definition

HAL_USART_ERROR_NONE

No error

HAL_USART_ERROR_PE

Parity error

HAL_USART_ERROR_NE

Noise error

HAL_USART_ERROR_FE

Frame error

HAL_USART_ERROR_ORE

Overrun error

HAL_USART_ERROR_DMA

DMA transfer error

HAL_USART_ERROR_UDR

SPI slave underrun error

HAL_USART_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_USART_ERROR_RTO

Receiver Timeout error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_TXFT TXFIFO threshold flag
  - USART_FLAG_RXFT RXFIFO threshold flag
  - USART_FLAG_RXFF RXFIFO Full flag
  - USART_FLAG_TXFE TXFIFO Empty flag
  - USART_FLAG_REACK Receive enable acknowledge flag
  - USART_FLAG_TEACK Transmit enable acknowledge flag
  - USART_FLAG_BUSY Busy flag
  - USART_FLAG_UDR SPI slave underrun error flag
  - USART_FLAG_TXE Transmit data register empty flag
  - USART_FLAG_TXFNF TXFIFO not full flag
  - USART_FLAG_TC Transmission Complete flag
  - USART_FLAG_RXNE Receive data register not empty flag
  - USART_FLAG_RXFNE RXFIFO not empty flag
  - USART_FLAG_RTOF Receiver Timeout flag
  - USART_FLAG_IDLE Idle Line detection flag
  - USART_FLAG_ORE OverRun Error flag
  - USART_FLAG_NE Noise Error flag
  - USART_FLAG_FE Framing Error flag
  - USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag
  - USART_CLEAR_RTOF Receiver Timeout clear flag
  - USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_RTOF Receiver timeout clear flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_TXFT

USART TXFIFO threshold flag

USART_FLAG_RXFT

USART RXFIFO threshold flag

USART_FLAG_RXFF

USART RXFIFO Full flag

USART_FLAG_TXFE

USART TXFIFO Empty flag

USART_FLAG_REACK

USART receive enable acknowledge flag

USART_FLAG_TEACK

USART transmit enable acknowledge flag

USART_FLAG_BUSY

USART busy flag

USART_FLAG_UDR

SPI slave underrun error flag

USART_FLAG_TXE

USART transmit data register empty

USART_FLAG_TXFNF

USART TXFIFO not full

USART_FLAG_RTOF

USART receiver timeout flag

USART_FLAG_TC

USART transmission complete

USART_FLAG_RXNE

USART read data register not empty

USART_FLAG_RXFNE

USART RXFIFO not empty

USART_FLAG_IDLE

USART idle flag

USART_FLAG_ORE

USART overrun error

USART_FLAG_NE

USART noise error

USART_FLAG_FE

USART frame error

USART_FLAG_PE

USART parity error

USART Interruption Flags Mask

USART_IT_MASK

USART interruptions flags mask

USART_CR_MASK

USART control register mask

USART_CR_POS

USART control register position

USART_ISR_MASK

USART ISR register mask

USART_ISR_POS

USART ISR register position

USART Interrupts Definition

USART_IT_PE

USART parity error interruption

USART_IT_TXE

USART transmit data register empty interruption

USART_IT_TXFNF

USART TX FIFO not full interruption

USART_IT_TC

USART transmission complete interruption

USART_IT_RXNE

USART read data register not empty interruption

USART_IT_RXFNE

USART RXFIFO not empty interruption

USART_IT_IDLE

USART idle interruption

USART_IT_ERR

USART error interruption

USART_IT_ORE

USART overrun error interruption

USART_IT_NE

USART noise error interruption

USART_IT_FE

USART frame error interruption

USART_IT_RXFF

USART RXFIFO full interruption

USART_IT_TXFE

USART TXFIFO empty interruption

USART_IT_RXFT

USART RXFIFO threshold reached interruption

USART_IT_TXFT

USART TXFIFO threshold reached interruption

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise Error detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRF

SPI slave underrun error Clear Flag

USART_CLEAR_TXFECF

TXFIFO Empty Clear Flag

USART_CLEAR_RTOF

USART receiver timeout clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Parity

USART_PARITY_NONE

No parity

USART_PARITY_EVEN

Even parity

USART_PARITY_ODD

Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

82 HAL USART Extension Driver

82.1 USARTEx Firmware driver API description

The following section lists the various functions of the USARTEx library.

82.1.1 USART peripheral extended features

82.1.2 IO operation functions

This section contains the following APIs:

- *HAL_USARTEx_RxFifoFullCallback()*
- *HAL_USARTEx_TxFifoEmptyCallback()*

82.1.3 Peripheral Control functions

This section provides the following functions:

- HAL_USARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_USARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_USARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_USARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_USARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_USARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_USARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- *HAL_USARTEx_EnableSlaveMode()*
- *HAL_USARTEx_DisableSlaveMode()*
- *HAL_USARTEx_ConfigNSS()*
- *HAL_USARTEx_EnableFifoMode()*
- *HAL_USARTEx_DisableFifoMode()*
- *HAL_USARTEx_SetTxFifoThreshold()*
- *HAL_USARTEx_SetRxFifoThreshold()*

82.1.4 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name

void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)

Function description

USART RX Fifo full callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_TxFifoEmptyCallback

Function name

void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)

Function description

USART TX Fifo empty callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_EnableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)

Function description

Enable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- When the USART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device.
- In SPI slave mode, the USART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the USART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master.
- The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_USARTEx_DisableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)

Function description

Disable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_ConfigNSS

Function name

HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)

Function description

Configure the Slave Select input pin (NSS).

Parameters

- **husart:** USART handle.
- **NSSConfig:** NSS configuration. This parameter can be one of the following values:
  - USART_NSS_HARD
  - USART_NSS_SOFT

Return values

- **HAL:** status

Notes

- Software NSS management: SPI slave will always be selected and NSS input pin will be ignored.
- Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_USARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)

Function description

Enable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)

Function description

Disable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - USART_TXFIFO_THRESHOLD_1_8
  - USART_TXFIFO_THRESHOLD_1_4
  - USART_TXFIFO_THRESHOLD_1_2
  - USART_TXFIFO_THRESHOLD_3_4
  - USART_TXFIFO_THRESHOLD_7_8
  - USART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_USARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - USART_RXFIFO_THRESHOLD_1_8
  - USART_RXFIFO_THRESHOLD_1_4
  - USART_RXFIFO_THRESHOLD_1_2
  - USART_RXFIFO_THRESHOLD_3_4
  - USART_RXFIFO_THRESHOLD_7_8
  - USART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

82.2 USARTEx Firmware driver defines

The following section lists the various define and macros of the module.

82.2.1 USARTEx

USARTEx

USARTEx FIFO mode

USART_FIFOMODE_DISABLE

FIFO mode disable

USART_FIFOMODE_ENABLE

FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode enable

USART_SLAVEMODE_DISABLE

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD

SPI slave selection depends on NSS input pin

USART_NSS_SOFT

SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

83 HAL WWDG Generic Driver

83.1 WWDG Firmware driver registers structures

83.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32h5xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interrupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

83.1.2 __WWDG_HandleTypeDef

__WWDG_HandleTypeDef is defined in the stm32h5xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***WWDG_TypeDef* __WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef __WWDG_HandleTypeDef::Init***
  WWDG required parameters
- ***void(* __WWDG_HandleTypeDef::EwiCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Early WakeUp Interrupt callback
- ***void(* __WWDG_HandleTypeDef::MspInitCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Msp Init callback

83.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

83.2.1 WWDG Specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls down from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.

- Once enabled the WWDG cannot be disabled except by a system reset.
- If required by application, an Early Wakeup Interrupt can be triggered in order to be warned before WWDG expiration. The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches 0x40, interrupt occurs. This mechanism requires WWDG interrupt line to be enabled in NVIC. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- WWDGRST flag in RCC CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (mS) = 1000 * (T[5;0] + 1) / WWDG clock (Hz) where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  - min time (mS) = 1000 * (Counter - Window) / WWDG clock
  - max time (mS) = 1000 * (Counter - 0x40) / WWDG clock
- Typical values:
  - Counter min (T[5;0] = 0x00) at 56MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 73.14us
  - Counter max (T[5;0] = 0x3F) at 56MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 599.18ms

83.2.2 How to use this driver

Common driver usage

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Configure the WWDG prescaler, refresh window value, counter value and early interrupt status using HAL_WWDG_Init() function. This will automatically enable WWDG and start its downcounter. Time reference can be taken from function exit. Care must be taken to provide a counter value greater than 0x40 to prevent generation of immediate reset.
- If the Early Wakeup Interrupt (EWI) feature is enabled, an interrupt is generated when the counter reaches 0x40. When HAL_WWDG_IRQHandler is triggered by the interrupt service routine, flag will be automatically cleared and HAL_WWDG_WakeupCallback user callback will be executed. User can add his own code by customization of callback HAL_WWDG_WakeupCallback.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_WWDG_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

Callback registration

The compilation define USE_HAL_WWDG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_WWDG_RegisterCallback() to register a user callback.

- Function HAL_WWDG_RegisterCallback() allows to register following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
- Use function HAL_WWDG_UnRegisterCallback() to reset a callback to the default weak function. HAL_WWDG_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit.

When calling HAL_WWDG_Init function, callbacks are reset to the corresponding legacy weak functions: HAL_WWDG_EarlyWakeupCallback() and HAL_WWDG_MspInit() only if they have not been registered before.

When compilation define USE_HAL_WWDG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

WWDG HAL driver macros list

Below the list of available macros in WWDG HAL driver.

- __HAL_WWDG_ENABLE: Enable the WWDG peripheral
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags
- __HAL_WWDG_ENABLE_IT: Enable the WWDG early wakeup interrupt

83.2.3 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- ***HAL_WWDG_Init()***
- ***HAL_WWDG_MspInit()***
- ***HAL_WWDG_RegisterCallback()***
- ***HAL_WWDG_UnRegisterCallback()***

83.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- ***HAL_WWDG_Refresh()***
- ***HAL_WWDG_IRQHandler()***
- ***HAL_WWDG_EarlyWakeupCallback()***

83.2.5 Detailed description of functions

HAL_WWDG_Init

Function name

HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG according to the specified.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_MspInit

Function name

void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG MSP.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_RegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID, pWWDG_CallbackTypeDef pCallback)

Function description

Register a User WWDG Callback To be used instead of the weak predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_WWDG_MSPINIT_CB_ID MspInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_WWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_UnRegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID)

Function description

Unregister a WWDG Callback WWDG Callback is redirected to the weak predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_WWDG_MSPINIT_CB_ID MspInit callback ID

Return values

- **status:**

HAL_WWDG_Refresh

Function name

HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)

Function description

Refresh the WWDG.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_IRQHandler

Function name

void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)

Function description

Handle WWDG interrupt request.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name

void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)

Function description

WWDG Early Wakeup callback.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

83.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

83.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF

Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

84 HAL XSPI Generic Driver

84.1 XSPI Firmware driver registers structures

84.1.1 XSPI_InitTypeDef

XSPI_InitTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t FifoThresholdByte***
- ***uint32_t MemoryMode***
- ***uint32_t MemoryType***
- ***uint32_t MemorySize***
- ***uint32_t ChipSelectHighTimeCycle***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***
- ***uint32_t DelayBlockBypass***
- ***uint32_t Refresh***

Field Documentation

- ***uint32_t XSPI_InitTypeDef::FifoThresholdByte***
  This is the threshold used by the Peripheral to generate the interrupt indicating that data are available in reception or free place is available in transmission. For OCTOSPI, this parameter can be a value between 1 and 32
- ***uint32_t XSPI_InitTypeDef::MemoryMode***
  It Specifies the memory mode. This parameter can be a value of ***XSPI_MemoryMode***
- ***uint32_t XSPI_InitTypeDef::MemoryType***
  It indicates the external device type connected to the XSPI. This parameter can be a value of ***XSPI_MemoryType***
- ***uint32_t XSPI_InitTypeDef::MemorySize***
  It defines the size of the external device connected to the XSPI, it corresponds to the number of address bits required to access the external device. This parameter can be a value of ***XSPI_MemorySize***
- ***uint32_t XSPI_InitTypeDef::ChipSelectHighTimeCycle***
  It defines the minimum number of clocks which the chip select must remain high between commands. This parameter can be a value between 1 and 64U
- ***uint32_t XSPI_InitTypeDef::FreeRunningClock***
  It enables or not the free running clock. This parameter can be a value of ***XSPI_FreeRunningClock***
- ***uint32_t XSPI_InitTypeDef::ClockMode***
  It indicates the level of clock when the chip select is released. This parameter can be a value of ***XSPI_ClockMode***
- ***uint32_t XSPI_InitTypeDef::WrapSize***
  It indicates the wrap-size corresponding the external device configuration. This parameter can be a value of ***XSPI_WrapSize***
- ***uint32_t XSPI_InitTypeDef::ClockPrescaler***
  It specifies the prescaler factor used for generating the external clock based on the AHB clock. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_InitTypeDef::SampleShifting***
  It allows to delay to 1/2 cycle the data sampling in order to take in account external signal delays. This parameter can be a value of ***XSPI_SampleShifting***

- ***uint32_t XSPI_InitTypeDef::DelayHoldQuarterCycle***
  It allows to hold to 1/4 cycle the data. This parameter can be a value of ***XSPI_DelayHoldQuarterCycle***
- ***uint32_t XSPI_InitTypeDef::ChipSelectBoundary***
  It enables the transaction boundary feature and defines the boundary of bytes to release the chip select. This parameter can be a value of ***XSPI_ChipSelectBoundary***
- ***uint32_t XSPI_InitTypeDef::DelayBlockBypass***
  It enables the delay block bypass, so the sampling is not affected by the delay block. This parameter can be a value of ***XSPI_DelayBlockBypass***
- ***uint32_t XSPI_InitTypeDef::Refresh***
  It enables the refresh rate feature. The chip select is released every Refresh+1 clock cycles. This parameter can be a value between 0 and 0xFFFFFFFF

84.1.2 __XSPI_HandleTypeDef

__XSPI_HandleTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***XSPI_TypeDef * Instance***
- ***XSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* StatusMatchCallback***
- ***void(* TimeOutCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***XSPI_TypeDef* __XSPI_HandleTypeDef::Instance***
  XSPI registers base address
- ***XSPI_InitTypeDef __XSPI_HandleTypeDef::Init***
  XSPI initialization parameters
- ***uint8_t* __XSPI_HandleTypeDef::pBuffPtr***
  Address of the XSPI buffer for transfer
- ***__IO uint32_t __XSPI_HandleTypeDef::XferSize***
  Number of data to transfer
- ***__IO uint32_t __XSPI_HandleTypeDef::XferCount***
  Counter of data transferred
- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmatx***
  Handle of the DMA channel used for transmit
- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmarx***
  Handle of the DMA channel used for receive

- ***__IO uint32_t __XSPI_HandleTypeDef::State***
  Internal state of the XSPI HAL driver
- ***__IO uint32_t __XSPI_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error
- ***uint32_t __XSPI_HandleTypeDef::Timeout***
  Timeout used for the XSPI external device access
- ***void(* __XSPI_HandleTypeDef::ErrorCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::AbortCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::FifoThresholdCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::CmdCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::StatusMatchCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TimeOutCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspDeInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***

84.1.3 XSPI_RegularCmdTypeDef

XSPI_RegularCmdTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t IOSelect***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionWidth***
- ***uint32_t InstructionDTRMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressWidth***
- ***uint32_t AddressDTRMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesWidth***
- ***uint32_t AlternateBytesDTRMode***
- ***uint32_t DataMode***
- ***uint32_t DataLength***
- ***uint32_t DataDTRMode***
- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t XSPI_RegularCmdTypeDef::OperationType***
  It indicates if the configuration applies to the common registers or to the registers for the write operation (these registers are only used for memory-mapped mode). This parameter can be a value of ***XSPI_OperationType***
- ***uint32_t XSPI_RegularCmdTypeDef::IOSelect***
  It indicates the IOs used to exchange data with external memory. This parameter can be a value of ***XSPI_IOSelect***

- ***uint32_t XSPI_RegularCmdTypeDef::Instruction***
  It contains the instruction to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionMode***
  It indicates the mode of the instruction. This parameter can be a value of ***XSPI_InstructionMode***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionWidth***
  It indicates the width of the instruction. This parameter can be a value of ***XSPI_InstructionWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionDTRMode***
  It enables or not the DTR mode for the instruction phase. This parameter can be a value of ***XSPI_InstructionDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_RegularCmdTypeDef::AddressMode***
  It indicates the address mode. Address mode precises number of lines for address (except no address). This parameter can be a value of ***XSPI_AddressMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressDTRMode***
  It enables or not the DTR mode for the address phase. This parameter can be a value of ***XSPI_AddressDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytes***
  It contains the alternate bytes to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesMode***
  It indicates the mode of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesWidth***
  It indicates the width of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesDTRMode***
  It enables or not the DTR mode for the alternate bytes phase. This parameter can be a value of ***XSPI_AlternateBytesDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataMode***
  It indicates the data mode. Data mode precises number of lines for data exchange (except no data). This parameter can be a value of ***XSPI_DataMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::DataDTRMode***
  It enables or not the DTR mode for the data phase. This parameter can be a value of ***XSPI_DataDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DummyCycles***
  It indicates the number of dummy cycles inserted before data phase. This parameter can be a value between 0 and 31U
- ***uint32_t XSPI_RegularCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***
- ***uint32_t XSPI_RegularCmdTypeDef::SIOOMode***
  It enables or not the SIOO mode. When SIOO mode enabled, instruction will be sent only once. This parameter can be a value of ***XSPI_SIOOMode***

84.1.4 XSPI_HyperbusCfgTypeDef

XSPI_HyperbusCfgTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t RWRecoveryTimeCycle***
- ***uint32_t AccessTimeCycle***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCfgTypeDef::RWRecoveryTimeCycle***
  It indicates the number of cycles for the device read write recovery time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::AccessTimeCycle***
  It indicates the number of cycles for the device access time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::WriteZeroLatency***
  It enables or not the latency for the write access. This parameter can be a value of ***XSPI_WriteZeroLatency***
- ***uint32_t XSPI_HyperbusCfgTypeDef::LatencyMode***
  It configures the latency mode. This parameter can be a value of ***XSPI_LatencyMode***

84.1.5 XSPI_HyperbusCmdTypeDef

XSPI_HyperbusCmdTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressWidth***
- ***uint32_t DataLength***
- ***uint32_t DQSMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressSpace***
  It indicates the address space accessed by the command. This parameter can be a value of ***XSPI_AddressSpace***
- ***uint32_t XSPI_HyperbusCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_HyperbusCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF In case of autopolling mode, this parameter can be any value between 1 and 4
- ***uint32_t XSPI_HyperbusCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***

84.1.6 XSPI_AutoPollingTypeDef

XSPI_AutoPollingTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t MatchValue***
- ***uint32_t MatchMask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t IntervalTime***

Field Documentation

- ***uint32_t XSPI_AutoPollingTypeDef::MatchValue***
  Specifies the value to be compared with the masked status register to get a match. This parameter can be any value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_AutoPollingTypeDef::MatchMask***
  Specifies the mask to be applied to the status bytes received. This parameter can be any value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_AutoPollingTypeDef::MatchMode***
  Specifies the method used for determining a match. This parameter can be a value of ***XSPI_MatchMode***

- ***uint32_t XSPI_AutoPollingTypeDef::AutomaticStop***
  Specifies if automatic polling is stopped after a match. This parameter can be a value of ***XSPI_AutomaticStop***
- ***uint32_t XSPI_AutoPollingTypeDef::IntervalTime***
  Specifies the number of clock cycles between two read during automatic polling phases. This parameter can be any value between 0 and 0xFFFFU

84.1.7 XSPI_MemoryMappedTypeDef

XSPI_MemoryMappedTypeDef is defined in the stm32h5xx_hal_xspi.h

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeoutPeriodClock***

Field Documentation

- ***uint32_t XSPI_MemoryMappedTypeDef::TimeOutActivation***
  Specifies if the timeout counter is enabled to release the chip select. This parameter can be a value of ***XSPI_TimeOutActivation***
- ***uint32_t XSPI_MemoryMappedTypeDef::TimeoutPeriodClock***
  Specifies the number of clock to wait when the FIFO is full before to release the chip select. This parameter can be any value between 0 and 0xFFFFU

84.2 XSPI Firmware driver API description

The following section lists the various functions of the XSPI library.

84.2.1 How to use this driver

Initialization

As prerequisite, fill in the HAL_XSPI_MspInit() :

- Enable OctoSPI clocks interface with __HAL_RCC_XSPI_CLK_ENABLE().
- Reset OctoSPI Peripheral with __HAL_RCC_XSPI_FORCE_RESET() and __HAL_RCC_XSPI_RELEASE_RESET().
- Enable the clocks for the OctoSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
- Configure these OctoSPI pins in alternate mode using HAL_GPIO_Init().
- If interrupt or DMA mode is used, enable and configure OctoSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
- If DMA mode is used, enable the clocks for the OctoSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with OctoSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

Configure the fifo threshold, the memory mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_XSPI_Init() function.

When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode using the HAL_XSPI_HyperbusCfg() function.

Indirect functional mode

In regular mode, configure the command sequence using the HAL_XSPI_Command() or HAL_XSPI_Command_IT() functions :

- Instruction phase : the mode used and if present the size, the instruction opcode and the DTR mode.
- Address phase : the mode used and if present the size, the address value and the DTR mode.
- Alternate-bytes phase : the mode used and if present the size, the alternate bytes values and the DTR mode.
- Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
- Data phase : the mode used and if present the number of bytes and the DTR mode.

- Data strobe (DQS) mode : the activation (or not) of this mode
- Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
- IO selection : to access external memory.
- Operation type : always common configuration.

In Hyperbus mode, configure the command sequence using the HAL_XSPI_HyperbusCmd() function :

- Address space : indicate if the access will be done in register or memory
- Address size
- Number of data
- Data strobe (DQS) mode : the activation (or not) of this mode

If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_CmdCpltCallback() will be called when the transfer is complete.

For the indirect write mode, use HAL_XSPI_Transmit(), HAL_XSPI_Transmit_DMA() or HAL_XSPI_Transmit_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.

For the indirect read mode, use HAL_XSPI_Receive(), HAL_XSPI_Receive_DMA() or HAL_XSPI_Receive_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

Configure the command sequence by the same way than the indirect mode

Configure the auto-polling functional mode using the HAL_XSPI_AutoPolling() or HAL_XSPI_AutoPolling_IT() functions :

- The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.

After the configuration :

- In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
- In interrupt mode, HAL_XSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode :

- Operation type equals to read configuration : the command configuration applies to read access in memory-mapped mode
- Operation type equals to write configuration : the command configuration applies to write access in memory-mapped mode
- Both read and write configuration should be performed before activating memory-mapped mode

Configure the memory-mapped functional mode using the HAL_XSPI_MemoryMapped() functions :

- The timeout activation and the timeout period.

After the configuration, the OctoSPI will be used as soon as an access on the AHB is done on the address range. HAL_XSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

HAL_XSPI_GetError() function gives the error raised during the last operation.

HAL_XSPI_Abort() and HAL_XSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :

- In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
- In interrupt mode, HAL_XSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

HAL_XSPI_GetState() function gives the current state of the HAL XSPI driver.

HAL_XSPI_SetTimeout() function configures the timeout value used in the driver.

HAL_XSPI_SetFifoThreshold() function configures the threshold on the Fifo of the OctoSPI Peripheral.

HAL_XSPI_SetMemoryType() function configures the type of the external memory.

HAL_XSPI_SetDeviceSize() function configures the size of the external memory.

HAL_XSPI_SetClockPrescaler() function configures the clock prescaler of the OctoSPI Peripheral.

HAL_XSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

Delay Block functions

The delay block (DLYB) is used to generate an output clock that is dephased from the input clock.

- The delay line length can be Configure to one period of the Input clock with HAL_XSPI_DLYB_GetClockPeriod().
- The phase of the output clock can be programmed directly with HAL_XSPI_DLYB_SetConfig().
- The phase of the output clock can be got with HAL_XSPI_DLYB_GetConfig().

Callback registration

The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_XSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_XSPI_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.

- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function) takes as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_XSPI_Init() and if the state is HAL_XSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_XSPI_Init() and HAL_XSPI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_XSPI_Init() and HAL_XSPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_XSPI_RegisterCallback() before calling HAL_XSPI_DeInit() or HAL_XSPI_Init() function.

When The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

84.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the XSPI.
- De-initialize the XSPI.

This section contains the following APIs:

- ***HAL_XSPI_Init()***
- ***HAL_XSPI_MspInit()***
- ***HAL_XSPI_DeInit()***
- ***HAL_XSPI_MspDeInit()***

84.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence (regular and Hyperbus).
- Handle the Hyperbus configuration.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- ***HAL_XSPI_IRQHandler()***
- ***HAL_XSPI_Command()***
- ***HAL_XSPI_Command_IT()***
- ***HAL_XSPI_HyperbusCfg()***
- ***HAL_XSPI_HyperbusCmd()***
- ***HAL_XSPI_Transmit()***
- ***HAL_XSPI_Receive()***
- ***HAL_XSPI_Transmit_IT()***
- ***HAL_XSPI_Receive_IT()***
- ***HAL_XSPI_Transmit_DMA()***
- ***HAL_XSPI_Receive_DMA()***

- *HAL_XSPI_AutoPolling()*
- *HAL_XSPI_AutoPolling_IT()*
- *HAL_XSPI_MemoryMapped()*
- *HAL_XSPI_ErrorCallback()*
- *HAL_XSPI_AbortCpltCallback()*
- *HAL_XSPI_FifoThresholdCallback()*
- *HAL_XSPI_CmdCpltCallback()*
- *HAL_XSPI_RxCpltCallback()*
- *HAL_XSPI_TxCpltCallback()*
- *HAL_XSPI_RxHalfCpltCallback()*
- *HAL_XSPI_TxHalfCpltCallback()*
- *HAL_XSPI_StatusMatchCallback()*
- *HAL_XSPI_TimeOutCallback()*
- *HAL_XSPI_RegisterCallback()*
- *HAL_XSPI_UnRegisterCallback()*

84.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.
- Manage the Fifo threshold.
- Configure the timeout duration used in the driver.

This section contains the following APIs:

- *HAL_XSPI_Abort()*
- *HAL_XSPI_Abort_IT()*
- *HAL_XSPI_SetFifoThreshold()*
- *HAL_XSPI_GetFifoThreshold()*
- *HAL_XSPI_SetMemoryType()*
- *HAL_XSPI_SetDeviceSize()*
- *HAL_XSPI_SetClockPrescaler()*
- *HAL_XSPI_SetTimeout()*
- *HAL_XSPI_GetError()*
- *HAL_XSPI_GetState()*

84.2.5 Delay Block function

This subsection provides a set of functions allowing to :

- Configure the delay block.

This section contains the following APIs:

- *HAL_XSPI_DLYB_SetConfig()*
- *HAL_XSPI_DLYB_GetConfig()*
- *HAL_XSPI_DLYB_GetClockPeriod()*

84.2.6 Detailed description of functions

HAL_XSPI_Init

Function name

HAL_StatusTypeDef HAL_XSPI_Init (XSPI_HandleTypeDef * hxspi)

Function description

Initialize the XSPI mode according to the specified parameters in the XSPI_InitTypeDef and initialize the associated handle.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_MspInit

Function name

void HAL_XSPI_MspInit (XSPI_HandleTypeDef * hxspi)

Function description

Initialize the XSPI MSP.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_DeInit

Function name

HAL_StatusTypeDef HAL_XSPI_DeInit (XSPI_HandleTypeDef * hxspi)

Function description

De-Initialize the XSPI peripheral.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_MspDeInit

Function name

void HAL_XSPI_MspDeInit (XSPI_HandleTypeDef * hxspi)

Function description

DeInitialize the XSPI MSP.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_IRQHandler

Function name

void HAL_XSPI_IRQHandler (XSPI_HandleTypeDef * hxspi)

Function description

Handle XSPI interrupt request.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_Command

Function name

HAL_StatusTypeDef HAL_XSPI_Command (XSPI_HandleTypeDef * hxspi, XSPI_RegularCmdTypeDef *const pCmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Command_IT (XSPI_HandleTypeDef * hxspi, XSPI_RegularCmdTypeDef *const pCmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_XSPI_HyperbusCfg

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCfg (XSPI_HandleTypeDef * hxspi, XSPI_HyperbusCfgTypeDef *const pCfg, uint32_t Timeout)

Function description

Configure the Hyperbus parameters.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to Structure containing the Hyperbus configuration
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_HyperbusCmd

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCmd (XSPI_HandleTypeDef * hxspi, XSPI_HyperbusCmdTypeDef *const pCmd, uint32_t Timeout)

Function description

Set the Hyperbus command configuration.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : Structure containing the Hyperbus command
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_Transmit

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit (XSPI_HandleTypeDef * hxspi, uint8_t *const pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_XSPI_Receive

Function name

HAL_StatusTypeDef HAL_XSPI_Receive (XSPI_HandleTypeDef * hxspi, uint8_t *const pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_XSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_IT (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_XSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_IT (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_XSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_DMA (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_XSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_DMA (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_XSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling (XSPI_HandleTypeDef * hxspi, XSPI_AutoPollingTypeDef *const pCfg, uint32_t Timeout)

Function description

Configure the XSPI Automatic Polling Mode in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_XSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling_IT (XSPI_HandleTypeDef * hxspi, XSPI_AutoPollingTypeDef *const pCfg)

Function description

Configure the XSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_XSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_XSPI_MemoryMapped (XSPI_HandleTypeDef * hxspi, XSPI_MemoryMappedTypeDef *const pCfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_XSPI_ErrorCallback

Function name

void HAL_XSPI_ErrorCallback (XSPI_HandleTypeDef * hxspi)

Function description

Transfer Error callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_AbortCpltCallback

Function name

void HAL_XSPI_AbortCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Abort completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_FifoThresholdCallback

Function name

void HAL_XSPI_FifoThresholdCallback (XSPI_HandleTypeDef * hxspi)

Function description

FIFO Threshold callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_CmdCpltCallback

Function name

void HAL_XSPI_CmdCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Command completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RxCpltCallback

Function name

void HAL_XSPI_RxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Rx Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TxCpltCallback

Function name

void HAL_XSPI_TxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Tx Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RxHalfCpltCallback

Function name

void HAL_XSPI_RxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TxHalfCpltCallback

Function name

void HAL_XSPI_TxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_StatusMatchCallback

Function name

void HAL_XSPI_StatusMatchCallback (XSPI_HandleTypeDef * hxspi)

Function description

Status Match callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TimeOutCallback

Function name

void HAL_XSPI_TimeOutCallback (XSPI_HandleTypeDef * hxspi)

Function description

Timeout callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_RegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID, pXSPI_CallbackTypeDef pCallback)

Function description

Register a User XSPI Callback To be used to override the weak predefined callback.

Parameters

- **hxspi:** : XSPI handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_XSPI_ERROR_CB_ID XSPI Error Callback ID
  - HAL_XSPI_ABORT_CB_ID XSPI Abort Callback ID
  - HAL_XSPI_FIFO_THRESHOLD_CB_ID XSPI FIFO Threshold Callback ID
  - HAL_XSPI_CMD_CPLT_CB_ID XSPI Command Complete Callback ID
  - HAL_XSPI_RX_CPLT_CB_ID XSPI Rx Complete Callback ID
  - HAL_XSPI_TX_CPLT_CB_ID XSPI Tx Complete Callback ID
  - HAL_XSPI_RX_HALF_CPLT_CB_ID XSPI Rx Half Complete Callback ID
  - HAL_XSPI_TX_HALF_CPLT_CB_ID XSPI Tx Half Complete Callback ID
  - HAL_XSPI_STATUS_MATCH_CB_ID XSPI Status Match Callback ID
  - HAL_XSPI_TIMEOUT_CB_ID XSPI Timeout Callback ID
  - HAL_XSPI_MSP_INIT_CB_ID XSPI MspInit callback ID
  - HAL_XSPI_MSP_DEINIT_CB_ID XSPI MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_XSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_UnRegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID)

Function description

Unregister a User XSPI Callback XSPI Callback is redirected to the weak predefined callback.

Parameters

- **hxspi:** : XSPI handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_XSPI_ERROR_CB_ID XSPI Error Callback ID
  - HAL_XSPI_ABORT_CB_ID XSPI Abort Callback ID
  - HAL_XSPI_FIFO_THRESHOLD_CB_ID XSPI FIFO Threshold Callback ID
  - HAL_XSPI_CMD_CPLT_CB_ID XSPI Command Complete Callback ID
  - HAL_XSPI_RX_CPLT_CB_ID XSPI Rx Complete Callback ID
  - HAL_XSPI_TX_CPLT_CB_ID XSPI Tx Complete Callback ID
  - HAL_XSPI_RX_HALF_CPLT_CB_ID XSPI Rx Half Complete Callback ID
  - HAL_XSPI_TX_HALF_CPLT_CB_ID XSPI Tx Half Complete Callback ID
  - HAL_XSPI_STATUS_MATCH_CB_ID XSPI Status Match Callback ID
  - HAL_XSPI_TIMEOUT_CB_ID XSPI Timeout Callback ID
  - HAL_XSPI_MSP_INIT_CB_ID XSPI MspInit callback ID
  - HAL_XSPI_MSP_DEINIT_CB_ID XSPI MspDeInit callback ID

Return values

- **status:**

HAL_XSPI_Abort

Function name

HAL_StatusTypeDef HAL_XSPI_Abort (XSPI_HandleTypeDef * hxspi)

Function description

Abort the current transmission.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Abort_IT (XSPI_HandleTypeDef * hxspi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_XSPI_SetFifoThreshold (XSPI_HandleTypeDef * hxspi, uint32_t Threshold)

Function description

Set XSPI Fifo threshold.

Parameters

- **hxspi:** : XSPI handle.
- **Threshold:** : Threshold of the Fifo.

Return values

- **HAL:** status

HAL_XSPI_GetFifoThreshold

Function name

uint32_t HAL_XSPI_GetFifoThreshold (const XSPI_HandleTypeDef * hxspi)

Function description

Get XSPI Fifo threshold.

Parameters

- **hxspi:** : XSPI handle.

Return values

- **Fifo:** threshold

HAL_XSPI_SetMemoryType

Function name

HAL_StatusTypeDef HAL_XSPI_SetMemoryType (XSPI_HandleTypeDef * hxspi, uint32_t Type)

Function description

Set XSPI Memory Type.

Parameters

- **hxspi:** : XSPI handle.
- **Type:** : Memory Type.

Return values

- **HAL:** status

HAL_XSPI_SetDeviceSize

Function name

HAL_StatusTypeDef HAL_XSPI_SetDeviceSize (XSPI_HandleTypeDef * hxspi, uint32_t Size)

Function description

Set XSPI Device Size.

Parameters

- **hxspi:** : XSPI handle.
- **Size:** : Device Size.

Return values

- **HAL:** status

HAL_XSPI_SetClockPrescaler

Function name

HAL_StatusTypeDef HAL_XSPI_SetClockPrescaler (XSPI_HandleTypeDef * hxspi, uint32_t Prescaler)

Function description

Set XSPI Clock prescaler.

Parameters

- **hxspi:** : XSPI handle.
- **Prescaler:** : Clock prescaler.

Return values

- **HAL:** status

HAL_XSPI_SetTimeout

Function name

HAL_StatusTypeDef HAL_XSPI_SetTimeout (XSPI_HandleTypeDef * hxspi, uint32_t Timeout)

Function description

Set XSPI timeout.

Parameters

- **hxspi:** : XSPI handle.
- **Timeout:** : Timeout for the memory access.

Return values

- **HAL:** state

HAL_XSPI_GetError

Function name

uint32_t HAL_XSPI_GetError (const XSPI_HandleTypeDef * hxspi)

Function description

Return the XSPI error code.

Parameters

- **hxspi:** : XSPI handle

Return values

- **XSPI:** Error Code

HAL_XSPI_GetState

Function name

uint32_t HAL_XSPI_GetState (const XSPI_HandleTypeDef * hxspi)

Function description

Return the XSPI handle state.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** state

HAL_XSPI_DLYB_SetConfig

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_SetConfig (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Set the Delay Block configuration.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

HAL_XSPI_DLYB_GetConfig

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_GetConfig (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Get the Delay Block configuration.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

HAL_XSPI_DLYB_GetClockPeriod

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_GetClockPeriod (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Get the Delay line length value.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

84.3 XSPI Firmware driver defines

The following section lists the various define and macros of the module.

84.3.1 XSPI

XSPI

XSPI Address DTR Mode

HAL_XSPI_ADDRESS_DTR_DISABLE

DTR mode disabled for address phase

HAL_XSPI_ADDRESS_DTR_ENABLE

DTR mode enabled for address phase

XSPI Address Mode

HAL_XSPI_ADDRESS_NONE

No address

HAL_XSPI_ADDRESS_1_LINE

Address on a single line

HAL_XSPI_ADDRESS_2_LINES

Address on two lines

HAL_XSPI_ADDRESS_4_LINES

Address on four lines

HAL_XSPI_ADDRESS_8_LINES

Address on eight lines

XSPI Hyperbus Address Space

HAL_XSPI_MEMORY_ADDRESS_SPACE

HyperBus memory mode

HAL_XSPI_REGISTER_ADDRESS_SPACE

HyperBus register mode

XSPI Address width

HAL_XSPI_ADDRESS_8_BITS

8-bit address

HAL_XSPI_ADDRESS_16_BITS

16-bit address

HAL_XSPI_ADDRESS_24_BITS

24-bit address

HAL_XSPI_ADDRESS_32_BITS

32-bit address

XSPI Alternate Bytes DTR Mode

HAL_XSPI_ALT_BYTES_DTR_DISABLE

DTR mode disabled for alternate bytes phase

HAL_XSPI_ALT_BYTES_DTR_ENABLE

DTR mode enabled for alternate bytes phase

XSPI Alternate Bytes Mode

HAL_XSPI_ALT_BYTES_NONE

No alternate bytes

HAL_XSPI_ALT_BYTES_1_LINE

Alternate bytes on a single line

HAL_XSPI_ALT_BYTES_2_LINES

Alternate bytes on two lines

HAL_XSPI_ALT_BYTES_4_LINES

Alternate bytes on four lines

HAL_XSPI_ALT_BYTES_8_LINES

Alternate bytes on eight lines

XSPI Alternate Bytes Width

HAL_XSPI_ALT_BYTES_8_BITS

8-bit alternate bytes

HAL_XSPI_ALT_BYTES_16_BITS

16-bit alternate bytes

HAL_XSPI_ALT_BYTES_24_BITS

24-bit alternate bytes

HAL_XSPI_ALT_BYTES_32_BITS

32-bit alternate bytes

XSPI Automatic Stop

HAL_XSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or XSPI disabling

HAL_XSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

XSPI Chip Select Boundary

HAL_XSPI_BONDARYOF_NONE

HAL_XSPI_BONDARYOF_16B

16 bits (2 Byte = 2^(1))

HAL_XSPI_BONDARYOF_32B

32 bits (4 Byte = 2^(2))

HAL_XSPI_BONDARYOF_64B

64 bits (8 Byte = 2^(3))

HAL_XSPI_BONDARYOF_128B

128 bits (16 Byte = 2^(4))

HAL_XSPI_BONDARYOF_256B

256 bits (32 Byte = 2^(5))

HAL_XSPI_BONDARYOF_512B

512 bits (64 Byte = 2^(6))

HAL_XSPI_BONDARYOF_1KB

1 Kbits (128 Byte = 2^(7))

HAL_XSPI_BONDARYOF_2KB

2 Kbits (256 Byte = 2^(8))

HAL_XSPI_BONDARYOF_4KB

4 Kbits (512 Byte = 2^(9))

HAL_XSPI_BONDARYOF_8KB

8 Kbits (1 KByte = 2^(10))

HAL_XSPI_BONDARYOF_16KB

16 Kbits (2 KByte = 2^(11))

HAL_XSPI_BONDARYOF_32KB

32 Kbits (4 KByte = 2^(12))

HAL_XSPI_BONDARYOF_64KB

64 Kbits (8 KByte = 2^(13))

HAL_XSPI_BONDARYOF_128KB

128 Kbits (16 KByte = 2^(14))

HAL_XSPI_BONDARYOF_256KB

256 Kbits (32 KByte = 2^(15))

HAL_XSPI_BONDARYOF_512KB

512 Kbits (64 KByte = 2^(16))

HAL_XSPI_BONDARYOF_1MB

1 Mbits (128 KByte = 2^(17))

HAL_XSPI_BONDARYOF_2MB

2 Mbits (256 KByte = 2^(18))

HAL_XSPI_BONDARYOF_4MB

4 Mbits (512 KByte = 2^(19))

HAL_XSPI_BONDARYOF_8MB

8 Mbits (1 MByte = 2^(20))

HAL_XSPI_BONDARYOF_16MB

16 Mbits (2 MByte = 2^(21))

HAL_XSPI_BONDARYOF_32MB

32 Mbits (4 MByte = 2^(22))

HAL_XSPI_BONDARYOF_64MB

64 Mbits (8 MByte = 2^(23))

HAL_XSPI_BONDARYOF_128MB

128 Mbits (16 MByte = 2^(24))

HAL_XSPI_BONDARYOF_256MB

256 Mbits (32 MByte = 2^(25))

HAL_XSPI_BONDARYOF_512MB

512 Mbits (64 MByte = 2^(26))

HAL_XSPI_BONDARYOF_1GB

1 Gbits (128 MByte = 2^(27))

HAL_XSPI_BONDARYOF_2GB

2 Gbits (256 MByte = 2^(28))

HAL_XSPI_BONDARYOF_4GB

4 Gbits (512 MByte = 2^(29))

HAL_XSPI_BONDARYOF_8GB

8 Gbits (1 GByte = 2^(30))

HAL_XSPI_BONDARYOF_16GB

16 Gbits (2 GByte = 2^(31))

XSPI Clock Mode

HAL_XSPI_CLOCK_MODE_0

CLK must stay low while nCS is high

HAL_XSPI_CLOCK_MODE_3

CLK must stay high while nCS is high

XSPI Data DTR Mode

HAL_XSPI_DATA_DTR_DISABLE

DTR mode disabled for data phase

HAL_XSPI_DATA_DTR_ENABLE

DTR mode enabled for data phase

XSPI Data Mode

HAL_XSPI_DATA_NONE

No data

HAL_XSPI_DATA_1_LINE

Data on a single line

HAL_XSPI_DATA_2_LINES

Data on two lines

HAL_XSPI_DATA_4_LINES

Data on four lines

HAL_XSPI_DATA_8_LINES

Data on eight lines

XSPI Delay Block Bypaas

HAL_XSPI_DELAY_BLOCK_ON

Sampling clock is delayed by the delay block

HAL_XSPI_DELAY_BLOCK_BYPASS

Delay block is bypassed

XSPI Delay Hold Quarter Cycle

HAL_XSPI_DHQC_DISABLE

No Delay

HAL_XSPI_DHQC_ENABLE

Delay Hold 1/4 cycle

XSPI DQS Mode

HAL_XSPI_DQS_DISABLE

DQS disabled

HAL_XSPI_DQS_ENABLE

DQS enabled

XSPI Error Code

HAL_XSPI_ERROR_NONE

No error

HAL_XSPI_ERROR_TIMEOUT

Timeout error

HAL_XSPI_ERROR_TRANSFER

Transfer error

HAL_XSPI_ERROR_DMA

DMA transfer error

HAL_XSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_XSPI_ERROR_INVALID_SEQUENCE

Sequence is incorrect

HAL_XSPI_ERROR_INVALID_CALLBACK

Invalid callback error

XSPI Exported Macros

HAL_XSPI_RESET_HANDLE_STATE

Description:

- Reset XSPI handle state.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE

Description:

- Enable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_DISABLE

Description:

- Disable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE_IT

Description:

- Enable the specified XSPI interrupt.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to enable. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- None

HAL_XSPI_DISABLE_IT

Description:

- Disable the specified XSPI interrupt.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to disable. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- None

HAL_XSPI_GET_IT_SOURCE

Description:

- Check whether the specified XSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to check. This parameter can be one of the following values:
  - HAL_XSPI_IT_TO: XSPI Timeout interrupt
  - HAL_XSPI_IT_SM: XSPI Status match interrupt
  - HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  - HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  - HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

HAL_XSPI_GET_FLAG

Description:

- Check whether the selected XSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI flag to check. This parameter can be one of the following values:
  - HAL_XSPI_FLAG_BUSY: XSPI Busy flag
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_FT: XSPI FIFO threshold flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

HAL_XSPI_CLEAR_FLAG

Description:

- Clears the specified XSPI's flag status.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI clear register flag that needs to be set This parameter can be one of the following values:
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

XSPI Exported Types

HAL_XSPI_DLYB_CfgTypeDef

XSPI Flags

HAL_XSPI_FLAG_BUSY

Busy flag: operation is ongoing

HAL_XSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

HAL_XSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

HAL_XSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

HAL_XSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

HAL_XSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

XSPI Free Running Clock

HAL_XSPI_FREERUNCLK_DISABLE

CLK is not free running

HAL_XSPI_FREERUNCLK_ENABLE

CLK is always provided (running)

XSPI Instruction DTR Mode

HAL_XSPI_INSTRUCTION_DTR_DISABLE

DTR mode disabled for instruction phase

HAL_XSPI_INSTRUCTION_DTR_ENABLE

DTR mode enabled for instruction phase

XSPI Instruction Mode

HAL_XSPI_INSTRUCTION_NONE

No instruction

HAL_XSPI_INSTRUCTION_1_LINE

Instruction on a single line

HAL_XSPI_INSTRUCTION_2_LINES

Instruction on two lines

HAL_XSPI_INSTRUCTION_4_LINES

Instruction on four lines

HAL_XSPI_INSTRUCTION_8_LINES

Instruction on eight lines

XSPI Instruction Width

HAL_XSPI_INSTRUCTION_8_BITS

8-bit instruction

HAL_XSPI_INSTRUCTION_16_BITS

16-bit instruction

HAL_XSPI_INSTRUCTION_24_BITS

24-bit instruction

HAL_XSPI_INSTRUCTION_32_BITS

32-bit instruction

XSPI Interrupts

HAL_XSPI_IT_TO

Interrupt on the timeout flag

HAL_XSPI_IT_SM

Interrupt on the status match flag

HAL_XSPI_IT_FT

Interrupt on the fifo threshold flag

HAL_XSPI_IT_TC

Interrupt on the transfer complete flag

HAL_XSPI_IT_TE

Interrupt on the transfer error flag

XSPI IO Select

HAL_XSPI_SELECT_IO_3_0

Data exchanged over IO[3:0]

HAL_XSPI_SELECT_IO_7_4

Data exchanged over IO[7:4]

HAL_XSPI_SELECT_IO_7_0

Data exchanged over IO[7:0]

XSPI Hyperbus Latency Mode

HAL_XSPI_VARIABLE_LATENCY

Variable initial latency

HAL_XSPI_FIXED_LATENCY

Fixed latency

XSPI Match Mode

HAL_XSPI_MATCH_MODE_AND

AND match mode between unmasked bits

HAL_XSPI_MATCH_MODE_OR

OR match mode between unmasked bits

XSPI Memory Mode

HAL_XSPI_SINGLE_MEM

Dual-memory mode disabled

HAL_XSPI_DUAL_MEM

Dual mode enabled

XSPI Memory Size

HAL_XSPI_SIZE_16B

16 bits (2 Byte = 2^(0+1))

HAL_XSPI_SIZE_32B

32 bits (4 Byte = 2^(1+1))

HAL_XSPI_SIZE_64B

64 bits (8 Byte = 2^(2+1))

HAL_XSPI_SIZE_128B

128 bits (16 Byte = 2^(3+1))

HAL_XSPI_SIZE_256B

256 bits (32 Byte = 2^(4+1))

HAL_XSPI_SIZE_512B

512 bits (64 Byte = 2^(5+1))

HAL_XSPI_SIZE_1KB

1 Kbits (128 Byte = 2^(6+1))

HAL_XSPI_SIZE_2KB

2 Kbits (256 Byte = 2^(7+1))

HAL_XSPI_SIZE_4KB

4 Kbits (512 Byte = 2^(8+1))

HAL_XSPI_SIZE_8KB

8 Kbits (1 KByte = 2^(9+1))

HAL_XSPI_SIZE_16KB

16 Kbits (2 KByte = 2^(10+1))

HAL_XSPI_SIZE_32KB

32 Kbits (4 KByte = 2^(11+1))

HAL_XSPI_SIZE_64KB

64 Kbits (8 KByte = 2^(12+1))

HAL_XSPI_SIZE_128KB

128 Kbits (16 KByte = 2^(13+1))

HAL_XSPI_SIZE_256KB

256 Kbits (32 KByte = 2^(14+1))

HAL_XSPI_SIZE_512KB

512 Kbits (64 KByte = 2^(15+1))

HAL_XSPI_SIZE_1MB

1 Mbits (128 KByte = 2^(16+1))

HAL_XSPI_SIZE_2MB

2 Mbits (256 KByte = 2^(17+1))

HAL_XSPI_SIZE_4MB

4 Mbits (512 KByte = 2^(18+1))

HAL_XSPI_SIZE_8MB

8 Mbits (1 MByte = 2^(19+1))

HAL_XSPI_SIZE_16MB

16 Mbits (2 MByte = 2^(20+1))

HAL_XSPI_SIZE_32MB

32 Mbits (4 MByte = 2^(21+1))

HAL_XSPI_SIZE_64MB

64 Mbits (8 MByte = 2^(22+1))

HAL_XSPI_SIZE_128MB

128 Mbits (16 MByte = 2^(23+1))

HAL_XSPI_SIZE_256MB

256 Mbits (32 MByte = 2^(24+1))

HAL_XSPI_SIZE_512MB

512 Mbits (64 MByte = 2^(25+1))

HAL_XSPI_SIZE_1GB

1 Gbits (128 MByte = 2^(26+1))

HAL_XSPI_SIZE_2GB

2 Gbits (256 MByte = 2^(27+1))

HAL_XSPI_SIZE_4GB

4 Gbits (256 MByte = 2^(28+1))

HAL_XSPI_SIZE_8GB

8 Gbits (256 MByte = 2^(29+1))

HAL_XSPI_SIZE_16GB

16 Gbits (256 MByte = 2^(30+1))

HAL_XSPI_SIZE_32GB

32 Gbits (256 MByte = 2^(31+1))

XSPI Memory Type

HAL_XSPI_MEMTYPE_MICRON

Micron mode

HAL_XSPI_MEMTYPE_MACRONIX

Macronix mode

HAL_XSPI_MEMTYPE_APMEM

AP Memory mode

HAL_XSPI_MEMTYPE_MACRONIX_RAM

Macronix RAM mode

HAL_XSPI_MEMTYPE_HYPERBUS

Hyperbus mode

HAL_XSPI_MEMTYPE_APMEM_16BITS

AP Memory mode

XSPI Operation Type

HAL_XSPI_OPTYPE_COMMON_CFG

Common configuration (indirect or auto-polling mode)

HAL_XSPI_OPTYPE_READ_CFG

Read configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRITE_CFG

Write configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRAP_CFG

Wrap configuration (memory-mapped mode)

XSPI Sample Shifting

HAL_XSPI_SAMPLE_SHIFT_NONE

No shift

HAL_XSPI_SAMPLE_SHIFT_HALFCYCLE

1/2 cycle shift

XSPI SIOO Mode

HAL_XSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

HAL_XSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

XSPI State

HAL_XSPI_STATE_RESET

Initial state

HAL_XSPI_STATE_READY

Driver ready to be used

HAL_XSPI_STATE_HYPERBUS_INIT

Initialization done in hyperbus mode but timing configuration not done

HAL_XSPI_STATE_CMD_CFG

Command (regular or hyperbus) configured, ready for an action

HAL_XSPI_STATE_READ_CMD_CFG

Read command configuration done, not the write command configuration

HAL_XSPI_STATE_WRITE_CMD_CFG

Write command configuration done, not the read command configuration

HAL_XSPI_STATE_BUSY_CMD

Command without data on-going

HAL_XSPI_STATE_BUSY_TX

Indirect Tx on-going

HAL_XSPI_STATE_BUSY_RX

Indirect Rx on-going

HAL_XSPI_STATE_BUSY_AUTO_POLLING

Auto-polling on-going

HAL_XSPI_STATE_BUSY_MEM_MAPPED

Memory-mapped on-going

HAL_XSPI_STATE_ABORT

Abort on-going

HAL_XSPI_STATE_ERROR

Blocking error, driver should be re-initialized

XSPI Timeout Activation

HAL_XSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

HAL_XSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

XSPI Timeout definition

HAL_XSPI_TIMEOUT_DEFAULT_VALUE

XSPI Wrap-Size

HAL_XSPI_WRAP_NOT_SUPPORTED

wrapped reads are not supported by the memory

HAL_XSPI_WRAP_16_BYTES

external memory supports wrap size of 16 bytes

HAL_XSPI_WRAP_32_BYTES

external memory supports wrap size of 32 bytes

HAL_XSPI_WRAP_64_BYTES

external memory supports wrap size of 64 bytes

HAL_XSPI_WRAP_128_BYTES

external memory supports wrap size of 128 bytes

XSPI Hyperbus Write Zero Latency Activation

HAL_XSPI_LATENCY_ON_WRITE

Latency on write accesses

HAL_XSPI_NO_LATENCY_ON_WRITE

No latency on write accesses

85 LL ADC Generic Driver

85.1 ADC Firmware driver registers structures

85.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32h5xx_ll_adc.h

Data Fields

- ***uint32_t CommonClock***
- ***uint32_t Multimode***
- ***uint32_t MultiDMATransfer***
- ***uint32_t MultiTwoSamplingDelay***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***
  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE***
  Note:
  – On this STM32 series, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual. This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***
  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiDMATransfer***
  Set ADC multimode conversion data transfer: no transfer or transfer by DMA. This parameter can be a value of ***ADC_LL_EC_MULTI_DMA_TRANSFER*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultiDMATransfer()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiTwoSamplingDelay***
  Set ADC multimode delay between 2 sampling phases. This parameter can be a value of ***ADC_LL_EC_MULTI_TWOSMP_DELAY*** This feature can be modified afterwards using unitary function **LL_ADC_SetMultiTwoSamplingDelay()**.

85.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32h5xx_ll_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t DataAlignment***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***
  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION*** This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::DataAlignment***
  Set ADC conversion data alignment. This parameter can be a value of ***ADC_LL_EC_DATA_ALIGN*** This feature can be modified afterwards using unitary function **LL_ADC_SetDataAlignment()**.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***
  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_SetLowPowerMode()**.

85.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32h5xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DMATransfer***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***
  Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge(default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***
  Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***
  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***
  Note:
  – This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***
  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DMATransfer***
  Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of ***ADC_LL_EC_REG_DMA_TRANSFER*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDMATransfer()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***
  Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

85.1.4 LL_ADC_INJ_InitTypeDef

LL_ADC_INJ_InitTypeDef is defined in the stm32h5xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***
  Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***
  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH*** This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***
  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***
  Note:
  – This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***
  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

85.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

85.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (const ADC_TypeDef * ADCx, uint32_t Register)

Function description

Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **ADCx:** ADC instance
- **Register:** This parameter can be one of the following values:
  – LL_ADC_DMA_REG_REGULAR_DATA
  – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)
  (1) Available on devices with several ADC instances.

Return values

- **ADC:** register address

Notes

- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_DMA_GetRegAddr
- CDR RDATA_MST LL_ADC_DMA_GetRegAddr
- CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name

__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)

Function description

Set parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **CommonClock:** This parameter can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Return values

- **None:**

Notes

- On this STM32 series, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_SetCommonClock
- CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_GetCommonClock
- CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) The values not selected are removed from configuration.
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literals LL_ADC_DELAY_TEMPSENSOR_STAB_US, LL_ADC_DELAY_TEMPSENSOR_BUFFER_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalCh
- CCR TSEN LL_ADC_SetCommonPathInternalCh
- CCR VBATEN LL_ADC_SetCommonPathInternalCh

LL_ADC_SetCommonPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChAdd (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literals LL_ADC_DELAY_TEMPSENSOR_STAB_US, LL_ADC_DELAY_TEMPSENSOR_BUFFER_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChAdd
- CCR TSEN LL_ADC_SetCommonPathInternalChAdd
- CCR VBATEN LL_ADC_SetCommonPathInternalChAdd

LL_ADC_SetCommonPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChRem (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChRem
- CCR TSEN LL_ADC_SetCommonPathInternalChRem
- CCR VBATEN LL_ADC_SetCommonPathInternalChRem

LL_ADC_GetCommonPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_GetCommonPathInternalCh
- CCR TSEN LL_ADC_GetCommonPathInternalCh
- CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_EnableChannelVDDcore

Function name

__STATIC_INLINE void LL_ADC_EnableChannelVDDcore (ADC_TypeDef * ADCx)

Function description

Enable VddCore (internal digital core voltage) channel.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, VddCore channel is controlled via a specific register.
- On this STM32 series, VddCore channel is on ADC2 instance only.

Reference Manual to LL API cross reference:

- OR OP0 LL_ADC_EnableChannelVDDcore

LL_ADC_DisableChannelVDDcore

Function name

__STATIC_INLINE void LL_ADC_DisableChannelVDDcore (ADC_TypeDef * ADCx)

Function description

Disable VddCore (internal digital core voltage) channel.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, VddCore channel is controlled via a specific register.
- On this STM32 series, VddCore channel is on ADC2 instance only.

Reference Manual to LL API cross reference:

- OR OP0 LL_ADC_DisableChannelVDDcore

LL_ADC_EnableChannel0_GPIO

Function name

__STATIC_INLINE void LL_ADC_EnableChannel0_GPIO (const ADC_TypeDef * ADCx)

Function description

Enable Channel 0 GPIO switch control.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, Channel 0 channel connection to GPIO is controlled via specific register.
- On this STM32 series, Channel 0 GPIO switch control must be enabled when INP0 is used.

Reference Manual to LL API cross reference:

- OR OP0 LL_ADC_EnableChannel0_GPIO

LL_ADC_DisableChannel0_GPIO

Function name

__STATIC_INLINE void LL_ADC_DisableChannel0_GPIO (const ADC_TypeDef * ADCx)

Function description

Disable Channel 0 GPIO switch control.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, Channel 0 connection to GPIO is controlled via specific register.

Reference Manual to LL API cross reference:

- OR OP0 LL_ADC_DisableChannel0_GPIO

LL_ADC_SetCalibrationFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED
  - LL_ADC_BOTH_SINGLE_DIFF_ENDED
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration factor must be specified for each of these differential modes, if used afterwards and if the application requires their calibration).
- In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_SetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (const ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7F

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_GetCalibrationFactor
- CALFACT CALFACT_D LL_ADC_GetCalibrationFactor

LL_ADC_SetResolution

Function name

__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)

Function description

Set ADC resolution.

Parameters

- **ADCx:** ADC instance
- **Resolution:** This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name

__STATIC_INLINE uint32_t LL_ADC_GetResolution (const ADC_TypeDef * ADCx)

Function description

Get ADC resolution.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_GetResolution

LL_ADC_SetDataAlignment

Function name

__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)

Function description

Set ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance
- **DataAlignment:** This parameter can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Return values

- **None:**

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (const ADC_TypeDef * ADCx)

Function description

Get ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.

Reference Manual to LL API cross reference:

- CFGR ALIGN LL_ADC_GetDataAlignment

LL_ADC_SetLowPowerMode

Function name

__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Return values

- **None:**

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start.
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (const ADC_TypeDef * ADCx)

Function description

Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start.
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_GetLowPowerMode

LL_ADC_SetOffset

Function name

__STATIC_INLINE void LL_ADC_SetOffset (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel, uint32_t OffsetLevel)

Function description

Set ADC selected offset instance 1, 2, 3 or 4.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- This function set the 2 items of offset configuration: ADC channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)Offset level (offset to be subtracted from the raw converted data).
- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
- This function enables the offset, by default. It can be forced to disable state using function LL_ADC_SetOffsetState().
- If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On STM32H5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_SetOffset
- OFR1 OFFSET1 LL_ADC_SetOffset
- OFR1 OFFSET1_EN LL_ADC_SetOffset
- OFR2 OFFSET2_CH LL_ADC_SetOffset
- OFR2 OFFSET2 LL_ADC_SetOffset
- OFR2 OFFSET2_EN LL_ADC_SetOffset
- OFR3 OFFSET3_CH LL_ADC_SetOffset
- OFR3 OFFSET3 LL_ADC_SetOffset
- OFR3 OFFSET3_EN LL_ADC_SetOffset
- OFR4 OFFSET4_CH LL_ADC_SetOffset
- OFR4 OFFSET4 LL_ADC_SetOffset
- OFR4 OFFSET4_EN LL_ADC_SetOffset

LL_ADC_GetOffsetChannel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(4)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)(4)
  - LL_ADC_CHANNEL_VBAT (2)(4)
  - LL_ADC_CHANNEL_VDDCORE (2)(4)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC) (4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().
- On STM32H5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_GetOffsetChannel
- OFR2 OFFSET2_CH LL_ADC_GetOffsetChannel
- OFR3 OFFSET3_CH LL_ADC_GetOffsetChannel
- OFR4 OFFSET4_CH LL_ADC_GetOffsetChannel

LL_ADC_GetOffsetLevel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_GetOffsetLevel
- OFR2 OFFSET2 LL_ADC_GetOffsetLevel
- OFR3 OFFSET3 LL_ADC_GetOffsetLevel
- OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetOffsetState

Function name

__STATIC_INLINE void LL_ADC_SetOffsetState (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetState)

Function description

Set for the ADC selected offset instance 1, 2, 3 or 4: force offset state disable or enable without modifying offset channel or offset value.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetState:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_DISABLE
  - LL_ADC_OFFSET_ENABLE

Return values

- **None:**

Notes

- This function should be needed only in case of offset to be enabled-disabled dynamically, and should not be needed in other cases: function LL_ADC_SetOffset() automatically enables the offset.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_EN LL_ADC_SetOffsetState
- OFR2 OFFSET2_EN LL_ADC_SetOffsetState
- OFR3 OFFSET3_EN LL_ADC_SetOffsetState
- OFR4 OFFSET4_EN LL_ADC_SetOffsetState

LL_ADC_GetOffsetState

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetState (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: offset state disabled or enabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_DISABLE
  - LL_ADC_OFFSET_ENABLE

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_EN LL_ADC_GetOffsetState
- OFR2 OFFSET2_EN LL_ADC_GetOffsetState
- OFR3 OFFSET3_EN LL_ADC_GetOffsetState
- OFR4 OFFSET4_EN LL_ADC_GetOffsetState

LL_ADC_SetOffsetSign

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSign (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSign)

Function description

Set for the ADC selected offset instance 1, 2, 3 or 4: choose offset sign.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSign:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_SetOffsetSign
- OFR2 OFFSETPOS LL_ADC_SetOffsetSign
- OFR3 OFFSETPOS LL_ADC_SetOffsetSign
- OFR4 OFFSETPOS LL_ADC_SetOffsetSign

LL_ADC_GetOffsetSign

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSign (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: offset sign if positive or negative.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_GetOffsetSign
- OFR2 OFFSETPOS LL_ADC_GetOffsetSign
- OFR3 OFFSETPOS LL_ADC_GetOffsetSign
- OFR4 OFFSETPOS LL_ADC_GetOffsetSign

LL_ADC_SetOffsetSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSaturation)

Function description

Set for the ADC selected offset instance 1, 2, 3 or 4: choose offset saturation mode.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SATURATION_ENABLE
  - LL_ADC_OFFSET_SATURATION_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 SATEN LL_ADC_SetOffsetSaturation
- OFR2 SATEN LL_ADC_SetOffsetSaturation
- OFR3 SATEN LL_ADC_SetOffsetSaturation
- OFR4 SATEN LL_ADC_SetOffsetSaturation

LL_ADC_GetOffsetSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSaturation (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset instance 1, 2, 3 or 4: offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SATURATION_ENABLE
  - LL_ADC_OFFSET_SATURATION_DISABLE

Reference Manual to LL API cross reference:

- OFR1 SATEN LL_ADC_GetOffsetSaturation
- OFR2 SATEN LL_ADC_GetOffsetSaturation
- OFR3 SATEN LL_ADC_GetOffsetSaturation
- OFR4 SATEN LL_ADC_GetOffsetSaturation

LL_ADC_SetSamplingTimeCommonConfig

Function name

__STATIC_INLINE void LL_ADC_SetSamplingTimeCommonConfig (ADC_TypeDef * ADCx, uint32_t SamplingTimeCommonConfig)

Function description

Set ADC sampling time common configuration impacting settings of sampling time channel wise.

Parameters

- **ADCx:** ADC instance
- **SamplingTimeCommonConfig:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_DEFAULT
  - LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMPPLUS LL_ADC_SetSamplingTimeCommonConfig

LL_ADC_GetSamplingTimeCommonConfig

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonConfig (const ADC_TypeDef * ADCx)

Function description

Get ADC sampling time common configuration impacting settings of sampling time channel wise.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_DEFAULT
  - LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

Reference Manual to LL API cross reference:

- SMPR1 SMPPLUS LL_ADC_GetSamplingTimeCommonConfig

LL_ADC_REG_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO (2)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE15

  (1) On STM32H5 series, parameter specific to devices: STM32H563/H573xx. (2) On STM32H5 series, parameter specific to devices: STM32H503xx.

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4 (1)
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM7_TRGO (2)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2 (1)
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE15

  (1) On STM32H5 series, parameter specific to devices: STM32H563/H573xx. (2) On STM32H5 series, parameter specific to devices: STM32H503xx.

Notes

- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Notes

- Applicable only for trigger source set to external trigger.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSamplingMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetSamplingMode (ADC_TypeDef * ADCx, uint32_t SamplingMode)

Function description

Set ADC sampling mode.

Parameters

- **ADCx:** ADC instance
- **SamplingMode:** This parameter can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Return values

- **None:**

Notes

- This function set the ADC conversion sampling mode
- This mode applies to regular group only.
- Set sampling mode is applied to all conversion of regular group.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_SetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_SetSamplingMode

LL_ADC_REG_GetSamplingMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSamplingMode (const ADC_TypeDef * ADCx)

Function description

Get the ADC sampling mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_GetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_GetSamplingMode

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_SetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_GetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(4)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)(4)
  - LL_ADC_CHANNEL_VBAT (2)(4)
  - LL_ADC_CHANNEL_VDDCORE (2)(4)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC) (4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetContinuousMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)

Function description

Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (const ADC_TypeDef * ADCx)

Function description

Get ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDMATransfer

Function name

__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)

Function description

Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance
- **DMATransfer:** This parameter can be one of the following values:
  - LL_ADC_REG_DMA_TRANSFER_NONE
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Return values

- **None:**

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_SetMultiDMATransfer().
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR DMAEN LL_ADC_REG_SetDMATransfer
- CFGR DMACFG LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DMA_TRANSFER_NONE
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- For devices with several ADC instances: ADC multimode DMA settings are available using function LL_ADC_GetMultiDMATransfer().
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CFGR DMAEN LL_ADC_REG_GetDMATransfer
- CFGR DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetOverrun

Function name

__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Set ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Return values

- **None:**

Notes

- Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

  (1) On STM32H5 series, parameter specific to devices: STM32H563/H573xx. (2) On STM32H5 series, parameter specific to devices: STM32H503xx.

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 series having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

  (1) On STM32H5 series, parameter specific to devices: STM32H563/H573xx. (2) On STM32H5 series, parameter specific to devices: STM32H503xx.

Notes

- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values

- **None:**

Notes

- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes

- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values

- **None:**

Notes

- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32H5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(4)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)(4)
  - LL_ADC_CHANNEL_VBAT (2)(4)
  - LL_ADC_CHANNEL_VDDCORE (2)(4)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC) (4) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_SetQueueMode

Function name

__STATIC_INLINE void LL_ADC_INJ_SetQueueMode (ADC_TypeDef * ADCx, uint32_t QueueMode)

Function description

Set ADC group injected contexts queue mode.

Parameters

- **ADCx:** ADC instance
- **QueueMode:** This parameter can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks If contexts queue is disabled:only 1 sequence can be configured and is active perpetually. If contexts queue is enabled:up to 2 contexts can be queued and are checked in and out as a FIFO stack (first-in, first-out).If a new context is set when queues is full, error is triggered by interruption "Injected Queue Overflow".Two behaviors are possible when all contexts have been processed: the contexts queue can maintain the last context active perpetually or can be empty and injected group triggers are disabled.Triggers can be only external (not internal SW start)Caution: The sequence must be fully configured in one time (one write of register JSQR makes a check-in of a new context into the queue). Therefore functions to set separately injected trigger and sequencer channels cannot be used, register JSQR must be set using function LL_ADC_INJ_ConfigQueueContext().
- This parameter can be modified only when no conversion is on going on either groups regular or injected.
- A modification of the context mode (bit JQDIS) causes the contexts queue to be flushed and the register JSQR is cleared.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_SetQueueMode
- CFGR JQDIS LL_ADC_INJ_SetQueueMode

LL_ADC_INJ_GetQueueMode

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetQueueMode (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected context queue mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_GetQueueMode
- CFGR JQDIS LL_ADC_INJ_GetQueueMode

LL_ADC_INJ_ConfigQueueContext

Function name

__STATIC_INLINE void LL_ADC_INJ_ConfigQueueContext (ADC_TypeDef * ADCx, uint32_t TriggerSource, uint32_t ExternalTriggerEdge, uint32_t SequencerNbRanks, uint32_t Rank1_Channel, uint32_t Rank2_Channel, uint32_t Rank3_Channel, uint32_t Rank4_Channel)

Function description

Set one context on ADC group injected that will be checked in contexts queue.

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM7_TRGO (2)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4 (1)
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO (1)
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

  (1) On STM32H5 series, parameter specific to devices: STM32H563/H573xx. (2) On STM32H5 series, parameter specific to devices: STM32H503xx.
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

  Note: This parameter is discarded in case of SW start: parameter "TriggerSource" set to "LL_ADC_INJ_TRIG_SOFTWARE".
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

- **Rank1_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

- **Rank2_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

- **Rank3_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

- **Rank4_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks This function is intended to be used when contexts queue is enabled, because the sequence must be fully configured in one time (functions to set separately injected trigger and sequencer channels cannot be used): Refer to function LL_ADC_INJ_SetQueueMode().
- In the contexts queue, only the active context can be read. The parameters of this function can be read using functions: LL_ADC_INJ_GetTriggerSource() LL_ADC_INJ_GetTriggerEdge() LL_ADC_INJ_GetSequencerRanks()
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32H5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_ConfigQueueContext
- JSQR JEXTEN LL_ADC_INJ_ConfigQueueContext
- JSQR JL LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ1 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ2 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ3 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ4 LL_ADC_INJ_ConfigQueueContext

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)
- **SamplingTime:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_24CYCLES_5
  - LL_ADC_SAMPLINGTIME_47CYCLES_5
  - LL_ADC_SAMPLINGTIME_92CYCLES_5
  - LL_ADC_SAMPLINGTIME_247CYCLES_5
  - LL_ADC_SAMPLINGTIME_640CYCLES_5

  (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

  (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1.
- (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.
- (3) On STM32H5, fast channel allows: 2.5 (sampling) + 12.5 (conversion 12b) = 15 ADC clock cycles (fADC) Other channels are slow channels: 6.5 (sampling) + 12.5 (conversion 12b) = 19 ADC clock cycles (fADC)

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_2CYCLES_5 (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_24CYCLES_5
  - LL_ADC_SAMPLINGTIME_47CYCLES_5
  - LL_ADC_SAMPLINGTIME_92CYCLES_5
  - LL_ADC_SAMPLINGTIME_247CYCLES_5
  - LL_ADC_SAMPLINGTIME_640CYCLES_5

  (1) On some devices, ADC sampling time 2.5 ADC clock cycles can be replaced by 3.5 ADC clock cycles. Refer to function LL_ADC_SetSamplingTimeCommonConfig().

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name

__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)

Function description

Set mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
- **SingleDiff:** This parameter can be a combination of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_SetChannelSingleDiff

LL_ADC_GetChannelSingleDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15

Return values

- **0:** channel in single-ended mode, else: channel in differential mode

Notes

- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Therefore, to ensure a channel is configured in single-ended mode, the configuration of channel itself and the channel 'i-1' must be read back (to ensure that the selected channel channel has not been configured in differential mode by the previous channel).
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- One or several values can be selected. In this case, the value returned is null if all channels are in single ended-mode. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_GetChannelSingleDiff

LL_ADC_SetAnalogWDMonitChannels

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)

Function description

Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ

  - LL_ADC_AWD_CHANNEL_15_REG (0)
  - LL_ADC_AWD_CHANNEL_15_INJ (0)
  - LL_ADC_AWD_CHANNEL_15_REG_INJ
  - LL_ADC_AWD_CHANNEL_16_REG (0)
  - LL_ADC_AWD_CHANNEL_16_INJ (0)
  - LL_ADC_AWD_CHANNEL_16_REG_INJ
  - LL_ADC_AWD_CHANNEL_17_REG (0)
  - LL_ADC_AWD_CHANNEL_17_INJ (0)
  - LL_ADC_AWD_CHANNEL_17_REG_INJ
  - LL_ADC_AWD_CHANNEL_18_REG (0)
  - LL_ADC_AWD_CHANNEL_18_INJ (0)
  - LL_ADC_AWD_CHANNEL_18_REG_INJ
  - LL_ADC_AWD_CHANNEL_19_REG (0)
  - LL_ADC_AWD_CHANNEL_19_INJ (0)
  - LL_ADC_AWD_CHANNEL_19_REG_INJ
  - LL_ADC_AWD_CH_VREFINT_REG (0)(1)
  - LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
  - LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
  - LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(1)
  - LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(1)
  - LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)
  - LL_ADC_AWD_CH_VBAT_REG (0)(2)
  - LL_ADC_AWD_CH_VBAT_INJ (0)(2)
  - LL_ADC_AWD_CH_VBAT_REG_INJ (2)
  - LL_ADC_AWD_CH_VDDCORE_REG (0)(2)
  - LL_ADC_AWD_CH_VDDCORE_INJ (0)(2)
  - LL_ADC_AWD_CH_VDDCORE_REG_INJ (2)

  (0) On STM32H5, parameter available only on analog watchdog number: AWD1.

- (1) On STM32H563xx/573xx, parameter available only on ADC instance: ADC1. (2) On STM32H563xx/573xx, parameter available only on ADC instance: ADC2.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_SetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_SetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2 (1)
  - LL_ADC_AWD3 (1)

  (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ

(0) On STM32H5, parameter available only on analog watchdog number: AWD1.

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_GetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_ConfigAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)

Function description

Set ADC analog watchdog thresholds value of both thresholds high and low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdHighValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits). Examples: Oversampling ratio and shift selected to have ADC conversion data on 12 bits (ratio 16 and shift 4, or ratio 32 and shift 5, ...): ADC analog watchdog thresholds must be divided by 16.Oversampling ratio and shift selected to have ADC conversion data on 14 bits (ratio 16 and shift 2, or ratio 32 and shift 3, ...): ADC analog watchdog thresholds must be divided by 4.Oversampling ratio and shift selected to have ADC conversion data on 16 bits (ratio 16 and shift none, or ratio 32 and shift 1, ...): ADC analog watchdog thresholds match directly to ADC data register.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 HT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 HT3 LL_ADC_ConfigAnalogWDThresholds
- TR1 LT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 LT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 LT3 LL_ADC_ConfigAnalogWDThresholds

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits). Examples: Oversampling ratio and shift selected to have ADC conversion data on 12 bits (ratio 16 and shift 4, or ratio 32 and shift 5, ...): ADC analog watchdog thresholds must be divided by 16.Oversampling ratio and shift selected to have ADC conversion data on 14 bits (ratio 16 and shift 2, or ratio 32 and shift 3, ...): ADC analog watchdog thresholds must be divided by 4.Oversampling ratio and shift selected to have ADC conversion data on 16 bits (ratio 16 and shift none, or ratio 32 and shift 1, ...): ADC analog watchdog thresholds match directly to ADC data register.
- On this STM32 series, setting of this feature is not conditioned to ADC state: ADC can be disabled, enabled with or without conversion on going on either ADC groups regular or injected.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_SetAnalogWDThresholds
- TR2 HT2 LL_ADC_SetAnalogWDThresholds
- TR3 HT3 LL_ADC_SetAnalogWDThresholds
- TR1 LT1 LL_ADC_SetAnalogWDThresholds
- TR2 LT2 LL_ADC_SetAnalogWDThresholds
- TR3 LT3 LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (const ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)

Function description

Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
  - LL_ADC_AWD_THRESHOLDS_HIGH_LOW

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_GetAnalogWDThresholds
- TR2 HT2 LL_ADC_GetAnalogWDThresholds
- TR3 HT3 LL_ADC_GetAnalogWDThresholds
- TR1 LT1 LL_ADC_GetAnalogWDThresholds
- TR2 LT2 LL_ADC_GetAnalogWDThresholds
- TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_SetAWDFilteringConfiguration

Function name

__STATIC_INLINE void LL_ADC_SetAWDFilteringConfiguration (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t FilteringConfig)

Function description

Set ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
- **FilteringConfig:** This parameter can be one of the following values:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_SetAWDFilteringConfiguration

LL_ADC_GetAWDFilteringConfiguration

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDFilteringConfiguration (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1

Return values

- **Returned:** value can be:
  - LL_ADC_AWD_FILTERING_NONE
  - LL_ADC_AWD_FILTERING_2SAMPLES
  - LL_ADC_AWD_FILTERING_3SAMPLES
  - LL_ADC_AWD_FILTERING_4SAMPLES
  - LL_ADC_AWD_FILTERING_5SAMPLES
  - LL_ADC_AWD_FILTERING_6SAMPLES
  - LL_ADC_AWD_FILTERING_7SAMPLES
  - LL_ADC_AWD_FILTERING_8SAMPLES

Notes

- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_GetAWDFilteringConfiguration

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 series).

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Return values

- **None:**

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_SetOverSamplingScope
- CFGR2 JOVSE LL_ADC_SetOverSamplingScope
- CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 series).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_GetOverSamplingScope
- CFGR2 JOVSE LL_ADC_GetOverSamplingScope
- CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)

Function description

Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance
- **OverSamplingDiscont:** This parameter can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Return values

- **None:**

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32 series, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name

__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)

Function description

Set ADC oversampling (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance
- **Ratio:** This parameter can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256
- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8

Return values

- **None:**

Notes

- This function set the 2 items of oversampling configuration: ratioshift
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift
- CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** This parameter can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256

Reference Manual to LL API cross reference:

- CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling shift (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_SetMultimode

Function name

__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)

Function description

Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Return values

- **None:**

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultimode (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_GetMultimode

LL_ADC_SetMultiDMATransfer

Function name

__STATIC_INLINE void LL_ADC_SetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiDMATransfer)

Function description

Set ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiDMATransfer:** This parameter can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

Return values

- **None:**

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CCR MDMA LL_ADC_SetMultiDMATransfer
- CCR DMACFG LL_ADC_SetMultiDMATransfer

LL_ADC_GetMultiDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiDMATransfer (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B
  - LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CCR MDMA LL_ADC_GetMultiDMATransfer
- CCR DMACFG LL_ADC_GetMultiDMATransfer

LL_ADC_SetMultiTwoSamplingDelay

Function name

__STATIC_INLINE void LL_ADC_SetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiTwoSamplingDelay)

Function description

Set ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiTwoSamplingDelay:** This parameter can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES (3)

  (1) Parameter available only if ADC resolution is 12, 10 or 8 bits.
- (2) Parameter available only if ADC resolution is 12 or 10 bits.
- (3) Parameter available only if ADC resolution is 12 bits.

Return values

- **None:**

Notes

- The sampling delay range depends on ADC resolution: ADC resolution 12 bits can have maximum delay of 12 cycles.ADC resolution 10 bits can have maximum delay of 10 cycles.ADC resolution 8 bits can have maximum delay of 8 cycles.ADC resolution 6 bits can have maximum delay of 6 cycles.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_SetMultiTwoSamplingDelay

LL_ADC_GetMultiTwoSamplingDelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiTwoSamplingDelay (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES (3)

  (1) Parameter available only if ADC resolution is 12, 10 or 8 bits.
- (2) Parameter available only if ADC resolution is 12 or 10 bits.
- (3) Parameter available only if ADC resolution is 12 bits.

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_GetMultiTwoSamplingDelay

LL_ADC_EnableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Put ADC instance in deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Disable ADC deep power down mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** deep power down is disabled, 1: deep power down is enabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_EnableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Enable ADC instance internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tADCVREG_STUP. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Disable ADC internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance internal voltage regulator state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** internal regulator is disabled, 1: internal regulator is enabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Enable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.
- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Disable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance enable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is disabled, 1: ADC is enabled.

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name

__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- On this STM32 series, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration
- CR ADCALDIF LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (const ADC_TypeDef * ADCx)

Function description

Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_StartSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StartSamplingPhase (ADC_TypeDef * ADCx)

Function description

Start ADC sampling phase for sampling time trigger mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger source
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StartSamplingPhase

LL_ADC_REG_StopSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StopSamplingPhase (ADC_TypeDef * ADCx)

Function description

Stop ADC sampling phase for sampling time trigger mode and start conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger sourceLL_ADC_REG_StartSamplingPhase has been called to start the sampling phase
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StopSamplingPhase

LL_ADC_REG_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData6

LL_ADC_REG_ReadMultiConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConversionData32 (const ADC_Common_TypeDef * ADCxy_COMMON, uint32_t ConversionData)

Function description

Get ADC multimode conversion data of ADC master, ADC slave or raw data with ADC master and slave concatenated.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ConversionData:** This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
  - LL_ADC_MULTI_MASTER_SLAVE

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- If raw data with ADC master and slave concatenated is retrieved, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). (however this macro is mainly intended for multimode transfer by DMA, because this function can do the same by getting multimode conversion data of ADC master or ADC slave separately).

Reference Manual to LL API cross reference:

- CDR RDATA_MST LL_ADC_REG_ReadMultiConversionData32
- CDR RDATA_SLV LL_ADC_REG_ReadMultiConversionData32

LL_ADC_INJ_StartConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL_ADC_INJ_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_INJ_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData6 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData6
- JDR2 JDATA LL_ADC_INJ_ReadConversionData6
- JDR3 JDATA LL_ADC_INJ_ReadConversionData6
- JDR4 JDATA LL_ADC_INJ_ReadConversionData6

LL_ADC_IsActiveFlag_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JQOVF (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_IsActiveFlag_JQOVF

LL_ADC_IsActiveFlag_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 1 flag.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_JQOVF

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JQOVF (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_ClearFlag_JQOVF

LL_ADC_ClearFlag_AWD1

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_IsActiveFlag_MST_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_ADRDY (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_MST LL_ADC_IsActiveFlag_MST_ADRDY

LL_ADC_IsActiveFlag_SLV_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_ADRDY (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_SLV LL_ADC_IsActiveFlag_SLV_ADRDY

LL_ADC_IsActiveFlag_MST_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_MST LL_ADC_IsActiveFlag_MST_EOC

LL_ADC_IsActiveFlag_SLV_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_SLV LL_ADC_IsActiveFlag_SLV_EOC

LL_ADC_IsActiveFlag_MST_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_MST LL_ADC_IsActiveFlag_MST_EOS

LL_ADC_IsActiveFlag_SLV_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_SLV LL_ADC_IsActiveFlag_SLV_EOS

LL_ADC_IsActiveFlag_MST_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_OVR (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_MST LL_ADC_IsActiveFlag_MST_OVR

LL_ADC_IsActiveFlag_SLV_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_OVR (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_SLV LL_ADC_IsActiveFlag_SLV_OVR

LL_ADC_IsActiveFlag_MST_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOSMP (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_MST LL_ADC_IsActiveFlag_MST_EOSMP

LL_ADC_IsActiveFlag_SLV_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOSMP (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_SLV LL_ADC_IsActiveFlag_SLV_EOSMP

LL_ADC_IsActiveFlag_MST_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_MST LL_ADC_IsActiveFlag_MST_JEOC

LL_ADC_IsActiveFlag_SLV_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOC (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_SLV LL_ADC_IsActiveFlag_SLV_JEOC

LL_ADC_IsActiveFlag_MST_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_MST LL_ADC_IsActiveFlag_MST_JEOS

LL_ADC_IsActiveFlag_SLV_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOS (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_SLV LL_ADC_IsActiveFlag_SLV_JEOS

LL_ADC_IsActiveFlag_MST_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JQOVF (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_MST LL_ADC_IsActiveFlag_MST_JQOVF

LL_ADC_IsActiveFlag_SLV_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JQOVF (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_SLV LL_ADC_IsActiveFlag_SLV_JQOVF

LL_ADC_IsActiveFlag_MST_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD1 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 1 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_MST LL_ADC_IsActiveFlag_MST_AWD1

LL_ADC_IsActiveFlag_SLV_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD1 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode analog watchdog 1 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_SLV LL_ADC_IsActiveFlag_SLV_AWD1

LL_ADC_IsActiveFlag_MST_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD2 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_MST LL_ADC_IsActiveFlag_MST_AWD2

LL_ADC_IsActiveFlag_SLV_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD2 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_SLV LL_ADC_IsActiveFlag_SLV_AWD2

LL_ADC_IsActiveFlag_MST_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD3 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_MST LL_ADC_IsActiveFlag_MST_AWD3

LL_ADC_IsActiveFlag_SLV_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD3 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_SLV LL_ADC_IsActiveFlag_SLV_AWD3

LL_ADC_EnableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Enable ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)

Function description

Enable ADC group regular interruption overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_EnableIT_JQOVF

LL_ADC_EnableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_DisableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_DisableIT_JQOVF

LL_ADC_DisableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_IsEnabledIT_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JQOVF (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected context queue overflow interrupt state (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_IsEnabledIT_JQOVF

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_CommonDeInit

Function name

ErrorStatus LL_ADC_CommonDeInit (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are de-initialized
  - ERROR: not applicable

Notes

- This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, const LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are initialized
  - ERROR: ADC common registers are not initialized

Notes

- The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name

void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_DeInit

Function name

ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **ADCx:** ADC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are de-initialized
  - ERROR: ADC registers are not de-initialized

Notes

- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().
- If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, const LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **pADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 series). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 series. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **pADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_REG_Init

Function name

ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, const LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **ADCx:** ADC instance
- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 series. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_INJ_Init

Function name

ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, const LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Initialize some features of ADC group injected.

Parameters

- **ADCx:** ADC instance
- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 series. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();
- Caution if feature ADC group injected contexts queue is enabled (refer to with function LL_ADC_INJ_SetQueueMode()): using successively several times this function will appear as having no effect. To set several features of ADC group injected, use function LL_ADC_INJ_ConfigQueueContext().

LL_ADC_INJ_StructInit

Function name

void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters

- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

85.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

85.3.1 ADC

ADC

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE

ADC analog watchdog monitoring disabled

LL_ADC_AWD_ALL_CHANNELS_REG

ADC analog watchdog monitoring of all channels, converted by group regular only

LL_ADC_AWD_ALL_CHANNELS_INJ

ADC analog watchdog monitoring of all channels, converted by group injected only

LL_ADC_AWD_ALL_CHANNELS_REG_INJ

ADC analog watchdog monitoring of all channels, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_0_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by group injected only

LL_ADC_AWD_CHANNEL_0_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_1_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by group injected only

LL_ADC_AWD_CHANNEL_1_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_2_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by group injected only

LL_ADC_AWD_CHANNEL_2_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_3_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by group injected only

LL_ADC_AWD_CHANNEL_3_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_4_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_5_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by group injected only

LL_ADC_AWD_CHANNEL_5_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_6_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by group injected only

LL_ADC_AWD_CHANNEL_6_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_7_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by group injected only

LL_ADC_AWD_CHANNEL_7_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_8_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by group injected only

LL_ADC_AWD_CHANNEL_8_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_9_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by group injected only

LL_ADC_AWD_CHANNEL_9_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_10_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by group injected only

LL_ADC_AWD_CHANNEL_10_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_11_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by group injected only

LL_ADC_AWD_CHANNEL_11_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_12_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by group injected only

LL_ADC_AWD_CHANNEL_12_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_13_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_14_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by group only

LL_ADC_AWD_CHANNEL_14_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring monitoring of ADC channel ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_15_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN15, converted by group injected only

LL_ADC_AWD_CHANNEL_15_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_16_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by group injected only

LL_ADC_AWD_CHANNEL_16_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_17_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by group injected only

LL_ADC_AWD_CHANNEL_17_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_18_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by group injected only

LL_ADC_AWD_CHANNEL_18_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_19_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_19_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_19_REG_INJ

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CH_VREFINT_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_VREFINT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_VREFINT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_TEMPSENSOR_REG

ADC analog watchdog monitoring of ADC internal channel connected to internal temperature sensor, converted by group regular only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_TEMPSENSOR_INJ

ADC analog watchdog monitoring of ADC internal channel connected to internal temperature sensor, converted by group injected only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to internal temperature sensor, converted by either group regular or injected. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_AWD_CH_VBAT_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda, converted by group regular only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VBAT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda, converted by group injected only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VBAT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VDDCORE_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vddcore, converted by group regular only On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VDDCORE_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vddcore, converted by group injected only. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VDDCORE_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vddcore, converted by either group regular or injected. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_AWD_CH_VCORE_REG

LL_ADC_AWD_CH_VCORE_INJ

LL_ADC_AWD_CH_VCORE_REG_INJ

Analog watchdog - filtering config

LL_ADC_AWD_FILTERING_NONE

ADC analog watchdog no filtering, one out-of-window sample is needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_2SAMPLES

ADC analog watchdog 2 out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_3SAMPLES

ADC analog watchdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_4SAMPLES

ADC analog watchdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_5SAMPLES

ADC analog watchdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_6SAMPLES

ADC analog watchdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_7SAMPLES

ADC analog watchdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_8SAMPLES

ADC analog watchdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

Analog watchdog - Analog watchdog number

LL_ADC_AWD1

ADC analog watchdog number 1

LL_ADC_AWD2

ADC analog watchdog number 2

LL_ADC_AWD3

ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH

ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOLD_LOW

ADC analog watchdog threshold low

LL_ADC_AWD_THRESHOLDS_HIGH_LOW

ADC analog watchdog both thresholds high and low concatenated into the same data

ADC instance - Channel number

LL_ADC_CHANNEL_0

ADC channel ADCx_IN0

LL_ADC_CHANNEL_1

ADC channel ADCx_IN1

LL_ADC_CHANNEL_2

ADC channel ADCx_IN2

LL_ADC_CHANNEL_3

ADC channel ADCx_IN3

LL_ADC_CHANNEL_4

ADC channel ADCx_IN4

LL_ADC_CHANNEL_5

ADC channel ADCx_IN5

LL_ADC_CHANNEL_6

ADC channel ADCx_IN6

LL_ADC_CHANNEL_7

ADC channel ADCx_IN7

LL_ADC_CHANNEL_8

ADC channel ADCx_IN8

LL_ADC_CHANNEL_9

ADC channel ADCx_IN9

LL_ADC_CHANNEL_10

ADC channel ADCx_IN10

LL_ADC_CHANNEL_11

ADC channel ADCx_IN11

LL_ADC_CHANNEL_12

ADC channel ADCx_IN12

LL_ADC_CHANNEL_13

ADC channel ADCx_IN13

LL_ADC_CHANNEL_14

ADC channel ADCx_IN14

LL_ADC_CHANNEL_15

ADC channel ADCx_IN15

LL_ADC_CHANNEL_16

ADC channel ADCx_IN16

LL_ADC_CHANNEL_17

ADC channel ADCx_IN17

LL_ADC_CHANNEL_18

ADC channel ADCx_IN18

LL_ADC_CHANNEL_19

ADC channel ADCx_IN19

LL_ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_CHANNEL_TEMPSENSOR

ADC internal channel connected to internal temperature sensor. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC1.

LL_ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have channel voltage always below Vdda. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_CHANNEL_VDDCORE

ADC internal channel connected to Vddcore. On STM32H563xx/573xx, ADC channel available only on ADC instance: ADC2.

LL_ADC_CHANNEL_VCORE

Channel - Sampling time

LL_ADC_SAMPLINGTIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_6CYCLES_5

Sampling time 6.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_24CYCLES_5

Sampling time 24.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_47CYCLES_5

Sampling time 47.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_92CYCLES_5

Sampling time 92.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_247CYCLES_5

Sampling time 247.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_640CYCLES_5

Sampling time 640.5 ADC clock cycles

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

LL_ADC_BOTH_SINGLE_DIFF_ENDED

ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock without prescaler

LL_ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock with prescaler division by 2

LL_ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

LL_ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

LL_ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

LL_ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

LL_ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

LL_ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

LL_ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

LL_ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

LL_ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

LL_ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

LL_ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE

ADC measurement paths all disabled

LL_ADC_PATH_INTERNAL_VREFINT

ADC measurement path to internal channel VrefInt

LL_ADC_PATH_INTERNAL_TEMPSENSOR

ADC measurement path to internal channel temperature sensor

LL_ADC_PATH_INTERNAL_VBAT

ADC measurement path to internal channel Vbat

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

LL_ADC_DATA_ALIGN_LEFT

ADC conversion data alignment: left aligned (alignment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_ADRDY

ADC flag ADC instance ready

LL_ADC_FLAG_EOC

ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS

ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR

ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP

ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_JEOC

ADC flag ADC group injected end of unitary conversion

LL_ADC_FLAG_JEOS

ADC flag ADC group injected end of sequence conversions

LL_ADC_FLAG_JQOVF

ADC flag ADC group injected contexts queue overflow

LL_ADC_FLAG_AWD1

ADC flag ADC analog watchdog 1

LL_ADC_FLAG_AWD2

ADC flag ADC analog watchdog 2

LL_ADC_FLAG_AWD3

ADC flag ADC analog watchdog 3

LL_ADC_FLAG_ADRDY_MST

ADC flag ADC multimode master instance ready

LL_ADC_FLAG_ADRDY_SLV

ADC flag ADC multimode slave instance ready

LL_ADC_FLAG_EOC_MST

ADC flag ADC multimode master group regular end of unitary conversion

LL_ADC_FLAG_EOC_SLV

ADC flag ADC multimode slave group regular end of unitary conversion

LL_ADC_FLAG_EOS_MST

ADC flag ADC multimode master group regular end of sequence conversions

LL_ADC_FLAG_EOS_SLV

ADC flag ADC multimode slave group regular end of sequence conversions

LL_ADC_FLAG_OVR_MST

ADC flag ADC multimode master group regular overrun

LL_ADC_FLAG_OVR_SLV

ADC flag ADC multimode slave group regular overrun

LL_ADC_FLAG_EOSMP_MST

ADC flag ADC multimode master group regular end of sampling phase

LL_ADC_FLAG_EOSMP_SLV

ADC flag ADC multimode slave group regular end of sampling phase

LL_ADC_FLAG_JEOC_MST

ADC flag ADC multimode master group injected end of unitary conversion

LL_ADC_FLAG_JEOC_SLV

ADC flag ADC multimode slave group injected end of unitary conversion

LL_ADC_FLAG_JEOS_MST

ADC flag ADC multimode master group injected end of sequence conversions

LL_ADC_FLAG_JEOS_SLV

ADC flag ADC multimode slave group injected end of sequence conversions

LL_ADC_FLAG_JQOVF_MST

ADC flag ADC multimode master group injected contexts queue overflow

LL_ADC_FLAG_JQOVF_SLV

ADC flag ADC multimode slave group injected contexts queue overflow

LL_ADC_FLAG_AWD1_MST

ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV

ADC flag ADC multimode slave analog watchdog 1 of the ADC slave

LL_ADC_FLAG_AWD2_MST

ADC flag ADC multimode master analog watchdog 2 of the ADC master

LL_ADC_FLAG_AWD2_SLV

ADC flag ADC multimode slave analog watchdog 2 of the ADC slave

LL_ADC_FLAG_AWD3_MST

ADC flag ADC multimode master analog watchdog 3 of the ADC master

LL_ADC_FLAG_AWD3_SLV

ADC flag ADC multimode slave analog watchdog 3 of the ADC slave

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

LL_ADC_GROUP_INJECTED

ADC group injected (not available on all STM32 devices)

LL_ADC_GROUP_REGULAR_INJECTED

ADC both groups regular and injected

Definitions of constants used by helper macro

LL_ADC_TEMPERATURE_CALC_ERROR

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US

Delay for ADC stabilization time (ADC voltage regulator start-up time)

LL_ADC_DELAY_VREFINT_STAB_US

Delay for internal voltage reference stabilization time

LL_ADC_DELAY_TEMPSENSOR_STAB_US

Delay for temperature sensor stabilization time

LL_ADC_DELAY_TEMPSENSOR_BUFFER_STAB_US

Delay for temperature sensor buffer stabilization time (starting from ADC enable, refer to

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC group injected - Context queue mode

LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE

LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

LL_ADC_INJ_QUEUE_DISABLE

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE

ADC group injected sequencer discontinuous mode disable

LL_ADC_INJ_SEQ_DISCONT_1RANK

ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1

ADC group inj. sequencer rank 1

LL_ADC_INJ_RANK_2

ADC group inj. sequencer rank 2

LL_ADC_INJ_RANK_3

ADC group inj. sequencer rank 3

LL_ADC_INJ_RANK_4

ADC group inj. sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE

ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS

ADC group injected sequencer enable with 2 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS

ADC group injected sequencer enable with 3 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING

ADC group injected conversion trigger polarity set to rising edge

LL_ADC_INJ_TRIG_EXT_FALLING

ADC group injected conversion trigger polarity set to falling edge

LL_ADC_INJ_TRIG_EXT_RISINGFALLING

ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE

ADC group injected conversion trigger internal: SW start.

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_CH4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_CH1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM1_CH1

ADC group injected conversion trigger from external peripheral: LPTIM1 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM2_CH1

ADC group injected conversion trigger from external peripheral: LPTIM2 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_INJ_TRIG_EXT_TIM8_CH4

ADC group injected conversion trigger from external peripheral: TIM9 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO

ADC group injected conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM9 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_INJ_TRIG_EXT_TIM15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT

ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.

LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_EOC

ADC interruption ADC group regular end of unitary conversion

LL_ADC_IT_EOS

ADC interruption ADC group regular end of sequence conversions

LL_ADC_IT_OVR

ADC interruption ADC group regular overrun

LL_ADC_IT_EOSMP

ADC interruption ADC group regular end of sampling phase

LL_ADC_IT_JEOC

ADC interruption ADC group injected end of unitary conversion

LL_ADC_IT_JEOS

ADC interruption ADC group injected end of sequence conversions

LL_ADC_IT_JQOVF

ADC interruption ADC group injected contexts queue overflow

LL_ADC_IT_AWD1

ADC interruption ADC analog watchdog 1

LL_ADC_IT_AWD2

ADC interruption ADC analog watchdog 2

LL_ADC_IT_AWD3

ADC interruption ADC analog watchdog 3

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE

No ADC low power mode activated

LL_ADC_LP_AUTOWAIT

ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

Multimode - DMA transfer

LL_ADC_MULTI_REG_DMA_EACH_ADC

ADC multimode group regular conversions are transferred by DMA: each ADC uses its own DMA channel, with its individual DMA transfer settings

LL_ADC_MULTI_REG_DMA_LIMIT_RES12_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC(DMA of ADC master), in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting for ADC resolution of 12 and 10 bits

LL_ADC_MULTI_REG_DMA_LIMIT_RES8_6B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC(DMA of ADC master), in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular. Setting for ADC resolution of 8 and 6 bits

LL_ADC_MULTI_REG_DMA_UNLMT_RES12_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC(DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. Setting for ADC resolution of 12 and 10 bits

LL_ADC_MULTI_REG_DMA_UNLMT_RES8_6B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master), in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular. Setting for ADC resolution of 8 and 6 bits

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER

In multimode, selection among several ADC instances: ADC master

LL_ADC_MULTI_SLAVE

In multimode, selection among several ADC instances: ADC slave

LL_ADC_MULTI_MASTER_SLAVE

In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

LL_ADC_MULTI_DUAL_REG_SIMULT

ADC dual mode enabled: group regular simultaneous

LL_ADC_MULTI_DUAL_REG_INTERL

ADC dual mode enabled: Combined group regular interleaved

LL_ADC_MULTI_DUAL_INJ_SIMULT

ADC dual mode enabled: group injected simultaneous

LL_ADC_MULTI_DUAL_INJ_ALTERN

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not SW start)

LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC instance - Offset instance

LL_ADC_OFFSET_1

ADC offset instance 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_2

ADC offset instance 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_3

ADC offset instance 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

LL_ADC_OFFSET_4

ADC offset instance 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or injected)

ADC instance - Offset saturation mode

LL_ADC_OFFSET_SATURATION_DISABLE

ADC offset saturation is disabled (among ADC selected offset instance 1, 2, 3 or 4)

LL_ADC_OFFSET_SATURATION_ENABLE

ADC offset saturation is enabled (among ADC selected offset instance 1, 2, 3 or 4)

ADC instance - Offset sign

LL_ADC_OFFSET_SIGN_NEGATIVE

ADC offset is negative

LL_ADC_OFFSET_SIGN_POSITIVE

ADC offset is positive

ADC instance - Offset state

LL_ADC_OFFSET_DISABLE

ADC offset disabled (setting offset instance wise)

LL_ADC_OFFSET_ENABLE

ADC offset enabled (setting offset instance wise)

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

LL_ADC_OVS_REG_DISCONT

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

LL_ADC_OVS_RATIO_2

ADC oversampling ratio of 2 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_4

ADC oversampling ratio of 4 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_8

ADC oversampling ratio of 8 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_16

ADC oversampling ratio of 16 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_32

ADC oversampling ratio of 32 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_64

ADC oversampling ratio of 64 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_128

ADC oversampling ratio of 128 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_256

ADC oversampling ratio of 256 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE

ADC oversampling disabled.

LL_ADC_OVS_GRP_REGULAR_CONTINUED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.

LL_ADC_OVS_GRP_REGULAR_RESUMED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

LL_ADC_OVS_GRP_INJECTED

ADC oversampling on conversions of ADC group injected.

LL_ADC_OVS_GRP_INJ_REG_RESUMED

ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data right shift

LL_ADC_OVS_SHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1

ADC oversampling right shift of 1 (sum of the ADC conversions data (after OVS ratio) is divided by 2 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_2

ADC oversampling right shift of 2 (sum of the ADC conversions data (after OVS ratio) is divided by 4 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3

ADC oversampling right shift of 3 (sum of the ADC conversions data (after OVS ratio) is divided by 8 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_4

ADC oversampling right shift of 4 (sum of the ADC conversions data (after OVS ratio) is divided by 16 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_5

ADC oversampling right shift of 5 (sum of the ADC conversions data (after OVS ratio) is divided by 32 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_6

ADC oversampling right shift of 6 (sum of the ADC conversions data (after OVS ratio) is divided by 64 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_7

ADC oversampling right shift of 7 (sum of the ADC conversions data (after OVS ratio) is divided by 128 to result as oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_8

ADC oversampling right shift of 8 (sum of the ADC conversions data (after OVS ratio) is divided by 256 to result as oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

LL_ADC_DMA_REG_REGULAR_DATA_MULTI

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE

ADC conversions are not transferred by DMA

LL_ADC_REG_DMA_TRANSFER_LIMITED

ADC conversion data are transferred by DMA in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sampling mode

LL_ADC_REG_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

LL_ADC_REG_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: trigger rising edge for start sampling, trigger falling edge for stop sampling and start conversion

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE

ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCONT_1RANK

ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

LL_ADC_REG_SEQ_DISCONT_2RANKS

ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks

LL_ADC_REG_SEQ_DISCONT_3RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks

LL_ADC_REG_SEQ_DISCONT_4RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks

LL_ADC_REG_SEQ_DISCONT_5RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks

LL_ADC_REG_SEQ_DISCONT_6RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks

LL_ADC_REG_SEQ_DISCONT_7RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks

LL_ADC_REG_SEQ_DISCONT_8RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8

ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9

ADC group regular sequencer rank 9

LL_ADC_REG_RANK_10

ADC group regular sequencer rank 10

LL_ADC_REG_RANK_11

ADC group regular sequencer rank 11

LL_ADC_REG_RANK_12

ADC group regular sequencer rank 12

LL_ADC_REG_RANK_13

ADC group regular sequencer rank 13

LL_ADC_REG_RANK_14

ADC group regular sequencer rank 14

LL_ADC_REG_RANK_15

ADC group regular sequencer rank 15

LL_ADC_REG_RANK_16

ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS

ADC group regular sequencer enable with 9 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS

ADC group regular sequencer enable with 10 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS

ADC group regular sequencer enable with 11 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS

ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS

ADC group regular sequencer enable with 13 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS

ADC group regular sequencer enable with 14 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS

ADC group regular sequencerenable with 15 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING

ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING

ADC group regular conversion trigger polarity set to falling edge

LL_ADC_REG_TRIG_EXT_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE

ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_TIM1_CH1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_CH4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE15

ADC group regular conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM3 event OUT. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_CH4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_REG_TRIG_EXT_TIM8_TRGO

ADC group regular conversion trigger from external peripheral: TIM12 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_REG_TRIG_EXT_TIM8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_REG_TRIG_EXT_TIM4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

LL_ADC_REG_TRIG_EXT_TIM15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting). Specific to devices: STM32H563/H573xx.

ADC instance - Resolution

LL_ADC_RESOLUTION_12B

ADC resolution 12 bits

LL_ADC_RESOLUTION_10B

ADC resolution 10 bits

LL_ADC_RESOLUTION_8B

ADC resolution 8 bits

LL_ADC_RESOLUTION_6B

ADC resolution 6 bits

ADC instance - ADC sampling time common configuration

LL_ADC_SAMPLINGTIME_COMMON_DEFAULT

ADC sampling time let to default settings.

LL_ADC_SAMPLINGTIME_COMMON_3C5_REPL_2C5

ADC additional sampling time 3.5 ADC clock cycles replacing 2.5 ADC clock cycles (this applies to all channels mapped with selection sampling time 2.5 ADC clock cycles, whatever channels mapped on ADC groups regular or injected).

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(4)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)(4)
  - LL_ADC_CHANNEL_VBAT (2)(4)
  - LL_ADC_CHANNEL_VDDCORE (2)(4)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (2)
  - LL_ADC_CHANNEL_VDDCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)(4)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)(4)
  - LL_ADC_CHANNEL_VBAT (2)(4)
  - LL_ADC_CHANNEL_VDDCORE (2)(4)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)(1)
- LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)
- LL_ADC_AWD_CH_VBAT_REG (0)(2)
- LL_ADC_AWD_CH_VBAT_INJ (0)(2)
- LL_ADC_AWD_CH_VBAT_REG_INJ (2)
- LL_ADC_AWD_CH_VDDCORE_REG (0)(2)
- LL_ADC_AWD_CH_VDDCORE_INJ (0)(2)
- LL_ADC_AWD_CH_VDDCORE_REG_INJ (2)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example:
  LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1,
  __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param >, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_6_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW

Description:

- Helper macro to get the ADC analog watchdog threshold high or low from raw value containing both thresholds concatenated.

Parameters:

- __AWD_THRESHOLD_TYPE__: This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- __AWD_THRESHOLDS__: Value between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, to get analog watchdog threshold high from the register raw value: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(LL_ADC_AWD_THRESHOLD_HIGH, <raw_value_with_both_thresholds>);

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x7F
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x7F

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_MULTI_INSTANCE_MASTER

Description:

- Helper macro to select, from a ADC instance, to which ADC instance it has a dependence in multimode (ADC master of the corresponding ADC common instance).

Parameters:

- __ADCx__: ADC instance

Return value:

- __ADCx__: ADC instance master of the corresponding ADC common instance

Notes:

- In case of device with multimode available and a mix of ADC instances compliant and not compliant with multimode feature, ADC instances not compliant with multimode feature are considered as master instances (do not depend to any other ADC instance).

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius) In case or error, value LL_ADC_TEMPERATURE_CALC_ERROR is returned (inconsistent temperature value)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32H5, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On this STM32 series, refer to datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) value (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12 bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

86 LL BUS Generic Driver

86.1 BUS Firmware driver API description

The following section lists the various functions of the BUS library.

86.1.1 Detailed description of functions

LL_AHB_DisableClock

Function name

__STATIC_INLINE void LL_AHB_DisableClock (uint32_t AHBx)

Function description

Disable of AHBx Clock Branch.

Parameters

- **AHBx:** This parameter can be a combination of the following values:
  - LL_AHB_BRANCH_CLK_AHB1
  - LL_AHB_BRANCH_CLK_AHB2
  - LL_AHB_BRANCH_CLK_AHB4 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB1DIS LL_AHB_DisableClock
- CFGR2 AHB2DIS LL_AHB_DisableClock
- CFGR2 AHB4DIS LL_AHB_DisableClock

LL_AHB_EnableClock

Function name

__STATIC_INLINE void LL_AHB_EnableClock (uint32_t AHBx)

Function description

Enable of AHBx Clock Branch.

Parameters

- **AHBx:** This parameter can be a combination of the following values:
  - LL_AHB_BRANCH_CLK_AHB1
  - LL_AHB_BRANCH_CLK_AHB2
  - LL_AHB_BRANCH_CLK_AHB4 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB1DIS LL_AHB_EnableClock
- CFGR2 AHB2DIS LL_AHB_EnableClock
- CFGR2 AHB4DIS LL_AHB_EnableClock

LL_AHB_IsDisabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB_IsDisabledClock (uint32_t AHBx)

Function description

Check if AHBx clock branch is disabled or not.

Parameters

- **AHBx:** This parameter can be a combination of the following values:
  - LL_AHB_BRANCH_CLK_AHB1
  - LL_AHB_BRANCH_CLK_AHB2
  - LL_AHB_BRANCH_CLK_AHB4 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of AHBx bus (1 or 0).

Reference Manual to LL API cross reference:

- CFGR2 AHB1DIS LL_AHB_IsDisabledClock
- CFGR2 AHB2DIS LL_AHB_IsDisabledClock
- CFGR2 AHB4DIS LL_AHB_IsDisabledClock

LL_APB_DisableClock

Function name

__STATIC_INLINE void LL_APB_DisableClock (uint32_t APBx)

Function description

Disable APBx Clock Branch.

Parameters

- **APBx:** This parameter can be a combination of the following values:
  - LL_APB_BRANCH_CLK_APB1
  - LL_APB_BRANCH_CLK_APB2
  - LL_APB_BRANCH_CLK_APB3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB1DIS LL_APB_DisableClock
- CFGR2 APB2DIS LL_APB_DisableClock
- CFGR2 APB3DIS LL_APB_DisableClock

LL_APB_EnableClock

Function name

__STATIC_INLINE void LL_APB_EnableClock (uint32_t APBx)

Function description

Enable of APBx Clock Branch.

Parameters

- **APBx:** This parameter can be a combination of the following values:
  - LL_APB_BRANCH_CLK_APB1
  - LL_APB_BRANCH_CLK_APB2
  - LL_APB_BRANCH_CLK_APB3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB1DIS LL_APB_EnableClock
- CFGR2 APB2DIS LL_APB_EnableClock
- CFGR2 APB3DIS LL_APB_EnableClock

LL_APB_IsDisabledClock

Function name

__STATIC_INLINE uint32_t LL_APB_IsDisabledClock (uint32_t APBx)

Function description

Check if APBx clock branch is disabled or not.

Parameters

- **APBx:** This parameter can be a combination of the following values:
  - LL_APB_BRANCH_CLK_APB1
  - LL_APB_BRANCH_CLK_APB2
  - LL_APB_BRANCH_CLK_APB3

Return values

- **State:** of APBx bus (1 or 0).

Reference Manual to LL API cross reference:

- CFGR2 APB1DIS LL_APB_IsDisabledClock
- CFGR2 APB2DIS LL_APB_IsDisabledClock
- CFGR2 APB3DIS LL_APB_IsDisabledClock

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR GPDMA2EN LL_AHB1_GRP1_EnableClock
- AHB1ENR FLITFEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_EnableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_EnableClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_EnableClock
- AHB1ENR ETHEN LL_AHB1_GRP1_EnableClock
- AHB1ENR ETHTXEN LL_AHB1_GRP1_EnableClock
- AHB1ENR ETHRXEN LL_AHB1_GRP1_EnableClock
- AHB1ENR TZSC1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR BKPRAMEN LL_AHB1_GRP1_EnableClock
- AHB1ENR DCACHE1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_EnableClock

LL_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR GPDMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FLITFEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FMACEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETHEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETHTXEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETHRXEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR TZSC1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR BKPRAMEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DCACHE1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR GPDMA2EN LL_AHB1_GRP1_DisableClock
- AHB1ENR FLITFEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_DisableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_DisableClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_DisableClock
- AHB1ENR ETHEN LL_AHB1_GRP1_DisableClock
- AHB1ENR ETHTXEN LL_AHB1_GRP1_DisableClock
- AHB1ENR ETHRXEN LL_AHB1_GRP1_DisableClock
- AHB1ENR TZSC1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR BKPRAMEN LL_AHB1_GRP1_DisableClock
- AHB1ENR DCACHE1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR GPDMA1RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR GPDMA2RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR CORDICRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR FMACRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR RAMCFGRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR ETHRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR TZSC1RST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR GPDMA1RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR GPDMA2RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CRCRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CORDICRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR FMACRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR RAMCFGRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR ETHRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR TZSC1RST LL_AHB1_GRP1_ReleaseReset

LL_AHB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR GPDMA1LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR GPDMA2LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR FLITFLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR CRCLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR CORDICLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR FMACLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR RAMCFGLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETHLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETHTXLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETHRXLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR TZSC1LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR BKPRAMLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR DCACHE1LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR SRAM1LPEN LL_AHB1_GRP1_EnableClockSleep

LL_AHB1_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if AHB1 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB1LPENR GPDMA1LPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR GPDMA2LPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR FLITFLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR CRCLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR CORDICLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR FMACLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR RAMCFGLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR ETHLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR ETHTXLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR ETHRXLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR TZSC1LPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR BKPRAMLPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR DCACHE1LPEN LL_AHB1_GRP1_IsEnabledClockSleep
- AHB1LPENR SRAM1LPEN LL_AHB1_GRP1_IsEnabledClockSleep

LL_AHB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_GPDMA2
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_CORDIC (*)
  - LL_AHB1_GRP1_PERIPH_FMAC (*)
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_ETH (*)
  - LL_AHB1_GRP1_PERIPH_ETHTX (*)
  - LL_AHB1_GRP1_PERIPH_ETHRX (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_DCACHE1 (*)
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR GPDMA1LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR GPDMA2LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR FLITFLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR CRCLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR CORDICLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR FMACLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR RAMCFGLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETHLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETHTXLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETHRXLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR TZSC1LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR BKPRAMLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR DCACHE1LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR SRAM1LPEN LL_AHB1_GRP1_DisableClockSleep

LL_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOHEN LL_AHB2_GRP1_EnableClock
- AHB2ENR GPIOIEN LL_AHB2_GRP1_EnableClock
- AHB2ENR ADCEN LL_AHB2_GRP1_EnableClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_EnableClock
- AHB2ENR DCMI_PSSIEN LL_AHB2_GRP1_EnableClock
- AHB2ENR AESEN LL_AHB2_GRP1_EnableClock
- AHB2ENR HASHEN LL_AHB2_GRP1_EnableClock
- AHB2ENR RNGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR PKAEN LL_AHB2_GRP1_EnableClock
- AHB2ENR SAESEN LL_AHB2_GRP1_EnableClock
- AHB2ENR SRAM2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR SRAM3EN LL_AHB2_GRP1_EnableClock

LL_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOHEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR GPIOIEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR ADCEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR DCMI_PSSIEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR AESEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR HASHEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR RNGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR PKAEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR SAESEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR SRAM2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR SRAM3EN LL_AHB2_GRP1_IsEnabledClock

LL_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR GPIOAEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOBEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOCEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIODEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOEEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOFEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOHEN LL_AHB2_GRP1_DisableClock
- AHB2ENR GPIOIEN LL_AHB2_GRP1_DisableClock
- AHB2ENR ADCEN LL_AHB2_GRP1_DisableClock
- AHB2ENR DAC1EN LL_AHB2_GRP1_DisableClock
- AHB2ENR DCMI_PSSIEN LL_AHB2_GRP1_DisableClock
- AHB2ENR AESEN LL_AHB2_GRP1_DisableClock
- AHB2ENR HASHEN LL_AHB2_GRP1_DisableClock
- AHB2ENR RNGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR PKAEN LL_AHB2_GRP1_DisableClock
- AHB2ENR SAESEN LL_AHB2_GRP1_DisableClock
- AHB2ENR SRAM2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR SRAM3EN LL_AHB2_GRP1_DisableClock

LL_AHB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RST GPIOARST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOBRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOCRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIODRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOERST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOFRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOGRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOHRST LL_AHB2_GRP1_ForceReset
- AHB2RST GPIOIRST LL_AHB2_GRP1_ForceReset
- AHB2RST ADCRST LL_AHB2_GRP1_ForceReset
- AHB2RST DAC1RST LL_AHB2_GRP1_ForceReset
- AHB2RST DCMI_PSSIRST LL_AHB2_GRP1_ForceReset
- AHB2RST AESRST LL_AHB2_GRP1_ForceReset
- AHB2RST HASHRST LL_AHB2_GRP1_ForceReset
- AHB2RST RNGRST LL_AHB2_GRP1_ForceReset
- AHB2RST PKARST LL_AHB2_GRP1_ForceReset
- AHB2RST SAESRST LL_AHB2_GRP1_ForceReset

LL_AHB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RST GPIOARST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOBRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOCRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIODRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOERST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOFRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOHRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST GPIOIRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST ADCRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST DAC1RST LL_AHB2_GRP1_ReleaseReset
- AHB2RST DCMI_PSSIRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST AESRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST HASHRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST RNGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RST PKARST LL_AHB2_GRP1_ReleaseReset
- AHB2RST SAESRST LL_AHB2_GRP1_ReleaseReset

LL_AHB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB2 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR GPIOALPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOBLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOCLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIODLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOELPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOFLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOGLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOHLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR GPIOILPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR ADCLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR DAC1LPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR DCMI_PSSILPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR AESLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR HASHLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR RNGLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR PKALPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SAESLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SRAM2LPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SRAM3LPEN LL_AHB2_GRP1_EnableClockSleep

LL_AHB2_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if AHB2 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2LPENR GPIOALPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOBLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOCLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIODLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOELPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOFLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOGLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOHLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR GPIOILPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR ADCLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR DAC1LPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR DCMI_PSSILPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR AESLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR HASHLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR RNGLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR PKALPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR SAESLPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR SRAM2LPEN LL_AHB2_GRP1_IsEnabledClockSleep
- AHB2LPENR SRAM3LPEN LL_AHB2_GRP1_IsEnabledClockSleep

LL_AHB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB2 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE (*)
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG (*)
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_ADC
  - LL_AHB2_GRP1_PERIPH_DAC1
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI (*)
  - LL_AHB2_GRP1_PERIPH_AES (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA (*)
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR GPIOALPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOBLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOCLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIODLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOELPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOFLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOGLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOHLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR GPIOILPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR ADCLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR DAC1LPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR DCMI_PSSILPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR AESLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR HASHLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR RNGLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR PKALPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SAESLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SRAM2LPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SRAM3LPEN LL_AHB2_GRP1_DisableClockSleep

LL_AHB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2 (*)
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR OTFDEC1EN LL_AHB4_GRP1_EnableClock
- AHB4ENR SDMMC1EN LL_AHB4_GRP1_EnableClock
- AHB4ENR SDMMC2EN LL_AHB4_GRP1_EnableClock
- AHB4ENR FMCEN LL_AHB4_GRP1_EnableClock
- AHB4ENR OCTOSPI1EN LL_AHB4_GRP1_EnableClock

LL_AHB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB4ENR OTFDEC1EN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR SDMMC1EN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR SDMMC2EN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR FMCEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR OCTOSPI1EN LL_AHB4_GRP1_IsEnabledClock

LL_AHB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR OTFDEC1EN LL_AHB4_GRP1_DisableClock
- AHB4ENR SDMMC1EN LL_AHB4_GRP1_DisableClock
- AHB4ENR SDMMC2EN LL_AHB4_GRP1_DisableClock
- AHB4ENR FMCEN LL_AHB4_GRP1_DisableClock
- AHB4ENR OCTOSPI1EN LL_AHB4_GRP1_DisableClock

LL_AHB4_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTR OTFDEC1RST LL_AHB4_GRP1_ForceReset
- AHB4RSTR SDMMC1RST LL_AHB4_GRP1_ForceReset
- AHB4RSTR SDMMC2RST LL_AHB4_GRP1_ForceReset
- AHB4RSTR FMCRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR OCTOSPI1RST LL_AHB4_GRP1_ForceReset

LL_AHB4_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTR OTFDEC1RST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR SDMMC1RST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR SDMMC2RST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR FMCRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR OCTOSPI1RST LL_AHB4_GRP1_ReleaseReset

LL_AHB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB4 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR OTFDEC1LPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR SDMMC1LPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR SDMMC2LPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR FMCLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR OCTOSPI1LPEN LL_AHB4_GRP1_EnableClockSleep

LL_AHB4_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_AHB4_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if AHB4 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB4LPENR OTFDEC1LPEN LL_AHB4_GRP1_IsEnabledClockSleep
- AHB4LPENR SDMMC1LPEN LL_AHB4_GRP1_IsEnabledClockSleep
- AHB4LPENR SDMMC2LPEN LL_AHB4_GRP1_IsEnabledClockSleep
- AHB4LPENR FMCLPEN LL_AHB4_GRP1_IsEnabledClockSleep
- AHB4LPENR OCTOSPI1LPEN LL_AHB4_GRP1_IsEnabledClockSleep

LL_AHB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB4 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_ALL
  - LL_AHB4_GRP1_PERIPH_OTFDEC
  - LL_AHB4_GRP1_PERIPH_SDMMC1
  - LL_AHB4_GRP1_PERIPH_SDMMC2
  - LL_AHB4_GRP1_PERIPH_FMC
  - LL_AHB4_GRP1_PERIPH_OSPI1

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR OTFDEC1LPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR SDMMC1LPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR SDMMC2LPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR FMCLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR OCTOSPI1LPEN LL_AHB4_GRP1_DisableClockSleep

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM3EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM4EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM5EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM6EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM7EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM12EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM13EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM14EN LL_APB1_GRP1_EnableClock
- APB1LENR WWDGEN LL_APB1_GRP1_EnableClock
- APB1LENR SPI2EN LL_APB1_GRP1_EnableClock
- APB1LENR SPI3EN LL_APB1_GRP1_EnableClock
- APB1LENR USART2EN LL_APB1_GRP1_EnableClock
- APB1LENR USART3EN LL_APB1_GRP1_EnableClock
- APB1LENR UART4EN LL_APB1_GRP1_EnableClock
- APB1LENR UART5EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C1EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C2EN LL_APB1_GRP1_EnableClock
- APB1LENR I3C1EN LL_APB1_GRP1_EnableClock
- APB1LENR CRSEN LL_APB1_GRP1_EnableClock
- APB1LENR USART6EN LL_APB1_GRP1_EnableClock
- APB1LENR USART10EN LL_APB1_GRP1_EnableClock
- APB1LENR USART11EN LL_APB1_GRP1_EnableClock
- APB1LENR CECEN LL_APB1_GRP1_EnableClock
- APB1LENR UART7EN LL_APB1_GRP1_EnableClock
- APB1LENR UART8EN LL_APB1_GRP1_EnableClock

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR UART9EN LL_APB1_GRP2_EnableClock
- APB1HENR UART12EN LL_APB1_GRP2_EnableClock
- APB1HENR DTSEN LL_APB1_GRP2_EnableClock
- APB1HENR LPTIM2EN LL_APB1_GRP2_EnableClock
- APB1HENR FDCANEN LL_APB1_GRP2_EnableClock
- APB1HENR FDCANEN LL_APB1_GRP2_EnableClock
- APB1HENR UCPD1EN LL_APB1_GRP2_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR WWDGEN LL_APB1_GRP1_IsEnabledClock
- APB1LENR SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR SPI3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I3C1EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR CRSEN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART6EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART10EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART11EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR CECEN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART7EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART8EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1HENR UART9EN LL_APB1_GRP2_IsEnabledClock
- APB1HENR UART12EN LL_APB1_GRP2_IsEnabledClock
- APB1HENR DTSEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR LPTIM2EN LL_APB1_GRP2_IsEnabledClock
- APB1HENR FDCANEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR UCPD1EN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM3EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM4EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM5EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM6EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM7EN LL_APB1_GRP1_DisableClock
- APB1LENR WWDGEN LL_APB1_GRP1_DisableClock
- APB1LENR SPI2EN LL_APB1_GRP1_DisableClock
- APB1LENR SPI3EN LL_APB1_GRP1_DisableClock
- APB1LENR USART2EN LL_APB1_GRP1_DisableClock
- APB1LENR USART3EN LL_APB1_GRP1_DisableClock
- APB1LENR UART4EN LL_APB1_GRP1_DisableClock
- APB1LENR UART5EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C1EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C2EN LL_APB1_GRP1_DisableClock
- APB1LENR I3C1EN LL_APB1_GRP1_DisableClock
- APB1LENR CRSEN LL_APB1_GRP1_DisableClock
- APB1LENR USART6EN LL_APB1_GRP1_DisableClock
- APB1LENR USART10EN LL_APB1_GRP1_DisableClock
- APB1LENR USART11EN LL_APB1_GRP1_DisableClock
- APB1LENR CECEN LL_APB1_GRP1_DisableClock
- APB1LENR UART7EN LL_APB1_GRP1_DisableClock
- APB1LENR UART8EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR UART9EN LL_APB1_GRP2_DisableClock
- APB1HENR UART12EN LL_APB1_GRP2_DisableClock
- APB1HENR DTSEN LL_APB1_GRP2_DisableClock
- APB1HENR LPTIM2EN LL_APB1_GRP2_DisableClock
- APB1HENR FDCANEN LL_APB1_GRP2_DisableClock
- APB1HENR UCPD1EN LL_APB1_GRP2_DisableClock

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LRSTR TIM2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM4RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM5RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM6RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM7RST LL_APB1_GRP1_ForceReset
- APB1LRSTR SPI2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR SPI3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART4RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART5RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C1RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I3C1RST LL_APB1_GRP1_ForceReset
- APB1LRSTR CRSRST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART6RST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART10RST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART11RST LL_APB1_GRP1_ForceReset
- APB1LRSTR CECRST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART7RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART8RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HRSTR UART9RST LL_APB1_GRP2_ForceReset
- APB1HRSTR UART12RST LL_APB1_GRP2_ForceReset
- APB1HRSTR DTSRST LL_APB1_GRP2_ForceReset
- APB1HRSTR LPTIM2RST LL_APB1_GRP2_ForceReset
- APB1HRSTR FDCANRST LL_APB1_GRP2_ForceReset
- APB1HRSTR UCPD1RST LL_APB1_GRP2_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LRSTR TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM4RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM5RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR SPI3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART4RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART5RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I3C1RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR CRSRST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART6RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART10RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART11RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR CECRST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART7RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART8RST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HRSTR UART9RST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR UART12RST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR DTSRST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR LPTIM2RST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR FDCAN LL_APB1_GRP2_ReleaseReset
- APB1HRSTR UCPD1RST LL_APB1_GRP2_ReleaseReset

LL_APB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM4LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM5LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM6LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM7LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR WWDGLPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART4LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART5LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C1LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I3C1LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR CRSLPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART6LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART10LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART11LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR CECLPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART7LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART8LPEN LL_APB1_GRP1_EnableClockSleep

LL_APB1_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if APB1 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR TIM3LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR TIM4LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR TIM5LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR TIM6LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR TIM7LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR WWDGLPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR SPI2LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR SPI3LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR USART2LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR USART3LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR UART4LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR UART5LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR I2C1LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR I2C2LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR I3C1LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR CRSLPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR USART6LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR USART10LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR USART11LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR CECLPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR UART7LPEN LL_APB1_GRP1_IsEnabledClockSleep
- APB1LLPENR UART8LPEN LL_APB1_GRP1_IsEnabledClockSleep

LL_APB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4 (*)
  - LL_APB1_GRP1_PERIPH_TIM5 (*)
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12 (*)
  - LL_APB1_GRP1_PERIPH_TIM13 (*)
  - LL_APB1_GRP1_PERIPH_TIM14 (*)
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_OPAMP (*)
  - LL_APB1_GRP1_PERIPH_COMP (*)
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4 (*)
  - LL_APB1_GRP1_PERIPH_UART5 (*)
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I3C1
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)
  - LL_APB1_GRP1_PERIPH_USART10 (*)
  - LL_APB1_GRP1_PERIPH_USART11 (*)
  - LL_APB1_GRP1_PERIPH_CEC (*)
  - LL_APB1_GRP1_PERIPH_UART7 (*)
  - LL_APB1_GRP1_PERIPH_UART8 (*)

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM4LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM5LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM6LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM7LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR WWDGLPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART4LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART5LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C1LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I3C1LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR CRSLPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART6LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART10LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART11LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR CECLPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART7LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART8LPEN LL_APB1_GRP1_DisableClockSleep

LL_APB1_GRP2_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR UART9LPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR UART12LPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR DTSLPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR LPTIM2LPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR FDCAN12LPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR FDCAN1LPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR UCPD1LPEN LL_APB1_GRP2_EnableClockSleep

LL_APB1_GRP2_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if APB1 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1HLPENR UART9LPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR UART12LPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR DTSLPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR LPTIM2LPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR FDCAN12LPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR FDCAN1LPEN LL_APB1_GRP2_IsEnabledClockSleep
- APB1HLPENR UCPD1LPEN LL_APB1_GRP2_IsEnabledClockSleep

LL_APB1_GRP2_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_UART9 (*)
  - LL_APB1_GRP2_PERIPH_UART12 (*)
  - LL_APB1_GRP2_PERIPH_DTS
  - LL_APB1_GRP2_PERIPH_LPTIM2
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_UCPD1

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR UART9LPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR UART12LPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR DTSLPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR LPTIM2LPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR FDCAN12LPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR FDCAN1LPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR UCPD1LPEN LL_APB1_GRP2_DisableClockSleep

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI4EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI6EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_APB2_GRP1_EnableClock
- APB2ENR USBEN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI6EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USBEN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI4EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI6EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI2EN LL_APB2_GRP1_DisableClock
- APB2ENR USBEN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI6RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ForceReset
- APB2RSTR USBRST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1 (*)
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI6RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USBRST LL_APB2_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB2 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM8LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR USART1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM15LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM16LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM17LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI4LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI6LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI2LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR USBLPEN LL_APB2_GRP1_EnableClockSleep

LL_APB2_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if APB2 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR SPI1LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR TIM8LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR USART1LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR TIM15LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR TIM16LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR TIM17LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR SPI4LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR SPI6LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR SAI1LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR SAI2LPEN LL_APB2_GRP1_IsEnabledClockSleep
- APB2LPENR USBLPEN LL_APB2_GRP1_IsEnabledClockSleep

LL_APB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB2 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8 (*)
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15 (*)
  - LL_APB2_GRP1_PERIPH_TIM16 (*)
  - LL_APB2_GRP1_PERIPH_TIM17 (*)
  - LL_APB2_GRP1_PERIPH_SPI4 (*)
  - LL_APB2_GRP1_PERIPH_SPI6 (*)
  - LL_APB2_GRP1_PERIPH_SAI1 (*)
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM8LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR USART1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM15LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM16LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM17LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI4LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI6LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI2LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR USBLPEN LL_APB2_GRP1_DisableClockSleep

LL_APB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR SBSEN LL_APB3_GRP1_EnableClock
- APB3ENR SPI5EN LL_APB3_GRP1_EnableClock
- APB3ENR LPUART1EN LL_APB3_GRP1_EnableClock
- APB3ENR I2C3EN LL_APB3_GRP1_EnableClock
- APB3ENR I2C4EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM5EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM6EN LL_APB3_GRP1_EnableClock
- APB3ENR VREFEN LL_APB3_GRP1_EnableClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_EnableClock

LL_APB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB3ENR SBSEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR SPI5EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPUART1EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR I2C3EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR I2C4EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM5EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM6EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR VREFEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_IsEnabledClock

LL_APB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR SBSEN LL_APB3_GRP1_DisableClock
- APB3ENR SPI5EN LL_APB3_GRP1_DisableClock
- APB3ENR LPUART1EN LL_APB3_GRP1_DisableClock
- APB3ENR I2C3EN LL_APB3_GRP1_DisableClock
- APB3ENR I2C4EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM5EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM6EN LL_APB3_GRP1_DisableClock
- APB3ENR VREFEN LL_APB3_GRP1_DisableClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_DisableClock

LL_APB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR SPI5RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPUART1RST LL_APB3_GRP1_ForceReset
- APB3RSTR I2C3RST LL_APB3_GRP1_ForceReset
- APB3RSTR I2C4RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM1RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM3RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM4RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM5RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM6RST LL_APB3_GRP1_ForceReset
- APB3RSTR VREFRST LL_APB3_GRP1_ForceReset

LL_APB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR SPI5RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPUART1RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR I2C3RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR I2C4RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM1RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM3RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM4RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM5RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM6RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR VREFRST LL_APB3_GRP1_ReleaseReset

LL_APB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB3 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR SBSLPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR SPI5LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPUART1LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR I2C3LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR I2C4LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPTIM1LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPTIM3LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPTIM4LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPTIM5LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR LPTIM6LPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR VREFLPEN LL_APB3_GRP1_EnableClockSleep
- APB3LPENR RTCAPBLPEN LL_APB3_GRP1_EnableClockSleep

LL_APB3_GRP1_IsEnabledClockSleep

Function name

__STATIC_INLINE uint32_t LL_APB3_GRP1_IsEnabledClockSleep (uint32_t Periphs)

Function description

Check if APB3 peripheral clocks in Sleep mode is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB3LPENR SBSLPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR SPI5LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPUART1LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR I2C3LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR I2C4LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPTIM1LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPTIM3LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPTIM4LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPTIM5LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR LPTIM6LPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR VREFLPEN LL_APB3_GRP1_IsEnabledClockSleep
- APB3LPENR RTCAPBLPEN LL_APB3_GRP1_IsEnabledClockSleep

LL_APB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB3 peripheral clocks in Sleep mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SBS
  - LL_APB3_GRP1_PERIPH_SPI5 (*)
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3 (*)
  - LL_APB3_GRP1_PERIPH_I2C4 (*)
  - LL_APB3_GRP1_PERIPH_I3C2 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM3 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB3_GRP1_PERIPH_LPTIM6 (*)
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

  (*) : Not available for all stm32h5xxxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR SBSLPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR SPI5LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPUART1LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR I2C3LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR I2C4LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPTIM1LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPTIM3LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPTIM4LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPTIM5LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR LPTIM6LPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR VREFLPEN LL_APB3_GRP1_DisableClockSleep
- APB3LPENR RTCAPBLPEN LL_APB3_GRP1_DisableClockSleep

86.2 BUS Firmware driver defines

The following section lists the various define and macros of the module.

86.2.1 BUS

BUS

BRANCH CLK AHBx

LL_AHB_BRANCH_CLK_AHB1

LL_AHB_BRANCH_CLK_AHB2

LL_AHB_BRANCH_CLK_AHB4

BRANCH CLK APBx

LL_APB_BRANCH_CLK_APB1

LL_APB_BRANCH_CLK_APB2

LL_APB_BRANCH_CLK_APB3

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_GPDMA1

LL_AHB1_GRP1_PERIPH_GPDMA2

LL_AHB1_GRP1_PERIPH_CORDIC

LL_AHB1_GRP1_PERIPH_FMAC

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_ETH

LL_AHB1_GRP1_PERIPH_ETHTX

LL_AHB1_GRP1_PERIPH_ETHRX

LL_AHB1_GRP1_PERIPH_CRC

LL_AHB1_GRP1_PERIPH_RAMCFG

LL_AHB1_GRP1_PERIPH_GTZC1

LL_AHB1_GRP1_PERIPH_BKPSRAM

LL_AHB1_GRP1_PERIPH_ICACHE

LL_AHB1_GRP1_PERIPH_DCACHE1

LL_AHB1_GRP1_PERIPH_SRAM1

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_ALL

LL_AHB2_GRP1_PERIPH_GPIOA

LL_AHB2_GRP1_PERIPH_GPIOB

LL_AHB2_GRP1_PERIPH_GPIOC

LL_AHB2_GRP1_PERIPH_GPIOD

LL_AHB2_GRP1_PERIPH_GPIOE

LL_AHB2_GRP1_PERIPH_GPIOF

LL_AHB2_GRP1_PERIPH_GPIOG

LL_AHB2_GRP1_PERIPH_GPIOH

LL_AHB2_GRP1_PERIPH_GPIOI

LL_AHB2_GRP1_PERIPH_ADC

LL_AHB2_GRP1_PERIPH_DAC1

LL_AHB2_GRP1_PERIPH_DCMI_PSSI

LL_AHB2_GRP1_PERIPH_AES

LL_AHB2_GRP1_PERIPH_HASH

LL_AHB2_GRP1_PERIPH_RNG

LL_AHB2_GRP1_PERIPH_PKA

LL_AHB2_GRP1_PERIPH_SAES

LL_AHB2_GRP1_PERIPH_SRAM2

LL_AHB2_GRP1_PERIPH_SRAM3

AHB4 GRP1 PERIPH

LL_AHB4_GRP1_PERIPH_ALL

LL_AHB4_GRP1_PERIPH_OTFDEC

LL_AHB4_GRP1_PERIPH_SDMMC1

LL_AHB4_GRP1_PERIPH_SDMMC2

LL_AHB4_GRP1_PERIPH_FMC

LL_AHB4_GRP1_PERIPH_OSPI1

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_TIM12

LL_APB1_GRP1_PERIPH_TIM13

LL_APB1_GRP1_PERIPH_TIM14

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_I3C1

LL_APB1_GRP1_PERIPH_CRS

LL_APB1_GRP1_PERIPH_USART6

LL_APB1_GRP1_PERIPH_USART10

LL_APB1_GRP1_PERIPH_USART11

LL_APB1_GRP1_PERIPH_CEC

LL_APB1_GRP1_PERIPH_UART7

LL_APB1_GRP1_PERIPH_UART8

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_ALL

LL_APB1_GRP2_PERIPH_UART9

LL_APB1_GRP2_PERIPH_UART12

LL_APB1_GRP2_PERIPH_DTS

LL_APB1_GRP2_PERIPH_LPTIM2

LL_APB1_GRP2_PERIPH_FDCAN

LL_APB1_GRP2_PERIPH_UCPD1

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_SPI4

LL_APB2_GRP1_PERIPH_SPI6

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_SAI2

LL_APB2_GRP1_PERIPH_USB

APB3 GRP1 PERIPH

LL_APB3_GRP1_PERIPH_ALL

LL_APB3_GRP1_PERIPH_SBS

LL_APB3_GRP1_PERIPH_SPI5

LL_APB3_GRP1_PERIPH_LPUART1

LL_APB3_GRP1_PERIPH_I2C3

LL_APB3_GRP1_PERIPH_I2C4

LL_APB3_GRP1_PERIPH_LPTIM1

LL_APB3_GRP1_PERIPH_LPTIM3

LL_APB3_GRP1_PERIPH_LPTIM4

LL_APB3_GRP1_PERIPH_LPTIM5

LL_APB3_GRP1_PERIPH_LPTIM6

LL_APB3_GRP1_PERIPH_VREF

LL_APB3_GRP1_PERIPH_RTCAPB

87 LL COMP Generic Driver

87.1 COMP Firmware driver registers structures

87.1.1 LL_COMP_InitTypeDef

LL_COMP_InitTypeDef is defined in the stm32h5xx_ll_comp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t InputHysteresis***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputBlankingSource***

Field Documentation

- ***uint32_t LL_COMP_InitTypeDef::PowerMode***
  Set comparator operating mode to adjust power and speed. This parameter can be a value of ***COMP_LL_EC_POWERMODE*** This feature can be modified afterwards using unitary function **LL_COMP_SetPowerMode()**.
- ***uint32_t LL_COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_PLUS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputPlus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_MINUS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputMinus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputHysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_LL_EC_INPUT_HYSTERESIS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputHysteresis()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputPolarity***
  Set comparator output polarity. This parameter can be a value of ***COMP_LL_EC_OUTPUT_POLARITY*** This feature can be modified afterwards using unitary function **LL_COMP_SetOutputPolarity()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputBlankingSource***
  Set comparator blanking source. This parameter can be a value of ***COMP_LL_EC_OUTPUT_BLANKING_SOURCE*** This feature can be modified afterwards using unitary function **LL_COMP_SetOutputBlankingSource()**.

87.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

87.2.1 Detailed description of functions

LL_COMP_SetPowerMode

Function name

__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)

Function description

Set comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 PWRMODE LL_COMP_SetPowerMode

LL_COMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (const COMP_TypeDef * COMPx)

Function description

Get comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Reference Manual to LL API cross reference:

- CFGR1 PWRMODE LL_COMP_GetPowerMode

LL_COMP_ConfigInputs

Function name

__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)

Function description

Set comparator inputs minus (inverting) and plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_DAC1_CH2
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_IO3
  - LL_COMP_INPUT_MINUS_TEMPSENSOR
  - LL_COMP_INPUT_MINUS_VBAT
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_IO3
  - LL_COMP_INPUT_PLUS_DAC1_CH1

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CFGR1 INMSEL LL_COMP_ConfigInputs
- CFGR1 INPSEL LL_COMP_ConfigInputs
- CFGR1 BRGEN LL_COMP_ConfigInputs
- CFGR1 SCALEN LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name

__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)

Function description

Set comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_IO3
  - LL_COMP_INPUT_PLUS_DAC1_CH1

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGR1 INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (const COMP_TypeDef * COMPx)

Function description

Get comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_IO3
  - LL_COMP_INPUT_PLUS_DAC1_CH1

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGR1 INPSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name

__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)

Function description

Set comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_IO3
  - LL_COMP_INPUT_MINUS_TEMPSENSOR
  - LL_COMP_INPUT_MINUS_VBAT

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CFGR1 INMSEL LL_COMP_SetInputMinus
- CFGR1 BRGEN LL_COMP_SetInputMinus
- CFGR1 SCALEN LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputMinus (const COMP_TypeDef * COMPx)

Function description

Get comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_IO3
  - LL_COMP_INPUT_MINUS_TEMPSENSOR
  - LL_COMP_INPUT_MINUS_VBAT

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGR1 INMSEL LL_COMP_GetInputMinus
- CFGR1 BRGEN LL_COMP_GetInputMinus
- CFGR1 SCALEN LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

Function name

__STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)

Function description

Set comparator instance hysteresis mode of the input minus (inverting input).

Parameters

- **COMPx:** Comparator instance
- **InputHysteresis:** This parameter can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (const COMP_TypeDef * COMPx)

Function description

Get comparator instance hysteresis mode of the minus (inverting) input.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_GetInputHysteresis

LL_COMP_SetOutputPolarity

Function name

__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)

Function description

Set comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance
- **OutputPolarity:** This parameter can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (const COMP_TypeDef * COMPx)

Function description

Get comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Reference Manual to LL API cross reference:

- CFGR1 POLARITY LL_COMP_GetOutputPolarity

LL_COMP_SetOutputBlankingSource

Function name

__STATIC_INLINE void LL_COMP_SetOutputBlankingSource (COMP_TypeDef * COMPx, uint32_t BlankingSource)

Function description

Set comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance
- **BlankingSource:** This parameter can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5
  - LL_COMP_BLANKINGSRC_TIM2_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC4
  - LL_COMP_BLANKINGSRC_LPTIM1_OC2
  - LL_COMP_BLANKINGSRC_LPTIM2_OC2

Return values

- **None:**

Notes

- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.
- Availability of parameters of blanking source from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR BLANKING LL_COMP_SetOutputBlankingSource

LL_COMP_GetOutputBlankingSource

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputBlankingSource (const COMP_TypeDef * COMPx)

Function description

Get comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5
  - LL_COMP_BLANKINGSRC_TIM2_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC4
  - LL_COMP_BLANKINGSRC_LPTIM1_OC2
  - LL_COMP_BLANKINGSRC_LPTIM2_OC2

Notes

- Availability of parameters of blanking source from timer depends on timers availability on the selected device.
- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGR BLANKING LL_COMP_GetOutputBlankingSource

LL_COMP_Enable

Function name

__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)

Function description

Enable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".

Reference Manual to LL API cross reference:

- CFGR1 EN LL_COMP_Enable

LL_COMP_Disable

Function name

__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)

Function description

Disable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_COMP_IsEnabled (const COMP_TypeDef * COMPx)

Function description

Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR1 EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name

__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)

Function description

Lock comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CFGR1 LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name

__STATIC_INLINE uint32_t LL_COMP_IsLocked (const COMP_TypeDef * COMPx)

Function description

Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CFGR1 LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name

__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (const COMP_TypeDef * COMPx)

Function description

Read comparator instance output level.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUT_LEVEL_LOW
  - LL_COMP_OUTPUT_LEVEL_HIGH

Notes

- The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus

Reference Manual to LL API cross reference:

- CFGR VALUE LL_COMP_ReadOutputLevel

LL_COMP_IsActiveFlag_OutputTrig

Function name

__STATIC_INLINE uint32_t LL_COMP_IsActiveFlag_OutputTrig (const COMP_TypeDef * COMPx)

Function description

Get comparator output trigger flag (latched)

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR C1IF LL_COMP_IsActiveFlag_OutputTrig

LL_COMP_ClearFlag_OutputTrig

Function name

__STATIC_INLINE void LL_COMP_ClearFlag_OutputTrig (COMP_TypeDef * COMPx)

Function description

Clear comparator comparator output trigger flag (latched)

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICFR CC1IF LL_COMP_ClearFlag_OutputTrig

LL_COMP_EnableIT_OutputTrig

Function name

__STATIC_INLINE void LL_COMP_EnableIT_OutputTrig (COMP_TypeDef * COMPx)

Function description

Enable comparator output trigger interrupt.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICFR ITEN LL_COMP_EnableIT_OutputTrig

LL_COMP_DisableIT_OutputTrig

Function name

__STATIC_INLINE void LL_COMP_DisableIT_OutputTrig (COMP_TypeDef * COMPx)

Function description

Disable comparator output trigger interrupt.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICFR ITEN LL_COMP_DisableIT_OutputTrig

LL_COMP_IsEnabledIT_OutputTrig

Function name

__STATIC_INLINE uint32_t LL_COMP_IsEnabledIT_OutputTrig (const COMP_TypeDef * COMPx)

Function description

Get comparator output trigger interrupt state.

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ICFR ITEN LL_COMP_IsEnabledIT_OutputTrig

LL_COMP_DeInit

Function name

ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)

Function description

De-initialize registers of the selected COMP instance to their default reset values.

Parameters

- **COMPx:** COMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are de-initialized
  - ERROR: COMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name

ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, const LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Initialize some features of COMP instance.

Parameters

- **COMPx:** COMP instance
- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are initialized
  - ERROR: COMP registers are not initialized

Notes

- This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name

void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Set each LL_COMP_InitTypeDef field to default value.

Parameters

- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

87.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

87.3.1 COMP

COMP

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US

Delay for COMP startup time

LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US

Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_NONE

No hysteresis

LL_COMP_HYSTERESIS_LOW

Hysteresis level low

LL_COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

LL_COMP_HYSTERESIS_HIGH

Hysteresis level high

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

LL_COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

LL_COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

LL_COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

LL_COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

LL_COMP_INPUT_MINUS_IO1

Comparator input minus connected to pin PC4

LL_COMP_INPUT_MINUS_IO2

Comparator input minus connected to pin PB1

LL_COMP_INPUT_MINUS_IO3

Comparator input minus connected to pin PA5

LL_COMP_INPUT_MINUS_TEMPSENSOR

Comparator input minus connected to internal temperature sensor (also accessible through ADC peripheral)

LL_COMP_INPUT_MINUS_VBAT

Comparator input minus connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have input voltage always below Vdda.

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PB0)

LL_COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PA0)

LL_COMP_INPUT_PLUS_IO3

Comparator input plus connected to IO3 (pin PB2)

LL_COMP_INPUT_PLUS_DAC1_CH1

Comparator input plus connected to DAC1 channel 1

Comparator output - Blanking source

LL_COMP_BLANKINGSRC_NONE

Comparator output without blanking

LL_COMP_BLANKINGSRC_TIM1_OC5

Comparator output blanking source TIM1 OC5

LL_COMP_BLANKINGSRC_TIM2_OC3

Comparator output blanking source TIM2 OC3

LL_COMP_BLANKINGSRC_TIM3_OC3

Comparator output blanking source TIM3 OC3

LL_COMP_BLANKINGSRC_TIM3_OC4

Comparator output blanking source TIM3 OC4

LL_COMP_BLANKINGSRC_LPTIM1_OC2

Comparator output blanking source LPTIM1 OC2

LL_COMP_BLANKINGSRC_LPTIM2_OC2

Comparator output blanking source LPTIM2 OC2

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW

Comparator output level low (if polarity not inverted, otherwise to be complemented)

LL_COMP_OUTPUT_LEVEL_HIGH

Comparator output level high (if polarity not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED

COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input

LL_COMP_OUTPUTPOL_INVERTED

COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator modes - Power mode

LL_COMP_POWERMODE_HIGHSPEED

COMP power mode to high speed

LL_COMP_POWERMODE_MEDIUMSPEED

COMP power mode to medium speed

LL_COMP_POWERMODE_ULTRALOWPOWER

COMP power mode to ultra-low power

Common write and read registers macro

LL_COMP_WriteReg

Description:

- Write a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg

Description:

- Read a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be read

Return value:

- Register: value

88 LL CORDIC Generic Driver

88.1 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

88.1.1 Detailed description of functions

LL_CORDIC_Config

Function name

__STATIC_INLINE void LL_CORDIC_Config (CORDIC_TypeDef * CORDICx, uint32_t Function, uint32_t Precision, uint32_t Scale, uint32_t NbWrite, uint32_t NbRead, uint32_t InSize, uint32_t OutSize)

Function description

Configure the CORDIC processing.

Parameters

- **CORDICx:** CORDIC instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS
- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Notes

- This function set all parameters of CORDIC processing. These parameters can also be set individually using dedicated functions: LL_CORDIC_SetFunction()LL_CORDIC_SetPrecision()LL_CORDIC_SetScale()LL_CORDIC_SetNbWrite()LL_CORDIC_SetNbRead()LL_CORDIC_SetInSize()LL_CORDIC_SetOutSize()

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_Config
- CSR PRECISION LL_CORDIC_Config
- CSR SCALE LL_CORDIC_Config
- CSR NARGS LL_CORDIC_Config
- CSR NRES LL_CORDIC_Config
- CSR ARGSIZE LL_CORDIC_Config
- CSR RESIZE LL_CORDIC_Config

LL_CORDIC_SetFunction

Function name

__STATIC_INLINE void LL_CORDIC_SetFunction (CORDIC_TypeDef * CORDICx, uint32_t Function)

Function description

Configure function.

Parameters

- **CORDICx:** CORDIC Instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_SetFunction

LL_CORDIC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetFunction (const CORDIC_TypeDef * CORDICx)

Function description

Return function.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_GetFunction

LL_CORDIC_SetPrecision

Function name

__STATIC_INLINE void LL_CORDIC_SetPrecision (CORDIC_TypeDef * CORDICx, uint32_t Precision)

Function description

Configure precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_SetPrecision

LL_CORDIC_GetPrecision

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetPrecision (const CORDIC_TypeDef * CORDICx)

Function description

Return precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_GetPrecision

LL_CORDIC_SetScale

Function name

__STATIC_INLINE void LL_CORDIC_SetScale (CORDIC_TypeDef * CORDICx, uint32_t Scale)

Function description

Configure scaling factor.

Parameters

- **CORDICx:** CORDIC Instance
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_SetScale

LL_CORDIC_GetScale

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetScale (const CORDIC_TypeDef * CORDICx)

Function description

Return scaling factor.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_GetScale

LL_CORDIC_SetNbWrite

Function name

__STATIC_INLINE void LL_CORDIC_SetNbWrite (CORDIC_TypeDef * CORDICx, uint32_t NbWrite)

Function description

Configure number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_SetNbWrite

LL_CORDIC_GetNbWrite

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbWrite (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_GetNbWrite

LL_CORDIC_SetNbRead

Function name

__STATIC_INLINE void LL_CORDIC_SetNbRead (CORDIC_TypeDef * CORDICx, uint32_t NbRead)

Function description

Configure number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_SetNbRead

LL_CORDIC_GetNbRead

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbRead (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_GetNbRead

LL_CORDIC_SetInSize

Function name

__STATIC_INLINE void LL_CORDIC_SetInSize (CORDIC_TypeDef * CORDICx, uint32_t InSize)

Function description

Configure width of input data.

Parameters

- **CORDICx:** CORDIC Instance
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_SetInSize

LL_CORDIC_GetInSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetInSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of input data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_GetInSize

LL_CORDIC_SetOutSize

Function name

__STATIC_INLINE void LL_CORDIC_SetOutSize (CORDIC_TypeDef * CORDICx, uint32_t OutSize)

Function description

Configure width of output data.

Parameters

- **CORDICx:** CORDIC Instance
- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_SetOutSize

LL_CORDIC_GetOutSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetOutSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of output data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_GetOutSize

LL_CORDIC_EnableIT

Function name

__STATIC_INLINE void LL_CORDIC_EnableIT (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_EnableIT

LL_CORDIC_DisableIT

Function name

__STATIC_INLINE void LL_CORDIC_DisableIT (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_DisableIT

LL_CORDIC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledIT (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready interrupt state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_IsEnabledIT

LL_CORDIC_EnableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_EnableDMAReq_RD

LL_CORDIC_DisableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_DisableDMAReq_RD

LL_CORDIC_IsEnabledDMAReq_RD

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_RD (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA read channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_IsEnabledDMAReq_RD

LL_CORDIC_EnableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_EnableDMAReq_WR

LL_CORDIC_DisableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_DisableDMAReq_WR

LL_CORDIC_IsEnabledDMAReq_WR

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_WR (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA write channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_IsEnabledDMAReq_WR

LL_CORDIC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_CORDIC_DMA_GetRegAddr (const CORDIC_TypeDef * CORDICx, uint32_t Direction)

Function description

Get the CORDIC data register address used for DMA transfer.

Parameters

- **CORDICx:** CORDIC Instance
- **Direction:** parameter can be one of the following values:
  - LL_CORDIC_DMA_REG_DATA_IN
  - LL_CORDIC_DMA_REG_DATA_OUT

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_DMA_GetRegAddr
-
- WDATA ARG LL_CORDIC_DMA_GetRegAddr

LL_CORDIC_IsActiveFlag_RRDY

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsActiveFlag_RRDY (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready flag state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR RRDY LL_CORDIC_IsActiveFlag_RRDY

LL_CORDIC_WriteData

Function name

__STATIC_INLINE void LL_CORDIC_WriteData (CORDIC_TypeDef * CORDICx, uint32_t InData)

Function description

Write 32-bit input data for the CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance
- **InData:** 0 .. 0xFFFFFFFF : 32-bit value to be provided as input data for CORDIC processing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA ARG LL_CORDIC_WriteData

LL_CORDIC_ReadData

Function name

__STATIC_INLINE uint32_t LL_CORDIC_ReadData (const CORDIC_TypeDef * CORDICx)

Function description

Return 32-bit output data of CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **32-bit:** output data of CORDIC processing.

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_ReadData

LL_CORDIC_DeInit

Function name

ErrorStatus LL_CORDIC_DeInit (const CORDIC_TypeDef * CORDICx)

Function description

De-Initialize CORDIC peripheral registers to their default reset values.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CORDIC registers are de-initialized
  - ERROR: CORDIC registers are not de-initialized

88.2 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

88.2.1 CORDIC

CORDIC

DMA register data

LL_CORDIC_DMA_REG_DATA_IN

Get address of input data register

LL_CORDIC_DMA_REG_DATA_OUT

Get address of output data register

FUNCTION

LL_CORDIC_FUNCTION_COSINE

Cosine

LL_CORDIC_FUNCTION_SINE

Sine

LL_CORDIC_FUNCTION_PHASE

Phase

LL_CORDIC_FUNCTION_MODULUS

Modulus

LL_CORDIC_FUNCTION_ARCTANGENT

Arctangent

LL_CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

LL_CORDIC_FUNCTION_HSINE

Hyperbolic Sine

LL_CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

LL_CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

LL_CORDIC_FUNCTION_SQUAREROOT

Square Root

Get Flags Defines

LL_CORDIC_FLAG_RRDY

INSIZE

LL_CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

LL_CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

IT Defines

LL_CORDIC_IT_IEN

Result Ready interrupt enable

NBREAD

LL_CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

NBWRITE

LL_CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

OUTSIZE

LL_CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

LL_CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

PRECISION

LL_CORDIC_PRECISION_1CYCLE

LL_CORDIC_PRECISION_2CYCLES

LL_CORDIC_PRECISION_3CYCLES

LL_CORDIC_PRECISION_4CYCLES

LL_CORDIC_PRECISION_5CYCLES

LL_CORDIC_PRECISION_6CYCLES

LL_CORDIC_PRECISION_7CYCLES

LL_CORDIC_PRECISION_8CYCLES

LL_CORDIC_PRECISION_9CYCLES

LL_CORDIC_PRECISION_10CYCLES

LL_CORDIC_PRECISION_11CYCLES

LL_CORDIC_PRECISION_12CYCLES

LL_CORDIC_PRECISION_13CYCLES

LL_CORDIC_PRECISION_14CYCLES

LL_CORDIC_PRECISION_15CYCLES

SCALE

LL_CORDIC_SCALE_0

LL_CORDIC_SCALE_1

LL_CORDIC_SCALE_2

LL_CORDIC_SCALE_3

LL_CORDIC_SCALE_4

LL_CORDIC_SCALE_5

LL_CORDIC_SCALE_6

LL_CORDIC_SCALE_7

Common Write and read registers Macros

LL_CORDIC_WriteReg

Description:

- Write a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CORDIC_ReadReg

Description:

- Read a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be read

Return value:

- Register: value

89 LL CORTEX Generic Driver

89.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

89.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)

Function description

This function checks if the Systick counter flag is active or not.

Return values

- **State:** of bit (1 or 0).

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

- STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name

__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)

Function description

Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)

Function description

Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name

__STATIC_INLINE void LL_SYSTICK_EnableIT (void)

Function description

Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name

__STATIC_INLINE void LL_SYSTICK_DisableIT (void)

Function description

Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)

Function description

Checks if the SYSTICK interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name

__STATIC_INLINE void LL_LPM_EnableSleep (void)

Function description

Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name

__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)

Function description

Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)

Function description

Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)

Function description

Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)

Function description

Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)

Function description

Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name

__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)

Function description

Enable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM
  - LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name

__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)

Function description

Disable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM
  - LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)

Function description

Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)

Function description

Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)

Function description

Get Architecture version.

Return values

- **Value:** should be equal to 0xF for Cortex-M33 ("ARMv8-M with Main Extension")

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)

Function description

Get Part number.

Return values

- **Value:** should be equal to 0xD21 for Cortex-M33

Reference Manual to LL API cross reference:

- SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)

Function description

Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)

Return values

- **Value:** between 0 and 255 (0x1: patch 1)

Reference Manual to LL API cross reference:

- SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name

__STATIC_INLINE void LL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable MPU with input options.

Parameters

- **MPU_Control:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Enable_NS

Function name

__STATIC_INLINE void LL_MPU_Enable_NS (uint32_t MPU_Control)

Function description

Enable non-secure MPU with input options.

Parameters

- **MPU_Control:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name

__STATIC_INLINE void LL_MPU_Disable (void)

Function description

Disable MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_Disable_NS

Function name

__STATIC_INLINE void LL_MPU_Disable_NS (void)

Function description

Disable the non-secure MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable_NS

LL_MPU_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)

Function description

Check if MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_IsEnabled_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_NS (void)

Function description

Check if non-secure MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled_NS

LL_MPU_EnableRegion

Function name

__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)

Function description

Enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RLAR ENABLE LL_MPU_EnableRegion

LL_MPU_IsEnabled_Region

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_Region (uint32_t Region)

Function description

Check if MPU region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR ENABLE LL_MPU_IsEnabled_Region

LL_MPU_EnableRegion_NS

Function name

__STATIC_INLINE void LL_MPU_EnableRegion_NS (uint32_t Region)

Function description

Enable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RLAR ENABLE LL_MPU_EnableRegion_NS

LL_MPU_IsEnabled_Region_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_Region_NS (uint32_t Region)

Function description

Check if non-secure MPU region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **State:** of bit (1 or 0).

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR ENABLE LL_MPU_IsEnabled_Region_NS

LL_MPU_DisableRegion

Function name

__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)

Function description

Disable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RLAR ENABLE LL_MPU_DisableRegion

LL_MPU_DisableRegion_NS

Function name

__STATIC_INLINE void LL_MPU_DisableRegion_NS (uint32_t Region)

Function description

Disable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion_NS
- MPU_RLAR ENABLE LL_MPU_DisableRegion_NS
-

LL_MPU_ConfigRegion

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t Attributes, uint32_t AttrIndx, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure and enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO
- **AttrIndx:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR ADDR LL_MPU_ConfigRegion
- MPU_RLAR ADDR LL_MPU_ConfigRegion
- MPU_RBAR XN LL_MPU_ConfigRegion
- MPU_RBAR AP LL_MPU_ConfigRegion
- MPU_RBAR SH LL_MPU_ConfigRegion
- MPU_RLAR EN LL_MPU_ConfigRegion
- MPU_RLAR AttrIndx LL_MPU_ConfigRegion
-

LL_MPU_ConfigRegion_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion_NS (uint32_t Region, uint32_t Attributes, uint32_t AttrIndx, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure and enable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO
- **AttrIndx:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion_NS
- MPU_RBAR ADDR LL_MPU_ConfigRegion_NS
- MPU_RLAR ADDR LL_MPU_ConfigRegion_NS
- MPU_RBAR XN LL_MPU_ConfigRegion_NS
- MPU_RBAR AP LL_MPU_ConfigRegion_NS
- MPU_RBAR SH LL_MPU_ConfigRegion_NS
- MPU_RLAR EN LL_MPU_ConfigRegion_NS
- MPU_RLAR AttrIndx LL_MPU_ConfigRegion_NS
-

LL_MPU_ConfigRegionAddress

Function name

__STATIC_INLINE void LL_MPU_ConfigRegionAddress (uint32_t Region, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure a MPU region address range.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegionAddress
- MPU_RBAR ADDR LL_MPU_ConfigRegionAddress
- MPU_RLAR ADDR LL_MPU_ConfigRegionAddress
-

LL_MPU_ConfigRegionAddress_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigRegionAddress_NS (uint32_t Region, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure a non-secure MPU region address range.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegionAddress_NS
- MPU_RBAR ADDR LL_MPU_ConfigRegionAddress_NS
- MPU_RLAR ADDR LL_MPU_ConfigRegionAddress_NS
-

LL_MPU_ConfigAttributes

Function name

__STATIC_INLINE void LL_MPU_ConfigAttributes (uint32_t AttIndex, uint32_t Attributes)

Function description

Configure a MPU attributes index.

Parameters

- **AttIndex:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **Attributes:** This parameter can be a combination of CORTEX LL MPU Attributes

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_MAIR0 Attribute LL_MPU_ConfigAttributes
- MPU_MAIR1 Attribute LL_MPU_ConfigAttributes
-

LL_MPU_ConfigAttributes_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigAttributes_NS (uint32_t AttIndex, uint32_t Attributes)

Function description

Configure a non-secure MPU attributes index.

Parameters

- **AttIndex:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **Attributes:** This parameter can be a combination of CORTEX LL MPU Attributes

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_MAIR0 Attribute LL_MPU_ConfigAttributes_NS
- MPU_MAIR1 Attribute LL_MPU_ConfigAttributes_NS
-

LL_MPU_SetRegionLimitAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionLimitAddress (uint32_t Region, uint32_t LimitAddress)

Function description

Configure a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionLimitAddress
- MPU_RLAR ADDR LL_MPU_SetRegionLimitAddress
-

LL_MPU_GetRegionLimitAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionLimitAddress (uint32_t Region)

Function description

Get a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*) (*) : For MPU_S only

Return values

- **Value:** of the region limit address

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionLimitAddress
-

LL_MPU_SetRegionBaseAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionBaseAddress (uint32_t Region, uint32_t BaseAddress)

Function description

Configure a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)
- **BaseAddress:** Value of region base address

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionBaseAddress
- MPU_RBAR ADDR LL_MPU_SetRegionBaseAddress
-

LL_MPU_GetRegionBaseAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionBaseAddress (uint32_t Region)

Function description

Get a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)

Return values

- **Value:** of the region base address

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionBaseAddress
-

LL_MPU_SetRegionAccess

Function name

__STATIC_INLINE void LL_MPU_SetRegionAccess (uint32_t Region, uint32_t Attributes)

Function description

Configure a MPU region access attributes and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*)
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Return values

- **None:**

Notes

- cortex-M33 supports 12 secure and 8 non secure regions. (*) : For MPU_S only

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionAccess
- MPU_RBAR XN LL_MPU_SetRegionAccess
- MPU_RBAR AP LL_MPU_SetRegionAccess
- MPU_RBAR SH LL_MPU_SetRegionAccess
-

LL_MPU_GetRegionAccess

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionAccess (uint32_t Region)

Function description

Get a MPU region access attributes.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
  - LL_MPU_REGION_NUMBER8 (*)
  - LL_MPU_REGION_NUMBER9 (*)
  - LL_MPU_REGION_NUMBER10 (*)
  - LL_MPU_REGION_NUMBER11 (*) (*) : For MPU_S only

Return values

- **return:** a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionAccess
- MPU_RBAR XN LL_MPU_GetRegionAccess
- MPU_RBAR AP LL_MPU_GetRegionAccess
- MPU_RBAR SH LL_MPU_GetRegionAccess
-

LL_MPU_SetRegionLimitAddress_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionLimitAddress_NS (uint32_t Region, uint32_t LimitAddress)

Function description

Configure a non-secure MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionLimitAddress_NS
- MPU_RLAR ADDR LL_MPU_SetRegionLimitAddress_NS
-

LL_MPU_GetRegionLimitAddress_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionLimitAddress_NS (uint32_t Region)

Function description

Get a non-secure MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **Value:** of the region limit address.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionLimitAddress_NS
-

LL_MPU_SetRegionBaseAddress_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionBaseAddress_NS (uint32_t Region, uint32_t BaseAddress)

Function description

Configure a non-secure MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address

Return values

- **None:**

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionBaseAddress_NS
- MPU_RBAR ADDR LL_MPU_SetRegionBaseAddress_NS
-

LL_MPU_GetRegionBaseAddress_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionBaseAddress_NS (uint32_t Region)

Function description

Get a non-secure MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **Value:** of the region base address.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionBaseAddress_NS
-

LL_MPU_SetRegionAccess_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionAccess_NS (uint32_t Region, uint32_t Attributes)

Function description

Configure a non-secure MPU region access attributes and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Return values

- **None:**

Notes

- cortex-M33 supports 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionAccess_NS
- MPU_RBAR XN LL_MPU_SetRegionAccess_NS
- MPU_RBAR AP LL_MPU_SetRegionAccess_NS
- MPU_RBAR SH LL_MPU_SetRegionAccess_NS
-

LL_MPU_GetRegionAccess_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionAccess_NS (uint32_t Region)

Function description

Get a non-secure MPU region access attributes.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **return:** a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionAccess_NS
- MPU_RBAR XN LL_MPU_GetRegionAccess_NS
- MPU_RBAR AP LL_MPU_GetRegionAccess_NS
- MPU_RBAR SH LL_MPU_GetRegionAccess_NS
-

89.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

89.2.1 CORTEX

CORTEX

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source

Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

LL_HANDLER_FAULT_SECURE

Secure fault

CORTEX LL MPU Region Access Attributes

MPU_ACCESS_MSK

CORTEX LL MPU Instruction Access Shareable

LL_MPU_ACCESS_NOT_SHAREABLE

MPU region not shareable

LL_MPU_ACCESS_OUTER_SHAREABLE

MPU region outer shareable

LL_MPU_ACCESS_INNER_SHAREABLE

MPU region inner shareable

CORTEX LL MPU Attributes

LL_MPU_DEVICE_nGnRnE

Device, noGather, noReorder, noEarly acknowledge.

LL_MPU_DEVICE_nGnRE

Device, noGather, noReorder, Early acknowledge.

LL_MPU_DEVICE_nGRE

Device, noGather, Reorder, Early acknowledge.

LL_MPU_DEVICE_GRE

Device, Gather, Reorder, Early acknowledge.

LL_MPU_WRITE_THROUGH

Normal memory, write-through.

LL_MPU_NOT_CACHEABLE

Normal memory, non-cacheable.

LL_MPU_WRITE_BACK

Normal memory, write-back.

LL_MPU_TRANSIENT

Normal memory, transient.

LL_MPU_NON_TRANSIENT

Normal memory, non-transient.

LL_MPU_NO_ALLOCATE

Normal memory, no allocate.

LL_MPU_W_ALLOCATE

Normal memory, write allocate.

LL_MPU_R_ALLOCATE

Normal memory, read allocate.

LL_MPU_RW_ALLOCATE

Normal memory, read/write allocate.

CORTEX LL MPU Memory Attributes Index

LL_MPU_ATTRIBUTES_NUMBER0

MPU attribute number 0

LL_MPU_ATTRIBUTES_NUMBER1

MPU attribute number 1

LL_MPU_ATTRIBUTES_NUMBER2

MPU attribute number 2

LL_MPU_ATTRIBUTES_NUMBER3

MPU attribute number 3

LL_MPU_ATTRIBUTES_NUMBER4

MPU attribute number 4

LL_MPU_ATTRIBUTES_NUMBER5

MPU attribute number 5

LL_MPU_ATTRIBUTES_NUMBER6

MPU attribute number 6

LL_MPU_ATTRIBUTES_NUMBER7

MPU attribute number 7

CORTEX LL MPU HFNMI and PRIVILEGED Access control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE

MPU is disabled during HardFault and NMI handlers, privileged software access to the default memory map is disabled

LL_MPU_CTRL_HARDFAULT_NMI

MPU is enabled during HardFault and NMI handlers, privileged software access to the default memory map is disabled

LL_MPU_CTRL_PRIVILEGED_DEFAULT

MPU is disabled during HardFault and NMI handlers, privileged software access to the default memory map is enabled

LL_MPU_CTRL_HFNMI_PRIVDEF

MPU is enabled during HardFault and NMI handlers, privileged software access to the default memory map is enabled

CORTEX LL MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

MPU region execution permitted if read permitted

LL_MPU_INSTRUCTION_ACCESS_DISABLE

MPU region execution not permitted

CORTEX LL MPU Region Enable

LL_MPU_REGION_ENABLE

MPU region enabled

LL_MPU_REGION_DISABLE

MPU region disabled

CORTEX LL MPU Region Index

LL_MPU_REGION_NUMBER0

MPU region number 0

LL_MPU_REGION_NUMBER1

MPU region number 1

LL_MPU_REGION_NUMBER2

MPU region number 2

LL_MPU_REGION_NUMBER3

MPU region number 3

LL_MPU_REGION_NUMBER4

MPU region number 4

LL_MPU_REGION_NUMBER5

MPU region number 5

LL_MPU_REGION_NUMBER6

MPU region number 6

LL_MPU_REGION_NUMBER7

MPU region number 7

LL_MPU_REGION_NUMBER8

MPU region number 8

LL_MPU_REGION_NUMBER9

MPU region number 9

LL_MPU_REGION_NUMBER10

MPU region number 10

LL_MPU_REGION_NUMBER11

MPU region number 11

CORTEX LL MPU Region Permission Attributes

LL_MPU_REGION_PRIV_RW

MPU region Read/write by privileged code only

LL_MPU_REGION_ALL_RW

MPU region Read/write by any privilege level

LL_MPU_REGION_PRIV_RO

MPU region Read-only by privileged code only

LL_MPU_REGION_ALL_RO

MPU region Read-only by any privilege level

90 LL CRC Generic Driver

90.1 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

90.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name

__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)

Function description

Reset the CRC calculation unit.

Parameters

- **CRCx:** CRC Instance

Return values

- **None:**

Notes

- If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to LL API cross reference:

- CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)

Function description

Configure size of the polynomial.

Parameters

- **CRCx:** CRC Instance
- **PolySize:** This parameter can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (const CRC_TypeDef * CRCx)

Function description

Return size of the polynomial.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return type of reversal for input data bit order.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return type of reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name

__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)

Function description

Initialize the Programmable initial CRC value.

Parameters

- **CRCx:** CRC Instance
- **InitCrc:** Value to be programmed in Programmable initial CRC value register

Return values

- **None:**

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to write the correct value
- LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInitialData (const CRC_TypeDef * CRCx)

Function description

Return current Initial CRC value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable initial CRC value register

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to read the correct value

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)

Function description

Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).

Parameters

- **CRCx:** CRC Instance
- **PolynomCoef:** Value to be programmed in Programmable Polynomial value register

Return values

- **None:**

Notes

- LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.
- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (const CRC_TypeDef * CRCx)

Function description

Return current Programmable polynomial value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable Polynomial value register

Notes

- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name

__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Write given 32-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name

__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)

Function description

Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name

__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)

Function description

Write given 8-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name

__STATIC_INLINE uint32_t LL_CRC_ReadData32 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (32 bits).

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name

__STATIC_INLINE uint16_t LL_CRC_ReadData16 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (16 bits).

Notes

- This function is expected to be used in a 16 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData8 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (8 bits).

Notes

- This function is expected to be used in a 8 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData7 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (7 bits).

Notes

- This function is expected to be used in a 7 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name

__STATIC_INLINE uint32_t LL_CRC_Read_IDR (const CRC_TypeDef * CRCx)

Function description

Return data stored in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** stored in CRC_IDR register (General-purpose 32-bit data register).

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name

__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Store data in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be stored in CRC_IDR register (32-bit) between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name

ErrorStatus LL_CRC_DeInit (const CRC_TypeDef * CRCx)

Function description

De-initialize CRC registers (Registers restored to their default values).

Parameters

- **CRCx:** CRC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRC registers are de-initialized
  - ERROR: CRC registers are not de-initialized

90.2 CRC Firmware driver defines

The following section lists the various define and macros of the module.

90.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE

Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY

Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE

Input Data bit order not affected

LL_CRC_INDATA_REVERSE_BYTE

Input Data bit reversal done by byte

LL_CRC_INDATA_REVERSE_HALFWORD

Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD

Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE

Output Data bit order not affected

LL_CRC_OUTDATA_REVERSE_BIT

Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B

32 bits Polynomial size

LL_CRC_POLYLENGTH_16B

16 bits Polynomial size

LL_CRC_POLYLENGTH_8B

8 bits Polynomial size

LL_CRC_POLYLENGTH_7B

7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

91 LL CRS Generic Driver

91.1 CRS Firmware driver API description

The following section lists the various functions of the CRS library.

91.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)

Function description

Enable Frequency error counter.

Return values

- **None:**

Notes

- When this bit is set, the CRS_CFGR register is write-protected and cannot be modified

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)

Function description

Disable Frequency error counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)

Function description

Check if Frequency error counter is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)

Function description

Enable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

LL_CRS_DisableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)

Function description

Disable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)

Function description

Check if Automatic trimming is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name

__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)

Function description

Set HSI48 oscillator smooth trimming.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 63

Return values

- **None:**

Notes

- When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only
- Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)

Function description

Get HSI48 oscillator smooth trimming.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 63

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name

__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)

Function description

Set counter reload value.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)

Function description

Get counter reload value.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name

__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)

Function description

Set frequency error limit.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 255

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)

Function description

Get frequency error limit.

Return values

- **A:** number between Min_Data = 0 and Max_Data = 255

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_GetFreqErrorLimit

LL_CRS_SetSyncDivider

Function name

__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)

Function description

Set division factor for SYNC signal.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)

Function description

Get division factor for SYNC signal.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name

__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)

Function description

Set SYNC signal source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)

Function description

Get SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name

__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)

Function description

Set input polarity for the SYNC signal source.

Parameters

- **Polarity:** This parameter can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)

Function description

Get input polarity for the SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name

__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)

Function description

Configure CRS for the synchronization.

Parameters

- **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 63
- **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF
- **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255
- **Settings:** This parameter can be a combination of the following values:
  - LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128
  - LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB
  - LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_ConfigSynchronization
- CFGR RELOAD LL_CRS_ConfigSynchronization
- CFGR FELIM LL_CRS_ConfigSynchronization
- CFGR SYNCDIV LL_CRS_ConfigSynchronization
- CFGR SYNCSRC LL_CRS_ConfigSynchronization
- CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name

__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)

Function description

Generate software SYNC event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)

Function description

Get the frequency error direction latched in the time of the last SYNC event.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_FREQ_ERROR_DIR_UP
  - LL_CRS_FREQ_ERROR_DIR_DOWN

Reference Manual to LL API cross reference:

- ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)

Function description

Get the frequency error counter value latched in the time of the last SYNC event.

Return values

- **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)

Function description

Check if SYNC event OK signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)

Function description

Check if SYNC warning signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)

Function description

Check if Synchronization or trimming error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)

Function description

Check if Expected SYNC signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)

Function description

Check if SYNC error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)

Function description

Check if SYNC missed error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)

Function description

Check if Trimming overflow or underflow occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)

Function description

Clear the SYNC event OK flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)

Function description

Clear the SYNC warning flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)

Function description

Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)

Function description

Clear Expected SYNC flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)

Function description

Enable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)

Function description

Disable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK (void)

Function description

Check if SYNC event OK interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)

Function description

Enable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)

Function description

Disable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)

Function description

Check if SYNC warning interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)

Function description

Enable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)

Function description

Disable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)

Function description

Check if Synchronization or trimming error interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)

Function description

Enable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)

Function description

Disable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)

Function description

Check if Expected SYNC interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name

ErrorStatus LL_CRS_DeInit (void)

Function description

De-Initializes CRS peripheral registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRS registers are de-initialized
  - ERROR: not applicable

91.2 CRS Firmware driver defines

The following section lists the various define and macros of the module.

91.2.1 CRS

CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is specified in the product datasheet. A higher TRIM value corresponds to a higher output frequency.

Frequency Error Direction

LL_CRS_FREQ_ERROR_DIR_UP

Upcounting direction, the actual frequency is above the target

LL_CRS_FREQ_ERROR_DIR_DOWN

Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1

Synchro Signal not divided (default)

LL_CRS_SYNC_DIV_2

Synchro Signal divided by 2

LL_CRS_SYNC_DIV_4

Synchro Signal divided by 4

LL_CRS_SYNC_DIV_8

Synchro Signal divided by 8

LL_CRS_SYNC_DIV_16

Synchro Signal divided by 16

LL_CRS_SYNC_DIV_32

Synchro Signal divided by 32

LL_CRS_SYNC_DIV_64

Synchro Signal divided by 64

LL_CRS_SYNC_DIV_128

Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

LL_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

LL_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

LL_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

Exported_Macros_Calculate_Reload

__LL_CRS_CALC_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

92 LL DAC Generic Driver

92.1 DAC Firmware driver registers structures

92.1.1 LL_DAC_InitTypeDef

LL_DAC_InitTypeDef is defined in the stm32h5xx_ll_dac.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***
- ***uint32_t OutputConnection***
- ***uint32_t OutputMode***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***
  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***
  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***
  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE***
  Note:
  – If waveform automatic generation mode is disabled, this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()**, **LL_DAC_SetWaveTriangleAmplitude()** depending on the wave automatic generation selected.
- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***
  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputConnection***
  Set the output connection for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_CONNECTION***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputConnection()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputMode***
  Set the output mode normal or sample-and-hold for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputMode()**.

92.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

92.2.1 Detailed description of functions

LL_DAC_SetHighFrequencyMode

Function name

__STATIC_INLINE void LL_DAC_SetHighFrequencyMode (DAC_TypeDef * DACx, uint32_t HighFreqMode)

Function description

Set the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance
- **HighFreqMode:** This parameter can be one of the following values:
  - LL_DAC_HIGH_FREQ_MODE_DISABLE
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_SetHighFrequencyMode

LL_DAC_GetHighFrequencyMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetHighFrequencyMode (const DAC_TypeDef * DACx)

Function description

Get the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_HIGH_FREQ_MODE_DISABLE
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  - LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_GetHighFrequencyMode

LL_DAC_SetMode

Function name

__STATIC_INLINE void LL_DAC_SetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ChannelMode)

Function description

Set the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **ChannelMode:** This parameter can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_SetMode
- CR CEN2 LL_DAC_SetMode

LL_DAC_GetMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetMode (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_GetMode
- CR CEN2 LL_DAC_GetMode

LL_DAC_SetTrimmingValue

Function name

__STATIC_INLINE void LL_DAC_SetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TrimmingValue)

Function description

Set the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_SetTrimmingValue
- CCR OTRIM2 LL_DAC_SetTrimmingValue

LL_DAC_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTrimmingValue (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_GetTrimmingValue
- CCR OTRIM2 LL_DAC_GetTrimmingValue

LL_DAC_SetTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM4_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM5_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM6_TRG
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM15_TRGO (2)
  - LL_DAC_TRIG_EXT_LPTIM1_CH1
  - LL_DAC_TRIG_EXT_LPTIM2_CH1
  - LL_DAC_TRIG_EXT_EXTI_LINE9

  (1) On this STM32 series, parameter not available on all devices. Only available on STM32H503xx (refer to device reference manual for supported features list) (2) On this STM32 series, parameter not available on all devices. Only available on STM32H563/H573xx (refer to device reference manual for supported features list)

Return values

- **None:**

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM3_TRGO (1)
  - LL_DAC_TRIG_EXT_TIM4_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM5_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM6_TRG
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO (2)
  - LL_DAC_TRIG_EXT_TIM15_TRGO (2)
  - LL_DAC_TRIG_EXT_LPTIM1_CH1
  - LL_DAC_TRIG_EXT_LPTIM2_CH1
  - LL_DAC_TRIG_EXT_EXTI_LINE9

  (1) On this STM32 series, parameter not available on all devices. Only available on STM32H503xx (refer to device reference manual for supported features list) (2) On this STM32 series, parameter not available on all devices. Only available on STM32H563/H573xx (refer to device reference manual for supported features list)

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name

__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)

Function description

Set the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **WaveAutoGeneration:** This parameter can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_SetWaveAutoGeneration
- CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_GetWaveAutoGeneration
- CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name

__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)

Function description

Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **NoiseLFSRMask:** This parameter can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
- CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_ConfigOutput

Function name

__STATIC_INLINE void LL_DAC_ConfigOutput (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode, uint32_t OutputBuffer, uint32_t OutputConnection)

Function description

Set the output for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- This function set several features: mode normal or sample-and-holdbufferconnection to GPIO or internal path. These features can also be set individually using dedicated functions:LL_DAC_SetOutputBuffer()LL_DAC_SetOutputMode()LL_DAC_SetOutputConnection()
- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).
- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_ConfigOutput
- CR MODE2 LL_DAC_ConfigOutput

LL_DAC_SetOutputMode

Function name

__STATIC_INLINE void LL_DAC_SetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode)

Function description

Set the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Return values

- **None:**

Notes

- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputMode
- CR MODE2 LL_DAC_SetOutputMode

LL_DAC_GetOutputMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputMode (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputMode
- CR MODE2 LL_DAC_GetOutputMode

LL_DAC_SetOutputBuffer

Function name

__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)

Function description

Set the output buffer for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, when buffer is enabled, its offset can be trimmed: factory calibration default values can be replaced by user trimming values, using function LL_DAC_SetTrimmingValue().

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputBuffer
- CR MODE2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output buffer state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputBuffer
- CR MODE2 LL_DAC_GetOutputBuffer

LL_DAC_SetOutputConnection

Function name

__STATIC_INLINE void LL_DAC_SetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputConnection)

Function description

Set the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputConnection
- CR MODE2 LL_DAC_SetOutputConnection

LL_DAC_GetOutputConnection

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputConnection (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputConnection
- CR MODE2 LL_DAC_GetOutputConnection

LL_DAC_SetSampleAndHoldSampleTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SampleTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **SampleTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Notes

- Sample time must be set when DAC channel is disabled or during DAC operation when DAC channel flag BWSTx is reset, otherwise the setting is ignored. Check BWSTx flag state using function "LL_DAC_IsActiveFlag_BWSTx()".

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_SetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_SetSampleAndHoldSampleTime

LL_DAC_GetSampleAndHoldSampleTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldSampleTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_GetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_GetSampleAndHoldSampleTime

LL_DAC_SetSampleAndHoldHoldTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t HoldTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **HoldTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_SetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_SetSampleAndHoldHoldTime

LL_DAC_GetSampleAndHoldHoldTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldHoldTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_GetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_GetSampleAndHoldHoldTime

LL_DAC_SetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t RefreshTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **RefreshTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_SetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_SetSampleAndHoldRefreshTime

LL_DAC_GetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldRefreshTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_GetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_GetSampleAndHoldRefreshTime

LL_DAC_SetSignedFormat

Function name

__STATIC_INLINE void LL_DAC_SetSignedFormat (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SignedFormat)

Function description

Set the signed format for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **SignedFormat:** This parameter can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, signed format can be used to inject Q1.15, Q1.11, Q1.7 signed format data to DAC. Ex when using 12bits data format (Q1.11 is used): 0x800 will output 0v level 0xFFF will output mid-scale level 0x000 will output mid-scale level 0x7FF will output full-scale level

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_SetSignedFormat
- MCR SINFORMAT2 LL_DAC_SetSignedFormat

LL_DAC_GetSignedFormat

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSignedFormat (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the signed format state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_GetSignedFormat
- MCR SINFORMAT2 LL_DAC_GetSignedFormat

LL_DAC_EnableDMAReq

Function name

__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_EnableDMAReq
- CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name

__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_DisableDMAReq
- CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA transfer request state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_IsDMAReqEnabled
- CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_EnableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_EnableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_EnableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_EnableDMADoubleDataMode

LL_DAC_DisableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_DisableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_DisableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_DisableDMADoubleDataMode

LL_DAC_IsDMADoubleDataModeEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMADoubleDataModeEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA double data mode state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_IsDMADoubleDataModeEnabled
- MCR DMADOUBLE2 LL_DAC_IsDMADoubleDataModeEnabled

LL_DAC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (const DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)

Function description

Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values

- **DAC:** register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name

__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name

__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC enable state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_IsReady

Function name

__STATIC_INLINE uint32_t LL_DAC_IsReady (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC ready for conversion state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsReady
- SR DAC2RDY LL_DAC_IsReady

LL_DAC_EnableTrigger

Function name

__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name

__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_DisableTrigger
- CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC trigger state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_IsTriggerEnabled
- CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name

__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Trig DAC conversion by software for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_Init() LL_DAC_SetTriggerSource() with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:

- SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion
- SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_ConvertData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned
- DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned
- DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF
- **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned
- DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name

__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Retrieve output data currently generated for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).

Reference Manual to LL API cross reference:

- DOR1 DACC1DOR LL_DAC_RetrieveOutputData
- DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_CAL1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL1 (const DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG1 LL_DAC_IsActiveFlag_CAL1

LL_DAC_IsActiveFlag_CAL2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL2 (const DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG2 LL_DAC_IsActiveFlag_CAL2

LL_DAC_IsActiveFlag_BWST1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST1 (const DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST1 LL_DAC_IsActiveFlag_BWST1

LL_DAC_IsActiveFlag_BWST2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST2 (const DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST2 LL_DAC_IsActiveFlag_BWST2

LL_DAC_IsActiveFlag_DAC1RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC1RDY (const DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsActiveFlag_DAC1RDY

LL_DAC_IsActiveFlag_DAC2RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC2RDY (const DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC2RDY LL_DAC_IsActiveFlag_DAC2RDY

LL_DAC_IsActiveFlag_DORSTAT1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT1 (const DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT1 LL_DAC_IsActiveFlag_DORSTAT1

LL_DAC_IsActiveFlag_DORSTAT2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT2 (const DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT2 LL_DAC_IsActiveFlag_DORSTAT2

LL_DAC_IsActiveFlag_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (const DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (const DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

LL_DAC_IsEnabledIT_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (const DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (const DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_DeInit

Function name

ErrorStatus LL_DAC_DeInit (const DAC_TypeDef * DACx)

Function description

De-initialize registers of the selected DAC instance to their default reset values.

Parameters

- **DACx:** DAC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are de-initialized
  - ERROR: not applicable

LL_DAC_Init

Function name

ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, const LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Initialize some features of DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are initialized
  - ERROR: DAC registers are not initialized

Notes

- LL_DAC_Init() aims to ease basic configuration of a DAC channel. Leaving it ready to be enabled and output: a level by calling one of LL_DAC_ConvertData12RightAligned LL_DAC_ConvertData12LeftAligned LL_DAC_ConvertData8RightAligned or one of the supported autogenerated wave.
- This function allows configuration of: Output modeTriggerWave generation
- The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC channel must be disabled.

LL_DAC_StructInit

Function name

void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Set each LL_DAC_InitTypeDef field to default value.

Parameters

- **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

92.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

92.3.1 DAC

DAC

DAC channels

LL_DAC_CHANNEL_1

DAC channel 1

LL_DAC_CHANNEL_2

DAC channel 2

DAC flags

LL_DAC_FLAG_DMAUDR1

DAC channel 1 flag DMA underrun

LL_DAC_FLAG_CAL1

DAC channel 1 flag offset calibration status

LL_DAC_FLAG_BWST1

DAC channel 1 flag busy writing sample time

LL_DAC_FLAG_DAC1RDY

DAC channel 1 flag ready

LL_DAC_FLAG_DORSTAT1

DAC channel 1 flag output register

LL_DAC_FLAG_DMAUDR2

DAC channel 2 flag DMA underrun

LL_DAC_FLAG_CAL2

DAC channel 2 flag offset calibration status

LL_DAC_FLAG_BWST2

DAC channel 2 flag busy writing sample time

LL_DAC_FLAG_DAC2RDY

DAC channel 2 flag ready

LL_DAC_FLAG_DORSTAT2

DAC channel 2 flag output register

DAC high frequency interface mode

LL_DAC_HIGH_FREQ_MODE_DISABLE

High frequency interface mode disabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ

High frequency interface mode compatible to AHB>80MHz enabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

High frequency interface mode compatible to AHB>160MHz enabled

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)

LL_DAC_DELAY_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1

DAC channel 1 interruption DMA underrun

LL_DAC_IT_DMAUDRIE2

DAC channel 2 interruption DMA underrun

DAC operating mode

LL_DAC_MODE_NORMAL_OPERATION

DAC channel in mode normal operation

LL_DAC_MODE_CALIBRATION

DAC channel in mode calibration

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE

The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption

LL_DAC_OUTPUT_BUFFER_DISABLE

The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC channel output connection

LL_DAC_OUTPUT_CONNECT_GPIO

The selected DAC channel output is connected to external pin

LL_DAC_OUTPUT_CONNECT_INTERNAL

The selected DAC channel output is connected to on-chip peripherals via internal paths. On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. Refer to comments of function

DAC channel output mode

LL_DAC_OUTPUT_MODE_NORMAL

The selected DAC channel output is on mode normal.

LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

The selected DAC channel output is on mode sample-and-hold. Mode sample-and-hold requires an external capacitor, refer to description of function

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED

DAC channel data holding register 12 bits right aligned

LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED

DAC channel data holding register 12 bits left aligned

LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B

DAC channel resolution 12 bits

LL_DAC_RESOLUTION_8B

DAC channel resolution 8 bits

DAC channel signed format

LL_DAC_SIGNED_FORMAT_DISABLE

The selected DAC channel data format is not signed

LL_DAC_SIGNED_FORMAT_ENABLE

The selected DAC channel data format is signed

DAC trigger source

LL_DAC_TRIG_SOFTWARE

DAC channel conversion trigger internal (SW start)

LL_DAC_TRIG_EXT_TIM1_TRGO

DAC channel conversion trigger from external peripheral: TIM1 TRGO.

LL_DAC_TRIG_EXT_TIM2_TRGO

DAC channel conversion trigger from external peripheral: TIM2 TRGO.

LL_DAC_TRIG_EXT_TIM6_TRGO

DAC channel conversion trigger from external peripheral: TIM6 TRGO.

LL_DAC_TRIG_EXT_TIM7_TRGO

DAC channel conversion trigger from external peripheral: TIM7 TRGO.

LL_DAC_TRIG_EXT_LPTIM1_CH1

DAC channel conversion trigger from external peripheral: LPTIM1 CH1.

LL_DAC_TRIG_EXT_LPTIM2_CH1

DAC channel conversion trigger from external peripheral: LPTIM2 CH1.

LL_DAC_TRIG_EXT_EXTI_LINE9

DAC channel conversion trigger from external peripheral: external interrupt line 9.

LL_DAC_TRIG_EXT_TIM4_TRGO

DAC channel conversion trigger from external peripheral: TIM4 TRGO.

LL_DAC_TRIG_EXT_TIM5_TRGO

DAC channel conversion trigger from external peripheral: TIM5 TRGO.

LL_DAC_TRIG_EXT_TIM8_TRGO

DAC channel conversion trigger from external peripheral: TIM8 TRGO.

LL_DAC_TRIG_EXT_TIM15_TRGO

DAC channel conversion trigger from external peripheral: TIM15 TRGO.

LL_DAC_TRIG_EXT_LPTIM1_OUT

Keep old definition for compatibility

LL_DAC_TRIG_EXT_LPTIM2_OUT

Keep old definition for compatibility

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE

DAC channel wave auto generation mode disabled.

LL_DAC_WAVE_AUTO_GENERATION_NOISE

DAC channel wave auto generation mode enabled, set generated noise waveform.

LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0

Noise wave generation, unmask LFSR bit0, for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS1_0

Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS2_0

Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS4_0

Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS5_0

Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS6_0

Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS7_0

Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS8_0

Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS9_0

Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS10_0

Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1

Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_3

Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_7

Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_15

Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_31

Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_63

Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_127

Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_255

Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_511

Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023

Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_2047

Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_4095

Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return value:

- 1...2

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2

Return value:

- Returned: value can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

93 LL DCACHE Generic Driver

93.1 DCACHE Firmware driver API description

The following section lists the various functions of the DCACHE library.

93.1.1 Detailed description of functions

LL_DCACHE_Enable

Function name

__STATIC_INLINE void LL_DCACHE_Enable (DCACHE_TypeDef * DCACHEx)

Function description

Enable the selected DCACHE instance.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_Enable

LL_DCACHE_Disable

Function name

__STATIC_INLINE void LL_DCACHE_Disable (DCACHE_TypeDef * DCACHEx)

Function description

Disable the selected DCACHE instance.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_Disable

LL_DCACHE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabled (const DCACHE_TypeDef * DCACHEx)

Function description

Get the selected DCACHE instance enable state.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **0:** DCACHE is disabled, 1: DCACHE is enabled.

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_IsEnabled

LL_DCACHE_SetStartAddress

Function name

__STATIC_INLINE void LL_DCACHE_SetStartAddress (DCACHE_TypeDef * DCACHEx, uint32_t addr)

Function description

Set the dcache instance start command address.

Parameters

- **addr:** dcache command start address(Clean, Invalidate or Clean and Invalidate).
- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CMDRSADDRR LL_DCACHE_SetStartAddress

LL_DCACHE_GetStartAddress

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetStartAddress (const DCACHE_TypeDef * DCACHEx)

Function description

Get the dcache command start address.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Start:** address of dcache command

Reference Manual to LL API cross reference:

- CR CMDRSADDRR LL_DCACHE_GetStartAddress

LL_DCACHE_SetEndAddress

Function name

__STATIC_INLINE void LL_DCACHE_SetEndAddress (DCACHE_TypeDef * DCACHEx, uint32_t addr)

Function description

Set the dcache instance End command address.

Parameters

- **DCACHEx:** DCACHE instance
- **addr:** dcache command end address(Clean, Invalidate or Clean and Invalidate).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CMDREADDRR LL_DCACHE_SetEndAddress

LL_DCACHE_GetEndAddress

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetEndAddress (const DCACHE_TypeDef * DCACHEx)

Function description

Get the dcache command End address.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **End:** address of dcache command

Reference Manual to LL API cross reference:

- CR CMDREADDRR LL_DCACHE_GetEndAddress

LL_DCACHE_SetCommand

Function name

__STATIC_INLINE void LL_DCACHE_SetCommand (DCACHE_TypeDef * DCACHEx, uint32_t Command)

Function description

Set Dcache command.

Parameters

- **DCACHEx:** DCACHE instance
- **Command:** command to be applied for the dcache Command can be one of the following values:
  - LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_NO_OPERATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_SetCommand

LL_DCACHE_GetCommand

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetCommand (const DCACHE_TypeDef * DCACHEx)

Function description

Set Dcache command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DCACHE_COMMAND_NO_OPERATION
  - LL_DCACHE_COMMAND_CLEAN_BY_ADDR
  - LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_GetCommand

LL_DCACHE_StartCommand

Function name

__STATIC_INLINE void LL_DCACHE_StartCommand (DCACHE_TypeDef * DCACHEx)

Function description

Launch Dcache Command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_StartCommand

LL_DCACHE_SetReadBurstType

Function name

__STATIC_INLINE void LL_DCACHE_SetReadBurstType (DCACHE_TypeDef * DCACHEx, uint32_t ReadBurstType)

Function description

Set requested read burst type.

Parameters

- **DCACHEx:** DCACHE instance
- **ReadBurstType:** Burst type to be applied for Data Cache Burst type can be one of the following values:
  - LL_DCACHE_READ_BURST_WRAP
  - LL_DCACHE_READ_BURST_INCR

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HBURST LL_DCACHE_SetReadBurstType

LL_DCACHE_GetReadBurstType

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetReadBurstType (const DCACHE_TypeDef * DCACHEx)

Function description

Get requested read burst type.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DCACHE_READ_BURST_WRAP
  - LL_DCACHE_READ_BURST_INCR

Reference Manual to LL API cross reference:

- CR HBURST LL_DCACHE_GetReadBurstType

LL_DCACHE_Invalidate

Function name

__STATIC_INLINE void LL_DCACHE_Invalidate (DCACHE_TypeDef * DCACHEx)

Function description

Invalidate the Data cache.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHEINV LL_DCACHE_Invalidate

LL_DCACHE_EnableMonitors

Function name

__STATIC_INLINE void LL_DCACHE_EnableMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_DCACHE_MONITOR_READ_HIT
  - LL_DCACHE_MONITOR_READ_MISS
  - LL_DCACHE_MONITOR_WRITE_HIT
  - LL_DCACHE_MONITOR_WRITE_MISS
  - LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_EnableMonitors

LL_DCACHE_DisableMonitors

Function name

__STATIC_INLINE void LL_DCACHE_DisableMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_DCACHE_MONITOR_READ_HIT
  - LL_DCACHE_MONITOR_READ_MISS
  - LL_DCACHE_MONITOR_WRITE_HIT
  - LL_DCACHE_MONITOR_WRITE_MISS
  - LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_DisableMonitors

LL_DCACHE_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledMonitors (const DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Return the hit/miss monitor(s) enable state.

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_DCACHE_MONITOR_READ_HIT
  - LL_DCACHE_MONITOR_READ_MISS
  - LL_DCACHE_MONITOR_WRITE_HIT
  - LL_DCACHE_MONITOR_WRITE_MISS
  - LL_DCACHE_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_IsEnabledMonitors

LL_DCACHE_ResetMonitors

Function name

__STATIC_INLINE void LL_DCACHE_ResetMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Reset the Data Cache performance monitoring.

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** Monitoring type This parameter can be a combination of the following values:
  - LL_DCACHE_MONITOR_READ_HIT
  - LL_DCACHE_MONITOR_READ_MISS
  - LL_DCACHE_MONITOR_WRITE_HIT
  - LL_DCACHE_MONITOR_WRITE_MISS
  - LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RHITMRST/RMISSMRST/WHITMRST/WMISSMRST) LL_DCACHE_ResetMonitors

LL_DCACHE_Monitor_GetReadHitValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetReadHitValue (DCACHE_TypeDef * DCACHEx)

Function description

Get the Read Hit monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RHMONR LL_DCACHE_Monitor_GetReadHitValue

LL_DCACHE_Monitor_GetReadMissValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetReadMissValue (DCACHE_TypeDef * DCACHEx)

Function description

Get the Read Miss monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RMMONR LL_DCACHE_Monitor_GetReadMissValue

LL_DCACHE_Monitor_GetWriteHitValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetWriteHitValue (DCACHE_TypeDef * DCACHEx)

Function description

Get the Write Hit monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- WHMONR LL_DCACHE_Monitor_GetWriteHitValue

LL_DCACHE_Monitor_GetWriteMissValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetWriteMissValue (DCACHE_TypeDef * DCACHEx)

Function description

Get the Write Miss monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR LL_DCACHE_Monitor_GetWriteMissValue

LL_DCACHE_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Enable BusyEnd interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_EnableIT_BSYEND

LL_DCACHE_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Disable BusyEnd interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_DisableIT_BSYEND

LL_DCACHE_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_BSYEND (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the Busyend interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_IsEnabledIT_BSYEND

LL_DCACHE_EnableIT_ERR

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Enable Error interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_EnableIT_ERR

LL_DCACHE_DisableIT_ERR

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Disable Error interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_DisableIT_ERR

LL_DCACHE_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_ERR (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the Error interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_IsEnabledIT_ERR

LL_DCACHE_EnableIT_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Enable command end interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_EnableIT_CMDEND

LL_DCACHE_DisableIT_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Disable command end interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_DisableIT_CMDEND

LL_DCACHE_IsEnabledIT_CMDEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_CMDEND (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the command end interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_IsEnabledIT_CMDEND

LL_DCACHE_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Clear full invalidate busy end flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_DCACHE_ClearFlag_BSYEND

LL_DCACHE_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Clear cache error flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CERRF LL_DCACHE_ClearFlag_ERR

LL_DCACHE_ClearFlag_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Clear command end flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CCMDENDF LL_DCACHE_ClearFlag_CMDEND

LL_DCACHE_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BUSY (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache BUSY.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_DCACHE_IsActiveFlag_BUSY

LL_DCACHE_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BSYEND (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Busyend.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYENDF LL_DCACHE_IsActiveFlag_BSYEND

LL_DCACHE_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_ERR (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Error.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_DCACHE_IsActiveFlag_ERR

LL_DCACHE_IsActiveFlag_BUSYCMD

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BUSYCMD (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Busy command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYCMDF LL_DCACHE_IsActiveFlag_BUSYCMD

LL_DCACHE_IsActiveFlag_CMDEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_CMDEND (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache command end.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CMDENDF LL_DCACHE_IsActiveFlag_CMDEND

93.2 DCACHE Firmware driver defines

The following section lists the various define and macros of the module.

93.2.1 DCACHE

DCACHE

Clear Flags Defines

LL_DCACHE_FCR_CERRF

Cache error flag

LL_DCACHE_FCR_CBSYENDF

Full invalidate busy end flag

LL_DCACHE_FCR_CCMDENDF

Command end flag

Get Flags Defines

LL_DCACHE_SR_ERRF

Cache error flag

LL_DCACHE_SR_BUSYF

Busy flag

LL_DCACHE_SR_CMDENDF

Command end flag

LL_DCACHE_SR_BSYENDF

Full invalidate busy end flag

LL_DCACHE_SR_BUSYCMDF

Command busy flag

IT Defines

LL_DCACHE_IER_BSYENDIE

Busy end interrupt

LL_DCACHE_IER_ERRIE

Cache error interrupt

LL_DCACHE_IER_CMDENDIE

Command end interrupt

Common write and read registers Macros

LL_DCACHE_WriteReg

Description:

- Write a value in DCACHE register.

Parameters:

- __INSTANCE__: DCACHE Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DCACHE_ReadReg

Description:

- Read a value in DCACHE register.

Parameters:

- __INSTANCE__: DCACHE Instance
- __REG__: Register to be read

Return value:

- Register: value

94 LL DLYB Generic Driver

94.1 DLYB Firmware driver registers structures

94.1.1 LL_DLYB_CfgTypeDef

LL_DLYB_CfgTypeDef is defined in the stm32h5xx_ll_dlyb.h

Data Fields

- ***uint32_t Units***
- ***uint32_t PhaseSel***

Field Documentation

- ***uint32_t LL_DLYB_CfgTypeDef::Units***
  Specifies the Delay of a unit delay cell. This parameter can be a value between 0 and DLYB_MAX_UNIT
- ***uint32_t LL_DLYB_CfgTypeDef::PhaseSel***
  Specifies the Phase for the output clock. This parameter can be a value between 0 and DLYB_MAX_SELECT

94.2 DLYB Firmware driver API description

The following section lists the various functions of the DLYB library.

94.2.1 DelayBlock peripheral features

The DelayBlock is used to generate an Output clock which is de-phased from the Input clock. The phase of the Output clock is programmed by FW. The Output clock is then used to clock the receive data in i.e. a SDMMC, OSPI or QSPI interface. The delay is Voltage and Temperature dependent, which may require FW to do re-tuning and recenter the Output clock phase to the receive data.

The DelayBlock features include the following:

- Input clock frequency.
- Up to 12 oversampling phases.

94.2.2 How to use this driver

This driver is a considered as a driver of service for external devices drivers that interfaces with the DELAY peripheral. The LL_DLYB_SetDelay() function, configure the Delay value configured on SEL and UNIT. The LL_DLYB_GetDelay() function, return the Delay value configured on SEL and UNIT. The LL_DLYB_GetClockPeriod()function, get the clock period.

94.2.3 Detailed description of functions

LL_DLYB_Enable

Function name

__STATIC_INLINE void LL_DLYB_Enable (DLYB_TypeDef * DLYBx)

Function description

DLYB Enable.

Parameters

- **DLYBx:** DLYB Instance

Return values

- **None:**

LL_DLYB_Disable

Function name

__STATIC_INLINE void LL_DLYB_Disable (DLYB_TypeDef * DLYBx)

Function description

Disable the DLYB.

Parameters

- **DLYBx:** DLYB Instance.

Return values

- **None:**

LL_DLYB_SetDelay

Function name

void LL_DLYB_SetDelay (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Set the Delay value configured on SEL and UNIT.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: the Delay value is set.
  - ERROR: the Delay value is not set.

LL_DLYB_GetDelay

Function name

void LL_DLYB_GetDelay (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Get the Delay value configured on SEL and UNIT.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: the Delay value is received.
  - ERROR: the Delay value is not received.

LL_DLYB_GetClockPeriod

Function name

uint32_t LL_DLYB_GetClockPeriod (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Get the clock period.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: there is a valid period detected and stored in pdlyb_cfg.
  - ERROR: there is no valid period detected.

94.3 DLYB Firmware driver defines

The following section lists the various define and macros of the module.

94.3.1 DLYB

DLYB

DLYB Flags

DLYB_FLAG_LNGF

95 LL DMA Generic Driver

95.1 DMA Firmware driver registers structures

95.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32h5xx_ll_dma.h

Data Fields

- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t Direction***
- ***uint32_t BlkHWRequest***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcIncMode***
- ***uint32_t DestIncMode***
- ***uint32_t Priority***
- ***uint32_t BlkDataLength***
- ***uint32_t BlkRptCount***
- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t Request***
- ***uint32_t TransferEventMode***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t SrcByteExchange***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t DestAllocatedPort***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t LinkStepMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrOffset***
- ***uint32_t DestAddrOffset***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t LinkedListBaseAddr***
- ***uint32_t LinkedListAddrOffset***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::SrcAddress***
  This field specify the data transfer source address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddress()**.

- ***uint32_t LL_DMA_InitTypeDef::DestAddress***
  This field specify the data transfer destination address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddress()**.
- ***uint32_t LL_DMA_InitTypeDef::Direction***
  This field specify the data transfer direction. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_DIRECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkHWRequest***
  This field specify the hardware request unity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_BLKHW_REQUEST***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkHWRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::DataAlignment***
  This field specify the transfer data alignment. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DATA_ALIGNMENT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataAlignment()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetDestBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcDataWidth***
  This field specify the source data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::DestDataWidth***
  This field specify the destination data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcIncMode***
  This field specify the source burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestIncMode***
  This field specify the destination burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_PRIORITY_LEVEL***. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptCount()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerMode***
  This field specify the trigger mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerMode()**.

- ***uint32_t LL_DMA_InitTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_POLARITY***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerPolarity()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerSelection***
  This field specify the trigger event selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetHWTrigger()**.
- ***uint32_t LL_DMA_InitTypeDef::Request***
  This field specify the peripheral request selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_REQUEST_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_HALFWORD_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestHWordExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::DestByteExchange***
  This field specify the destination byte exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcByteExchange***
  This field specify the source byte exchange. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SRC_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINK_STEP_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrUpdateMode***
  This field specify the source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_SRC_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAddrUpdateMode***
  This field specify the destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_DEST_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrOffset***
  This field specifies the source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdateValue()**.

- ***uint32_t LL_DMA_InitTypeDef::DestAddrOffset***
  This field specifies the destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListBaseAddr***
  This field specify the linked list base address. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0xFFFF0000 (where the 4 first bytes are always forced to 0). This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListBaseAddr()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListAddrOffset***
  Specifies the linked list address offset. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0x0000FFFC. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListAddrOffset()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***
  Specifies the transfer mode for the DMA channel. This parameter can be a value of ***DMA_LL_TRANSFER_MODE***

95.1.2 LL_DMA_InitLinkedListTypeDef

LL_DMA_InitLinkedListTypeDef is defined in the stm32h5xx_ll_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***

Field Documentation

- ***uint32_t LL_DMA_InitLinkedListTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_PRIORITY_LEVEL***. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINK_STEP_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.

- ***uint32_t LL_DMA_InitLinkedListTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.

95.1.3 LL_DMA_InitNodeTypeDef

LL_DMA_InitNodeTypeDef is defined in the stm32h5xx_ll_dma.h

Data Fields

- ***uint32_t DestSecure***
- ***uint32_t DestAllocatedPort***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t DestBurstLength***
- ***uint32_t DestIncMode***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcSecure***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t SrcByteExchange***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t SrcIncMode***
- ***uint32_t SrcDataWidth***
- ***uint32_t TransferEventMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerMode***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t Request***
- ***uint32_t Mode***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t BlkRptCount***
- ***uint32_t BlkDataLength***
- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t DestAddrOffset***
- ***uint32_t SrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t UpdateRegisters***
- ***uint32_t NodeType***

Field Documentation

- ***uint32_t LL_DMA_InitNodeTypeDef::DestSecure***
  This field specify the destination secure. This parameter can be a value of ***DMA_LL_EC_DESTINATION_SECURITY_ATTRIBUTE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. This parameter can be a value of ***DMA_LL_EC_DESTINATION_ALLOCATED_PORT***.

- ***uint32_t LL_DMA_InitNodeTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. This parameter can be a value of ***DMA_LL_EC_DEST_HALFWORD_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestByteExchange***
  This field specify the destination byte exchange. This parameter can be a value of ***DMA_LL_EC_DEST_BYTE_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestIncMode***
  This field specify the destination increment mode. This parameter can be a value of ***DMA_LL_EC_DESTINATION_INCREMENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestDataWidth***
  This field specify the destination data width. This parameter can be a value of ***DMA_LL_EC_DESTINATION_DATA_WIDTH***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcSecure***
  This field specify the source secure. This parameter can be a value of ***DMA_LL_EC_SOURCE_SECURITY_ATTRIBUTE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. This parameter can be a value of ***DMA_LL_EC_SOURCE_ALLOCATED_PORT***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcByteExchange***
  This field specify the source byte exchange. This parameter can be a value of ***DMA_LL_EC_SRC_BYTE_EXCHANGE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DataAlignment***
  This field specify the transfer data alignment. This parameter can be a value of ***DMA_LL_EC_DATA_ALIGNMENT***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcIncMode***
  This field specify the source increment mode. This parameter can be a value of ***DMA_LL_EC_SOURCE_INCREMENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcDataWidth***
  This field specify the source data width. This parameter can be a value of ***DMA_LL_EC_SOURCE_DATA_WIDTH***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TransferEventMode***
  This field specify the transfer event mode. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. This parameter can be a value of ***DMA_LL_EC_TRIGGER_POLARITY***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerSelection***
  This field specify the trigger event selection. This parameter can be a value of ***DMA_LL_EC_TRIGGER_SELECTION***.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerMode***
  This field specify the trigger mode. This parameter can be a value of ***DMA_LL_EC_TRIGGER_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkHWRequest***
  This field specify the hardware request unity. This parameter can be a value of ***DMA_LL_EC_BLKHW_REQUEST***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Direction***
  This field specify the transfer direction. This parameter can be a value of ***DMA_LL_EC_TRANSFER_DIRECTION***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Request***
  This field specify the peripheral request selection. This parameter can be a value of ***DMA_LL_EC_REQUEST_SELECTION***.
- ***uint32_t LL_DMA_InitNodeTypeDef::Mode***
  This field DMA Transfer Mode. This parameter can be a value of ***Section 95.3.1 DMA_Transfer_Mode***.

- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. This parameter can be a value of ***DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrUpdateMode***
  This field specify the Destination address update mode. This parameter can be a value of ***DMA_LL_EC_DEST_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrUpdateMode***
  This field specify the Source address update mode. This parameter can be a value of ***DMA_LL_EC_SRC_ADDR_UPDATE_MODE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddress***
  This field specify the transfer source address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddress***
  This field specify the transfer destination address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrOffset***
  This field specifies the destination address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrOffset***
  This field specifies the source address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::UpdateRegisters***
  Specifies the linked list register update. This parameter can be a value of ***DMA_LL_EC_LINKEDLIST_REGISTER_UPDATE***.
- ***uint32_t LL_DMA_InitNodeTypeDef::NodeType***
  Specifies the node type to be created. This parameter can be a value of ***DMA_LL_EC_LINKEDLIST_NODE_TYPE***.

95.1.4 LL_DMA_LinkNodeTypeDef

LL_DMA_LinkNodeTypeDef is defined in the stm32h5xx_ll_dma.h

Data Fields

- ***__IO uint32_t LinkRegisters***

Field Documentation

- ***__IO uint32_t LL_DMA_LinkNodeTypeDef::LinkRegisters[8]***

95.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

95.2.1 LL DMA driver acronyms

Acronyms table : ========== || Acronym || || ========== || SRC || Source || || DEST || Destination || || ADDR || Address || || ADDRS || Addresses || || INC || Increment / Incremented || || DEC || Decrement / Decremented || || BLK || Block || || RPT || Repeat / Repeated || || TRIG || Trigger || ==========

95.2.2 Detailed description of functions

LL_DMA_EnableChannel

Function name

__STATIC_INLINE void LL_DMA_EnableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name

__STATIC_INLINE void LL_DMA_DisableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ResetChannel

Function name

__STATIC_INLINE void LL_DMA_ResetChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Reset channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR RESET LL_DMA_ResetChannel

LL_DMA_SuspendChannel

Function name

__STATIC_INLINE void LL_DMA_SuspendChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Suspend channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_SuspendChannel

LL_DMA_ResumeChannel

Function name

__STATIC_INLINE void LL_DMA_ResumeChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Resume channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_ResumeChannel

LL_DMA_IsSuspendedChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsSuspendedChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is suspended.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_IsSuspendedChannel

LL_DMA_SetLinkedListBaseAddr

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListBaseAddr)

Function description

Set linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **LinkedListBaseAddr:** Between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_SetLinkedListBaseAddr

LL_DMA_GetLinkedListBaseAddr

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Value:** between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_GetLinkedListBaseAddr

LL_DMA_ConfigControl

Function name

__STATIC_INLINE void LL_DMA_ConfigControl (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters linked to channel control.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT or LL_DMA_LOW_PRIORITY_MID_WEIGHT or LL_DMA_LOW_PRIORITY_HIGH_WEIGHT or LL_DMA_HIGH_PRIORITY
  - LL_DMA_LINK_ALLOCATED_PORT0 or LL_DMA_LINK_ALLOCATED_PORT1
  - LL_DMA_LSM_FULL_EXECUTION or LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_ConfigControl
- CCR LAP LL_DMA_ConfigControl
- CCR LSM LL_DMA_ConfigControl

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)

Function description

Set priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_GetChannelPriorityLevel

LL_DMA_SetLinkAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkAllocatedPort)

Function description

Set linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **LinkAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_LINK_ALLOCATED_PORT0
  - LL_DMA_LINK_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_SetLinkAllocatedPort

LL_DMA_GetLinkAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LINK_ALLOCATED_PORT0
  - LL_DMA_LINK_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_GetLinkAllocatedPort

LL_DMA_SetLinkStepMode

Function name

__STATIC_INLINE void LL_DMA_SetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkStepMode)

Function description

Set link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **LinkStepMode:** This parameter can be one of the following values:
  - LL_DMA_LSM_FULL_EXECUTION
  - LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_SetLinkStepMode

LL_DMA_GetLinkStepMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LSM_FULL_EXECUTION
  - LL_DMA_LSM_1LINK_EXECUTION

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_GetLinkStepMode

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure data transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0 or LL_DMA_DEST_ALLOCATED_PORT1
  - LL_DMA_DEST_HALFWORD_PRESERVE or LL_DMA_DEST_HALFWORD_EXCHANGE
  - LL_DMA_DEST_BYTE_PRESERVE or LL_DMA_DEST_BYTE_EXCHANGE
  - LL_DMA_SRC_BYTE_PRESERVE or LL_DMA_SRC_BYTE_EXCHANGE
  - LL_DMA_DEST_FIXED or LL_DMA_DEST_INCREMENT
  - LL_DMA_DEST_DATAWIDTH_BYTE or LL_DMA_DEST_DATAWIDTH_HALFWORD or LL_DMA_DEST_DATAWIDTH_WORD
  - LL_DMA_SRC_ALLOCATED_PORT0 or LL_DMA_SRC_ALLOCATED_PORT1
  - LL_DMA_DATA_ALIGN_ZEROPADD or LL_DMA_DATA_ALIGN_SIGNEXTPADD or LL_DMA_DATA_PACK_UNPACK
  - LL_DMA_SRC_FIXED or LL_DMA_SRC_INCREMENT
  - LL_DMA_SRC_DATAWIDTH_BYTE or LL_DMA_SRC_DATAWIDTH_HALFWORD or LL_DMA_SRC_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_ConfigTransfer
- CTR1 DHX LL_DMA_ConfigTransfer
- CTR1 DBX LL_DMA_ConfigTransfer
- CTR1 DINC LL_DMA_ConfigTransfer
- CTR1 SAP LL_DMA_ConfigTransfer
- CTR1 SBX LL_DMA_ConfigTransfer
- CTR1 PAM LL_DMA_ConfigTransfer
- CTR1 SINC LL_DMA_ConfigTransfer

LL_DMA_ConfigBurstLength

Function name

__STATIC_INLINE void LL_DMA_ConfigBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength, uint32_t DestBurstLength)

Function description

Configure source and destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcBurstLength:** Between 1 to 64
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength
-
- CTR1 SBL_1 LL_DMA_SetDestBurstLength

LL_DMA_ConfigChannelSecure

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all secure parameters linked to DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_CHANNEL_NSEC or LL_DMA_CHANNEL_SEC
  - LL_DMA_CHANNEL_SRC_NSEC or LL_DMA_CHANNEL_SRC_SEC
  - LL_DMA_CHANNEL_DEST_NSEC or LL_DMA_CHANNEL_DEST_SEC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SEC LL_DMA_ConfigChannelSecure
-
- CTR1 SSEC LL_DMA_ConfigChannelSecure
-
- CTR1 DSEC LL_DMA_ConfigChannelSecure

LL_DMA_EnableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_EnableChannelDestSecure

LL_DMA_DisableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_DisableChannelDestSecure

LL_DMA_IsEnabledChannelDestSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_IsEnabledChannelDestSecure

LL_DMA_EnableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_EnableChannelSrcSecure

LL_DMA_DisableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_DisableChannelSrcSecure

LL_DMA_IsEnabledChannelSrcSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_IsEnabledChannelSrcSecure

LL_DMA_SetDestAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAllocatedPort)

Function description

Set destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0
  - LL_DMA_DEST_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_SetDestAllocatedPort

LL_DMA_GetDestAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0
  - LL_DMA_DEST_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_GetDestAllocatedPort

LL_DMA_SetDestHWordExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestHWordExchange)

Function description

Set destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestHWordExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_SetDestHWordExchange

LL_DMA_GetDestHWordExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_GetDestHWordExchange

LL_DMA_SetDestByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestByteExchange)

Function description

Set destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestByteExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_BYTE_PRESERVE
  - LL_DMA_DEST_BYTE_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_SetDestByteExchange

LL_DMA_GetDestByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_BYTE_PRESERVE
  - LL_DMA_DEST_BYTE_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_GetDestByteExchange

LL_DMA_SetSrcByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcByteExchange)

Function description

Set source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcByteExchange:** This parameter can be one of the following values:
  - LL_DMA_SRC_BYTE_PRESERVE
  - LL_DMA_SRC_BYTE_EXCHANGE

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_SetSrcByteExchange

LL_DMA_GetSrcByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_BYTE_PRESERVE
  - LL_DMA_SRC_BYTE_EXCHANGE

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_GetSrcByteExchange

LL_DMA_SetDestBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestBurstLength)

Function description

Set destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength

LL_DMA_GetDestBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 1 to 64.

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_GetDestBurstLength

LL_DMA_SetDestIncMode

Function name

__STATIC_INLINE void LL_DMA_SetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestInc)

Function description

Set destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestInc:** This parameter can be one of the following values:
  - LL_DMA_DEST_FIXED
  - LL_DMA_DEST_INCREMENT

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_SetDestIncMode

LL_DMA_GetDestIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_FIXED
  - LL_DMA_DEST_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_GetDestIncMode

LL_DMA_SetDestDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestDataWidth)

Function description

Set destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestDataWidth:** This parameter can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_SetDestDataWidth

LL_DMA_GetDestDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_GetDestDataWidth

LL_DMA_SetSrcAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAllocatedPort)

Function description

Set source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_SetSrcAllocatedPort

LL_DMA_GetSrcAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_GetSrcAllocatedPort

LL_DMA_SetDataAlignment

Function name

__STATIC_INLINE void LL_DMA_SetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DataAlignment)

Function description

Set data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DataAlignment:** This parameter can be one of the following values:
  - LL_DMA_DATA_ALIGN_ZEROPADD
  - LL_DMA_DATA_ALIGN_SIGNEXTPADD
  - LL_DMA_DATA_PACK_UNPACK

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_SetDataAlignment

LL_DMA_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DATA_ALIGN_ZEROPADD
  - LL_DMA_DATA_ALIGN_SIGNEXTPADD
  - LL_DMA_DATA_PACK_UNPACK

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_GetDataAlignment

LL_DMA_SetSrcBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength)

Function description

Set source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcBurstLength:** Between 1 to 64

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_SetSrcBurstLength

LL_DMA_GetSrcBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 1 to 64
- **None.:**

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_GetSrcBurstLength

LL_DMA_SetSrcIncMode

Function name

__STATIC_INLINE void LL_DMA_SetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcInc)

Function description

Set source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcInc:** This parameter can be one of the following values:
  - LL_DMA_SRC_FIXED
  - LL_DMA_SRC_INCREMENT

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_SetSrcIncMode

LL_DMA_GetSrcIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_FIXED
  - LL_DMA_SRC_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_GetSrcIncMode

LL_DMA_SetSrcDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcDataWidth)

Function description

Set source data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcDataWidth:** This parameter can be one of the following values:
  - LL_DMA_SRC_DATAWIDTH_BYTE
  - LL_DMA_SRC_DATAWIDTH_HALFWORD
  - LL_DMA_SRC_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_SetSrcDataWidth

LL_DMA_GetSrcDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Source Data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_DATAWIDTH_BYTE
  - LL_DMA_SRC_DATAWIDTH_HALFWORD
  - LL_DMA_SRC_DATAWIDTH_WORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_GetSrcDataWidth

LL_DMA_ConfigChannelTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure channel transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_TCEM_BLK_TRANSFER or LL_DMA_TCEM_RPT_BLK_TRANSFER or LL_DMA_TCEM_EACH_LLITEM_TRANSFER or LL_DMA_TCEM_LAST_LLITEM_TRANSFER
  - LL_DMA_TRIG_POLARITY_MASKED or LL_DMA_HWREQUEST_BLK
  - LL_DMA_HWREQUEST_SINGLEBURST or LL_DMA_TRIG_POLARITY_RISING or LL_DMA_TRIG_POLARITY_FALLING
  - LL_DMA_TRIGM_BLK_TRANSFER or LL_DMA_TRIGM_RPT_BLK_TRANSFER or LL_DMA_TRIGM_LLI_LINK_TRANSFER or LL_DMA_TRIGM_SINGLBURST_TRANSFER
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_ConfigChannelTransfer
- CTR2 TRIGPOL LL_DMA_ConfigChannelTransfer
- CTR2 TRIGM LL_DMA_ConfigChannelTransfer
- CTR2 BREQ LL_DMA_ConfigChannelTransfer
- CTR2 DREQ LL_DMA_ConfigChannelTransfer
- CTR2 SWREQ LL_DMA_ConfigChannelTransfer

LL_DMA_SetTransferEventMode

Function name

__STATIC_INLINE void LL_DMA_SetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TransferEventMode)

Function description

Set transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **TransferEventMode:** This parameter can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_SetTransferEventMode

LL_DMA_GetTransferEventMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_GetTransferEventMode

LL_DMA_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_DMA_SetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerPolarity)

Function description

Set trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **TriggerPolarity:** This parameter can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_SetTriggerPolarity

LL_DMA_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_GetTriggerPolarity

LL_DMA_SetTriggerMode

Function name

__STATIC_INLINE void LL_DMA_SetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerMode)

Function description

Set trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **TriggerMode:** This parameter can be one of the following values:
  - LL_DMA_TRIGM_BLK_TRANSFER
  - LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  - LL_DMA_TRIGM_LLI_LINK_TRANSFER
  - LL_DMA_TRIGM_SINGLBURST_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_SetTriggerMode

LL_DMA_GetTriggerMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIGM_BLK_TRANSFER
  - LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  - LL_DMA_TRIGM_LLI_LINK_TRANSFER
  - LL_DMA_TRIGM_SINGLBURST_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_GetTriggerMode

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description

Set destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_SetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_GetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_GetDataTransferDirection

LL_DMA_SetBlkHWRequest

Function name

__STATIC_INLINE void LL_DMA_SetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkHWRequest)

Function description

Set block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkHWRequest:** This parameter can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_SetBlkHWRequest
-

LL_DMA_GetBlkHWRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_GetBlkHWRequest
-

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Request)

Function description

Set hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

- **Request:** This parameter can be one of the following values:
  - LL_GPDMA1_REQUEST_ADC1
  - LL_GPDMA1_REQUEST_ADC2 (*)
  - LL_GPDMA1_REQUEST_DAC1_CH1
  - LL_GPDMA1_REQUEST_DAC1_CH2
  - LL_GPDMA1_REQUEST_TIM6_UP
  - LL_GPDMA1_REQUEST_TIM7_UP
  - LL_GPDMA1_REQUEST_SPI1_RX
  - LL_GPDMA1_REQUEST_SPI1_TX
  - LL_GPDMA1_REQUEST_SPI2_RX
  - LL_GPDMA1_REQUEST_SPI2_TX
  - LL_GPDMA1_REQUEST_SPI3_RX
  - LL_GPDMA1_REQUEST_SPI3_TX
  - LL_GPDMA1_REQUEST_I2C1_RX
  - LL_GPDMA1_REQUEST_I2C1_TX
  - LL_GPDMA1_REQUEST_I2C2_RX
  - LL_GPDMA1_REQUEST_I2C2_TX
  - LL_GPDMA1_REQUEST_I2C3_RX (*)
  - LL_GPDMA1_REQUEST_I2C3_TX (*)
  - LL_GPDMA1_REQUEST_USART1_RX
  - LL_GPDMA1_REQUEST_USART1_TX
  - LL_GPDMA1_REQUEST_USART2_RX
  - LL_GPDMA1_REQUEST_USART2_TX
  - LL_GPDMA1_REQUEST_USART3_RX
  - LL_GPDMA1_REQUEST_USART3_TX
  - LL_GPDMA1_REQUEST_UART4_RX (*)
  - LL_GPDMA1_REQUEST_UART4_TX (*)
  - LL_GPDMA1_REQUEST_UART5_RX (*)
  - LL_GPDMA1_REQUEST_UART5_TX (*)
  - LL_GPDMA1_REQUEST_USART6_RX (*)
  - LL_GPDMA1_REQUEST_USART6_TX (*)
  - LL_GPDMA1_REQUEST_UART7_RX (*)
  - LL_GPDMA1_REQUEST_UART7_TX (*)
  - LL_GPDMA1_REQUEST_UART8_RX (*)
  - LL_GPDMA1_REQUEST_UART8_TX (*)
  - LL_GPDMA1_REQUEST_UART9_RX (*)
  - LL_GPDMA1_REQUEST_UART9_TX (*)
  - LL_GPDMA1_REQUEST_USART10_RX (*)
  - LL_GPDMA1_REQUEST_USART10_TX (*)
  - LL_GPDMA1_REQUEST_USART11_RX (*)
  - LL_GPDMA1_REQUEST_USART11_TX (*)
  - LL_GPDMA1_REQUEST_UART12_RX (*)
  - LL_GPDMA1_REQUEST_UART12_TX (*)
  - LL_GPDMA1_REQUEST_LPUART1_RX
  - LL_GPDMA1_REQUEST_LPUART1_TX
  - LL_GPDMA1_REQUEST_SPI4_RX (*)
  - LL_GPDMA1_REQUEST_SPI4_TX (*)
  - LL_GPDMA1_REQUEST_SPI5_RX (*)
  - LL_GPDMA1_REQUEST_SPI5_TX (*)
  - LL_GPDMA1_REQUEST_SPI6_RX (*)

- LL_GPDMA1_REQUEST_SPI6_TX (*)
- LL_GPDMA1_REQUEST_SAI1_A (*)
- LL_GPDMA1_REQUEST_SAI1_B (*)
- LL_GPDMA1_REQUEST_SAI2_A (*)
- LL_GPDMA1_REQUEST_SAI2_B (*)
- LL_GPDMA1_REQUEST_OCTOSPI1 (*)
- LL_GPDMA1_REQUEST_TIM1_CH1
- LL_GPDMA1_REQUEST_TIM1_CH2
- LL_GPDMA1_REQUEST_TIM1_CH3
- LL_GPDMA1_REQUEST_TIM1_CH4
- LL_GPDMA1_REQUEST_TIM1_UP
- LL_GPDMA1_REQUEST_TIM1_TRIG
- LL_GPDMA1_REQUEST_TIM1_COM
- LL_GPDMA1_REQUEST_TIM8_CH1 (*)
- LL_GPDMA1_REQUEST_TIM8_CH2 (*)
- LL_GPDMA1_REQUEST_TIM8_CH3 (*)
- LL_GPDMA1_REQUEST_TIM8_CH4 (*)
- LL_GPDMA1_REQUEST_TIM8_UP (*)
- LL_GPDMA1_REQUEST_TIM8_TRIG (*)
- LL_GPDMA1_REQUEST_TIM8_COM (*)
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1 (*)
- LL_GPDMA1_REQUEST_TIM4_CH2 (*)
- LL_GPDMA1_REQUEST_TIM4_CH3 (*)
- LL_GPDMA1_REQUEST_TIM4_CH4 (*)
- LL_GPDMA1_REQUEST_TIM4_UP (*)
- LL_GPDMA1_REQUEST_TIM5_CH1 (*)
- LL_GPDMA1_REQUEST_TIM5_CH2 (*)
- LL_GPDMA1_REQUEST_TIM5_CH3 (*)
- LL_GPDMA1_REQUEST_TIM5_CH4 (*)
- LL_GPDMA1_REQUEST_TIM5_UP (*)
- LL_GPDMA1_REQUEST_TIM5_TRIG (*)
- LL_GPDMA1_REQUEST_TIM15_CH1 (*)
- LL_GPDMA1_REQUEST_TIM15_UP (*)
- LL_GPDMA1_REQUEST_TIM15_TRIG (*)
- LL_GPDMA1_REQUEST_TIM15_COM (*)
- LL_GPDMA1_REQUEST_TIM16_CH1 (*)
- LL_GPDMA1_REQUEST_TIM16_UP (*)
- LL_GPDMA1_REQUEST_TIM17_CH1 (*)
- LL_GPDMA1_REQUEST_TIM17_UP (*)

- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_DCMI (*)
- LL_GPDMA1_REQUEST_AES_OUT (*)
- LL_GPDMA1_REQUEST_AES_IN (*)
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_UCPD1_RX (*)
- LL_GPDMA1_REQUEST_UCPD1_TX (*)
- LL_GPDMA1_REQUEST_CORDIC_READ (*)
- LL_GPDMA1_REQUEST_CORDIC_WRITE (*)
- LL_GPDMA1_REQUEST_FMAC_READ (*)
- LL_GPDMA1_REQUEST_FMAC_WRITE (*)
- LL_GPDMA1_REQUEST_SAES_OUT (*)
- LL_GPDMA1_REQUEST_SAES_IN (*)
- LL_GPDMA1_REQUEST_I3C1_RX
- LL_GPDMA1_REQUEST_I3C1_TX
- LL_GPDMA1_REQUEST_I3C1_TC
- LL_GPDMA1_REQUEST_I3C1_RS
- LL_GPDMA1_REQUEST_I2C4_RX (*)
- LL_GPDMA1_REQUEST_I2C4_TX (*)
- LL_GPDMA1_REQUEST_LPTIM3_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM3_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM3_UE (*)
- LL_GPDMA1_REQUEST_LPTIM5_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM5_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM5_UE (*)
- LL_GPDMA1_REQUEST_LPTIM6_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM6_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM6_UE (*)
- LL_GPDMA1_REQUEST_I3C2_RX (*)
- LL_GPDMA1_REQUEST_I3C2_TX (*)
- LL_GPDMA1_REQUEST_I3C2_TC (*)
- LL_GPDMA1_REQUEST_I3C2_RS (*)
- LL_GPDMA2_REQUEST_ADC1
- LL_GPDMA2_REQUEST_ADC2 (*)
- LL_GPDMA2_REQUEST_DAC1_CH1
- LL_GPDMA2_REQUEST_DAC1_CH2
- LL_GPDMA2_REQUEST_TIM6_UP
- LL_GPDMA2_REQUEST_TIM7_UP
- LL_GPDMA2_REQUEST_SPI1_RX
- LL_GPDMA2_REQUEST_SPI1_TX
- LL_GPDMA2_REQUEST_SPI2_RX
- LL_GPDMA2_REQUEST_SPI2_TX
- LL_GPDMA2_REQUEST_SPI3_RX
- LL_GPDMA2_REQUEST_SPI3_TX
- LL_GPDMA2_REQUEST_I2C1_RX

- LL_GPDMA2_REQUEST_I2C1_TX
- LL_GPDMA2_REQUEST_I2C2_RX
- LL_GPDMA2_REQUEST_I2C2_TX
- LL_GPDMA2_REQUEST_I2C3_RX (*)
- LL_GPDMA2_REQUEST_I2C3_TX (*)
- LL_GPDMA2_REQUEST_USART1_RX
- LL_GPDMA2_REQUEST_USART1_TX
- LL_GPDMA2_REQUEST_USART2_RX
- LL_GPDMA2_REQUEST_USART2_TX
- LL_GPDMA2_REQUEST_USART3_RX
- LL_GPDMA2_REQUEST_USART3_TX
- LL_GPDMA2_REQUEST_UART4_RX (*)
- LL_GPDMA2_REQUEST_UART4_TX (*)
- LL_GPDMA2_REQUEST_UART5_RX (*)
- LL_GPDMA2_REQUEST_UART5_TX (*)
- LL_GPDMA2_REQUEST_USART6_RX (*)
- LL_GPDMA2_REQUEST_USART6_TX (*)
- LL_GPDMA2_REQUEST_UART7_RX (*)
- LL_GPDMA2_REQUEST_UART7_TX (*)
- LL_GPDMA2_REQUEST_UART8_RX (*)
- LL_GPDMA2_REQUEST_UART8_TX (*)
- LL_GPDMA2_REQUEST_UART9_RX (*)
- LL_GPDMA2_REQUEST_UART9_TX (*)
- LL_GPDMA2_REQUEST_USART10_RX (*)
- LL_GPDMA2_REQUEST_USART10_TX (*)
- LL_GPDMA2_REQUEST_USART11_RX (*)
- LL_GPDMA2_REQUEST_USART11_TX (*)
- LL_GPDMA2_REQUEST_UART12_RX (*)
- LL_GPDMA2_REQUEST_UART12_TX (*)
- LL_GPDMA2_REQUEST_LPUART1_RX
- LL_GPDMA2_REQUEST_LPUART1_TX
- LL_GPDMA2_REQUEST_SPI4_RX (*)
- LL_GPDMA2_REQUEST_SPI4_TX (*)
- LL_GPDMA2_REQUEST_SPI5_RX (*)
- LL_GPDMA2_REQUEST_SPI5_TX (*)
- LL_GPDMA2_REQUEST_SPI6_RX (*)
- LL_GPDMA2_REQUEST_SPI6_TX (*)
- LL_GPDMA2_REQUEST_SAI1_A (*)
- LL_GPDMA2_REQUEST_SAI1_B (*)
- LL_GPDMA2_REQUEST_SAI2_A (*)
- LL_GPDMA2_REQUEST_SAI2_B (*)
- LL_GPDMA2_REQUEST_OCTOSPI1 (*)
- LL_GPDMA2_REQUEST_TIM1_CH1
- LL_GPDMA2_REQUEST_TIM1_CH2
- LL_GPDMA2_REQUEST_TIM1_CH3
- LL_GPDMA2_REQUEST_TIM1_CH4
- LL_GPDMA2_REQUEST_TIM1_UP
- LL_GPDMA2_REQUEST_TIM1_TRIG
- LL_GPDMA2_REQUEST_TIM1_COM
- LL_GPDMA2_REQUEST_TIM8_CH1 (*)

- LL_GPDMA2_REQUEST_TIM8_CH2 (*)
- LL_GPDMA2_REQUEST_TIM8_CH3 (*)
- LL_GPDMA2_REQUEST_TIM8_CH4 (*)
- LL_GPDMA2_REQUEST_TIM8_UP (*)
- LL_GPDMA2_REQUEST_TIM8_TRIG (*)
- LL_GPDMA2_REQUEST_TIM8_COM (*)
- LL_GPDMA2_REQUEST_TIM2_CH1
- LL_GPDMA2_REQUEST_TIM2_CH2
- LL_GPDMA2_REQUEST_TIM2_CH3
- LL_GPDMA2_REQUEST_TIM2_CH4
- LL_GPDMA2_REQUEST_TIM2_UP
- LL_GPDMA2_REQUEST_TIM3_CH1
- LL_GPDMA2_REQUEST_TIM3_CH2
- LL_GPDMA2_REQUEST_TIM3_CH3
- LL_GPDMA2_REQUEST_TIM3_CH4
- LL_GPDMA2_REQUEST_TIM3_UP
- LL_GPDMA2_REQUEST_TIM3_TRIG
- LL_GPDMA2_REQUEST_TIM4_CH1 (*)
- LL_GPDMA2_REQUEST_TIM4_CH2 (*)
- LL_GPDMA2_REQUEST_TIM4_CH3 (*)
- LL_GPDMA2_REQUEST_TIM4_CH4 (*)
- LL_GPDMA2_REQUEST_TIM4_UP (*)
- LL_GPDMA2_REQUEST_TIM5_CH1 (*)
- LL_GPDMA2_REQUEST_TIM5_CH2 (*)
- LL_GPDMA2_REQUEST_TIM5_CH3 (*)
- LL_GPDMA2_REQUEST_TIM5_CH4 (*)
- LL_GPDMA2_REQUEST_TIM5_UP (*)
- LL_GPDMA2_REQUEST_TIM5_TRIG (*)
- LL_GPDMA2_REQUEST_TIM15_CH1 (*)
- LL_GPDMA2_REQUEST_TIM15_UP (*)
- LL_GPDMA2_REQUEST_TIM15_TRIG (*)
- LL_GPDMA2_REQUEST_TIM15_COM (*)
- LL_GPDMA2_REQUEST_TIM16_CH1 (*)
- LL_GPDMA2_REQUEST_TIM16_UP (*)
- LL_GPDMA2_REQUEST_TIM17_CH1 (*)
- LL_GPDMA2_REQUEST_TIM17_UP (*)
- LL_GPDMA2_REQUEST_LPTIM1_IC1
- LL_GPDMA2_REQUEST_LPTIM1_IC2
- LL_GPDMA2_REQUEST_LPTIM1_UE
- LL_GPDMA2_REQUEST_LPTIM2_IC1
- LL_GPDMA2_REQUEST_LPTIM2_IC2
- LL_GPDMA2_REQUEST_LPTIM2_UE
- LL_GPDMA2_REQUEST_DCMI (*)
- LL_GPDMA2_REQUEST_AES_OUT (*)
- LL_GPDMA2_REQUEST_AES_IN (*)
- LL_GPDMA2_REQUEST_HASH_IN
- LL_GPDMA2_REQUEST_UCPD1_RX (*)
- LL_GPDMA2_REQUEST_UCPD1_TX (*)
- LL_GPDMA2_REQUEST_CORDIC_READ (*)
- LL_GPDMA2_REQUEST_CORDIC_WRITE (*)

- LL_GPDMA2_REQUEST_FMAC_READ (*)
- LL_GPDMA2_REQUEST_FMAC_WRITE (*)
- LL_GPDMA2_REQUEST_SAES_OUT (*)
- LL_GPDMA2_REQUEST_SAES_IN (*)
- LL_GPDMA2_REQUEST_I3C1_RX
- LL_GPDMA2_REQUEST_I3C1_TX
- LL_GPDMA2_REQUEST_I3C1_TC
- LL_GPDMA2_REQUEST_I3C1_RS
- LL_GPDMA2_REQUEST_I2C4_RX (*)
- LL_GPDMA2_REQUEST_I2C4_TX (*)
- LL_GPDMA2_REQUEST_LPTIM3_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM3_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM3_UE (*)
- LL_GPDMA2_REQUEST_LPTIM5_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM5_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM5_UE (*)
- LL_GPDMA2_REQUEST_LPTIM6_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM6_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM6_UE (*)
- LL_GPDMA2_REQUEST_I3C2_RX (*)
- LL_GPDMA2_REQUEST_I3C2_TX (*)
- LL_GPDMA2_REQUEST_I3C2_TC (*)
- LL_GPDMA2_REQUEST_I3C2_RS (*)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPDMA1_REQUEST_ADC1
  - LL_GPDMA1_REQUEST_ADC2 (*)
  - LL_GPDMA1_REQUEST_DAC1_CH1
  - LL_GPDMA1_REQUEST_DAC1_CH2
  - LL_GPDMA1_REQUEST_TIM6_UP
  - LL_GPDMA1_REQUEST_TIM7_UP
  - LL_GPDMA1_REQUEST_SPI1_RX
  - LL_GPDMA1_REQUEST_SPI1_TX
  - LL_GPDMA1_REQUEST_SPI2_RX
  - LL_GPDMA1_REQUEST_SPI2_TX
  - LL_GPDMA1_REQUEST_SPI3_RX
  - LL_GPDMA1_REQUEST_SPI3_TX
  - LL_GPDMA1_REQUEST_I2C1_RX
  - LL_GPDMA1_REQUEST_I2C1_TX
  - LL_GPDMA1_REQUEST_I2C2_RX
  - LL_GPDMA1_REQUEST_I2C2_TX
  - LL_GPDMA1_REQUEST_I2C3_RX (*)
  - LL_GPDMA1_REQUEST_I2C3_TX (*)
  - LL_GPDMA1_REQUEST_USART1_RX
  - LL_GPDMA1_REQUEST_USART1_TX
  - LL_GPDMA1_REQUEST_USART2_RX
  - LL_GPDMA1_REQUEST_USART2_TX
  - LL_GPDMA1_REQUEST_USART3_RX
  - LL_GPDMA1_REQUEST_USART3_TX
  - LL_GPDMA1_REQUEST_UART4_RX (*)
  - LL_GPDMA1_REQUEST_UART4_TX (*)
  - LL_GPDMA1_REQUEST_UART5_RX (*)
  - LL_GPDMA1_REQUEST_UART5_TX (*)
  - LL_GPDMA1_REQUEST_USART6_RX (*)
  - LL_GPDMA1_REQUEST_USART6_TX (*)
  - LL_GPDMA1_REQUEST_UART7_RX (*)
  - LL_GPDMA1_REQUEST_UART7_TX (*)
  - LL_GPDMA1_REQUEST_UART8_RX (*)
  - LL_GPDMA1_REQUEST_UART8_TX (*)
  - LL_GPDMA1_REQUEST_UART9_RX (*)
  - LL_GPDMA1_REQUEST_UART9_TX (*)
  - LL_GPDMA1_REQUEST_USART10_RX (*)
  - LL_GPDMA1_REQUEST_USART10_TX (*)
  - LL_GPDMA1_REQUEST_USART11_RX (*)
  - LL_GPDMA1_REQUEST_USART11_TX (*)
  - LL_GPDMA1_REQUEST_UART12_RX (*)
  - LL_GPDMA1_REQUEST_UART12_TX (*)
  - LL_GPDMA1_REQUEST_LPUART1_RX
  - LL_GPDMA1_REQUEST_LPUART1_TX
  - LL_GPDMA1_REQUEST_SPI4_RX (*)
  - LL_GPDMA1_REQUEST_SPI4_TX (*)
  - LL_GPDMA1_REQUEST_SPI5_RX (*)
  - LL_GPDMA1_REQUEST_SPI5_TX (*)

- LL_GPDMA1_REQUEST_SPI6_RX (*)
- LL_GPDMA1_REQUEST_SPI6_TX (*)
- LL_GPDMA1_REQUEST_SAI1_A (*)
- LL_GPDMA1_REQUEST_SAI1_B (*)
- LL_GPDMA1_REQUEST_SAI2_A (*)
- LL_GPDMA1_REQUEST_SAI2_B (*)
- LL_GPDMA1_REQUEST_OCTOSPI1 (*)
- LL_GPDMA1_REQUEST_TIM1_CH1
- LL_GPDMA1_REQUEST_TIM1_CH2
- LL_GPDMA1_REQUEST_TIM1_CH3
- LL_GPDMA1_REQUEST_TIM1_CH4
- LL_GPDMA1_REQUEST_TIM1_UP
- LL_GPDMA1_REQUEST_TIM1_TRIG
- LL_GPDMA1_REQUEST_TIM1_COM
- LL_GPDMA1_REQUEST_TIM8_CH1 (*)
- LL_GPDMA1_REQUEST_TIM8_CH2 (*)
- LL_GPDMA1_REQUEST_TIM8_CH3 (*)
- LL_GPDMA1_REQUEST_TIM8_CH4 (*)
- LL_GPDMA1_REQUEST_TIM8_UP (*)
- LL_GPDMA1_REQUEST_TIM8_TRIG (*)
- LL_GPDMA1_REQUEST_TIM8_COM (*)
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1 (*)
- LL_GPDMA1_REQUEST_TIM4_CH2 (*)
- LL_GPDMA1_REQUEST_TIM4_CH3 (*)
- LL_GPDMA1_REQUEST_TIM4_CH4 (*)
- LL_GPDMA1_REQUEST_TIM4_UP (*)
- LL_GPDMA1_REQUEST_TIM5_CH1 (*)
- LL_GPDMA1_REQUEST_TIM5_CH2 (*)
- LL_GPDMA1_REQUEST_TIM5_CH3 (*)
- LL_GPDMA1_REQUEST_TIM5_CH4 (*)
- LL_GPDMA1_REQUEST_TIM5_UP (*)
- LL_GPDMA1_REQUEST_TIM5_TRIG (*)
- LL_GPDMA1_REQUEST_TIM15_CH1 (*)
- LL_GPDMA1_REQUEST_TIM15_UP (*)
- LL_GPDMA1_REQUEST_TIM15_TRIG (*)
- LL_GPDMA1_REQUEST_TIM15_COM (*)
- LL_GPDMA1_REQUEST_TIM16_CH1 (*)
- LL_GPDMA1_REQUEST_TIM16_UP (*)
- LL_GPDMA1_REQUEST_TIM17_CH1 (*)

- LL_GPDMA1_REQUEST_TIM17_UP (*)
- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_DCMI (*)
- LL_GPDMA1_REQUEST_AES_OUT (*)
- LL_GPDMA1_REQUEST_AES_IN (*)
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_UCPD1_RX (*)
- LL_GPDMA1_REQUEST_UCPD1_TX (*)
- LL_GPDMA1_REQUEST_CORDIC_READ (*)
- LL_GPDMA1_REQUEST_CORDIC_WRITE (*)
- LL_GPDMA1_REQUEST_FMAC_READ (*)
- LL_GPDMA1_REQUEST_FMAC_WRITE (*)
- LL_GPDMA1_REQUEST_SAES_OUT (*)
- LL_GPDMA1_REQUEST_SAES_IN (*)
- LL_GPDMA1_REQUEST_I3C1_RX
- LL_GPDMA1_REQUEST_I3C1_TX
- LL_GPDMA1_REQUEST_I3C1_TC
- LL_GPDMA1_REQUEST_I3C1_RS
- LL_GPDMA1_REQUEST_I2C4_RX (*)
- LL_GPDMA1_REQUEST_I2C4_TX (*)
- LL_GPDMA1_REQUEST_LPTIM3_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM3_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM3_UE (*)
- LL_GPDMA1_REQUEST_LPTIM5_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM5_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM5_UE (*)
- LL_GPDMA1_REQUEST_LPTIM6_IC1 (*)
- LL_GPDMA1_REQUEST_LPTIM6_IC2 (*)
- LL_GPDMA1_REQUEST_LPTIM6_UE (*)
- LL_GPDMA1_REQUEST_I3C2_RX (*)
- LL_GPDMA1_REQUEST_I3C2_TX (*)
- LL_GPDMA1_REQUEST_I3C2_TC (*)
- LL_GPDMA1_REQUEST_I3C2_RS (*)
- LL_GPDMA2_REQUEST_ADC1
- LL_GPDMA2_REQUEST_ADC2 (*)
- LL_GPDMA2_REQUEST_DAC1_CH1
- LL_GPDMA2_REQUEST_DAC1_CH2
- LL_GPDMA2_REQUEST_TIM6_UP
- LL_GPDMA2_REQUEST_TIM7_UP
- LL_GPDMA2_REQUEST_SPI1_RX
- LL_GPDMA2_REQUEST_SPI1_TX
- LL_GPDMA2_REQUEST_SPI2_RX
- LL_GPDMA2_REQUEST_SPI2_TX
- LL_GPDMA2_REQUEST_SPI3_RX
- LL_GPDMA2_REQUEST_SPI3_TX

- LL_GPDMA2_REQUEST_I2C1_RX
- LL_GPDMA2_REQUEST_I2C1_TX
- LL_GPDMA2_REQUEST_I2C2_RX
- LL_GPDMA2_REQUEST_I2C2_TX
- LL_GPDMA2_REQUEST_I2C3_RX (*)
- LL_GPDMA2_REQUEST_I2C3_TX (*)
- LL_GPDMA2_REQUEST_USART1_RX
- LL_GPDMA2_REQUEST_USART1_TX
- LL_GPDMA2_REQUEST_USART2_RX
- LL_GPDMA2_REQUEST_USART2_TX
- LL_GPDMA2_REQUEST_USART3_RX
- LL_GPDMA2_REQUEST_USART3_TX
- LL_GPDMA2_REQUEST_UART4_RX (*)
- LL_GPDMA2_REQUEST_UART4_TX (*)
- LL_GPDMA2_REQUEST_UART5_RX (*)
- LL_GPDMA2_REQUEST_UART5_TX (*)
- LL_GPDMA2_REQUEST_USART6_RX (*)
- LL_GPDMA2_REQUEST_USART6_TX (*)
- LL_GPDMA2_REQUEST_UART7_RX (*)
- LL_GPDMA2_REQUEST_UART7_TX (*)
- LL_GPDMA2_REQUEST_UART8_RX (*)
- LL_GPDMA2_REQUEST_UART8_TX (*)
- LL_GPDMA2_REQUEST_UART9_RX (*)
- LL_GPDMA2_REQUEST_UART9_TX (*)
- LL_GPDMA2_REQUEST_USART10_RX (*)
- LL_GPDMA2_REQUEST_USART10_TX (*)
- LL_GPDMA2_REQUEST_USART11_RX (*)
- LL_GPDMA2_REQUEST_USART11_TX (*)
- LL_GPDMA2_REQUEST_UART12_RX (*)
- LL_GPDMA2_REQUEST_UART12_TX (*)
- LL_GPDMA2_REQUEST_LPUART1_RX
- LL_GPDMA2_REQUEST_LPUART1_TX
- LL_GPDMA2_REQUEST_SPI4_RX (*)
- LL_GPDMA2_REQUEST_SPI4_TX (*)
- LL_GPDMA2_REQUEST_SPI5_RX (*)
- LL_GPDMA2_REQUEST_SPI5_TX (*)
- LL_GPDMA2_REQUEST_SPI6_RX (*)
- LL_GPDMA2_REQUEST_SPI6_TX (*)
- LL_GPDMA2_REQUEST_SAI1_A (*)
- LL_GPDMA2_REQUEST_SAI1_B (*)
- LL_GPDMA2_REQUEST_SAI2_A (*)
- LL_GPDMA2_REQUEST_SAI2_B (*)
- LL_GPDMA2_REQUEST_OCTOSPI1 (*)
- LL_GPDMA2_REQUEST_TIM1_CH1
- LL_GPDMA2_REQUEST_TIM1_CH2
- LL_GPDMA2_REQUEST_TIM1_CH3
- LL_GPDMA2_REQUEST_TIM1_CH4
- LL_GPDMA2_REQUEST_TIM1_UP
- LL_GPDMA2_REQUEST_TIM1_TRIG
- LL_GPDMA2_REQUEST_TIM1_COM

- LL_GPDMA2_REQUEST_TIM8_CH1 (*)
- LL_GPDMA2_REQUEST_TIM8_CH2 (*)
- LL_GPDMA2_REQUEST_TIM8_CH3 (*)
- LL_GPDMA2_REQUEST_TIM8_CH4 (*)
- LL_GPDMA2_REQUEST_TIM8_UP (*)
- LL_GPDMA2_REQUEST_TIM8_TRIG (*)
- LL_GPDMA2_REQUEST_TIM8_COM (*)
- LL_GPDMA2_REQUEST_TIM2_CH1
- LL_GPDMA2_REQUEST_TIM2_CH2
- LL_GPDMA2_REQUEST_TIM2_CH3
- LL_GPDMA2_REQUEST_TIM2_CH4
- LL_GPDMA2_REQUEST_TIM2_UP
- LL_GPDMA2_REQUEST_TIM3_CH1
- LL_GPDMA2_REQUEST_TIM3_CH2
- LL_GPDMA2_REQUEST_TIM3_CH3
- LL_GPDMA2_REQUEST_TIM3_CH4
- LL_GPDMA2_REQUEST_TIM3_UP
- LL_GPDMA2_REQUEST_TIM3_TRIG
- LL_GPDMA2_REQUEST_TIM4_CH1 (*)
- LL_GPDMA2_REQUEST_TIM4_CH2 (*)
- LL_GPDMA2_REQUEST_TIM4_CH3 (*)
- LL_GPDMA2_REQUEST_TIM4_CH4 (*)
- LL_GPDMA2_REQUEST_TIM4_UP (*)
- LL_GPDMA2_REQUEST_TIM5_CH1 (*)
- LL_GPDMA2_REQUEST_TIM5_CH2 (*)
- LL_GPDMA2_REQUEST_TIM5_CH3 (*)
- LL_GPDMA2_REQUEST_TIM5_CH4 (*)
- LL_GPDMA2_REQUEST_TIM5_UP (*)
- LL_GPDMA2_REQUEST_TIM5_TRIG (*)
- LL_GPDMA2_REQUEST_TIM15_CH1 (*)
- LL_GPDMA2_REQUEST_TIM15_UP (*)
- LL_GPDMA2_REQUEST_TIM15_TRIG (*)
- LL_GPDMA2_REQUEST_TIM15_COM (*)
- LL_GPDMA2_REQUEST_TIM16_CH1 (*)
- LL_GPDMA2_REQUEST_TIM16_UP (*)
- LL_GPDMA2_REQUEST_TIM17_CH1 (*)
- LL_GPDMA2_REQUEST_TIM17_UP (*)
- LL_GPDMA2_REQUEST_LPTIM1_IC1
- LL_GPDMA2_REQUEST_LPTIM1_IC2
- LL_GPDMA2_REQUEST_LPTIM1_UE
- LL_GPDMA2_REQUEST_LPTIM2_IC1
- LL_GPDMA2_REQUEST_LPTIM2_IC2
- LL_GPDMA2_REQUEST_LPTIM2_UE
- LL_GPDMA2_REQUEST_DCMI (*)
- LL_GPDMA2_REQUEST_AES_OUT (*)
- LL_GPDMA2_REQUEST_AES_IN (*)
- LL_GPDMA2_REQUEST_HASH_IN
- LL_GPDMA2_REQUEST_UCPD1_RX (*)
- LL_GPDMA2_REQUEST_UCPD1_TX (*)
- LL_GPDMA2_REQUEST_CORDIC_READ (*)

- LL_GPDMA2_REQUEST_CORDIC_WRITE (*)
- LL_GPDMA2_REQUEST_FMAC_READ (*)
- LL_GPDMA2_REQUEST_FMAC_WRITE (*)
- LL_GPDMA2_REQUEST_SAES_OUT (*)
- LL_GPDMA2_REQUEST_SAES_IN (*)
- LL_GPDMA2_REQUEST_I3C1_RX
- LL_GPDMA2_REQUEST_I3C1_TX
- LL_GPDMA2_REQUEST_I3C1_TC
- LL_GPDMA2_REQUEST_I3C1_RS
- LL_GPDMA2_REQUEST_I2C4_RX (*)
- LL_GPDMA2_REQUEST_I2C4_TX (*)
- LL_GPDMA2_REQUEST_LPTIM3_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM3_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM3_UE (*)
- LL_GPDMA2_REQUEST_LPTIM5_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM5_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM5_UE (*)
- LL_GPDMA2_REQUEST_LPTIM6_IC1 (*)
- LL_GPDMA2_REQUEST_LPTIM6_IC2 (*)
- LL_GPDMA2_REQUEST_LPTIM6_UE (*)
- LL_GPDMA2_REQUEST_I3C2_RX (*)
- LL_GPDMA2_REQUEST_I3C2_TX (*)
- LL_GPDMA2_REQUEST_I3C2_TC (*)
- LL_GPDMA2_REQUEST_I3C2_RS (*)

Notes

- This API is used for all available DMA channels.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_GetPeriphRequest

LL_DMA_SetHWTrigger

Function name

__STATIC_INLINE void LL_DMA_SetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Trigger)

Function description

Set hardware trigger.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

- **Trigger:** This parameter can be one of the following values:
  - LL_GPDMA1_TRIGGER_EXTI_LINE0
  - LL_GPDMA1_TRIGGER_EXTI_LINE1
  - LL_GPDMA1_TRIGGER_EXTI_LINE2
  - LL_GPDMA1_TRIGGER_EXTI_LINE3
  - LL_GPDMA1_TRIGGER_EXTI_LINE4
  - LL_GPDMA1_TRIGGER_EXTI_LINE5
  - LL_GPDMA1_TRIGGER_EXTI_LINE6
  - LL_GPDMA1_TRIGGER_EXTI_LINE7
  - LL_GPDMA1_TRIGGER_TAMP_TRG1
  - LL_GPDMA1_TRIGGER_TAMP_TRG2
  - LL_GPDMA1_TRIGGER_TAMP_TRG3 (*)
  - LL_GPDMA1_TRIGGER_LPTIM1_CH1
  - LL_GPDMA1_TRIGGER_LPTIM1_CH2
  - LL_GPDMA1_TRIGGER_LPTIM2_CH1
  - LL_GPDMA1_TRIGGER_LPTIM2_CH2
  - LL_GPDMA1_TRIGGER_RTC_ALRA_TRG
  - LL_GPDMA1_TRIGGER_RTC_ALRB_TRG
  - LL_GPDMA1_TRIGGER_RTC_WUT_TRG
  - LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH7_TCF
  - LL_GPDMA1_TRIGGER_TIM2_TRGO
  - LL_GPDMA1_TRIGGER_TIM15_TRGO
  - LL_GPDMA1_TRIGGER_TIM12_TRGO
  - LL_GPDMA1_TRIGGER_LPTIM3_CH1
  - LL_GPDMA1_TRIGGER_LPTIM3_CH2
  - LL_GPDMA1_TRIGGER_LPTIM4_AIT
  - LL_GPDMA1_TRIGGER_LPTIM5_CH1
  - LL_GPDMA1_TRIGGER_LPTIM5_CH2
  - LL_GPDMA1_TRIGGER_LPTIM6_CH1
  - LL_GPDMA1_TRIGGER_LPTIM6_CH2
  - LL_GPDMA1_TRIGGER_TIM15_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM12_TRGO (*)
  - LL_GPDMA1_TRIGGER_LPTIM3_CH1 (*)
  - LL_GPDMA1_TRIGGER_LPTIM3_CH2 (*)
  - LL_GPDMA1_TRIGGER_LPTIM4_AIT (*)

- LL_GPDMA1_TRIGGER_LPTIM5_CH1 (*)
- LL_GPDMA1_TRIGGER_LPTIM5_CH2 (*)
- LL_GPDMA1_TRIGGER_LPTIM6_CH1 (*)
- LL_GPDMA1_TRIGGER_LPTIM6_CH2 (*)
- LL_GPDMA1_TRIGGER_COMP1_OUT (*)
- LL_GPDMA1_TRIGGER_EVENTOUT (*)
- LL_GPDMA2_TRIGGER_EXTI_LINE0
- LL_GPDMA2_TRIGGER_EXTI_LINE1
- LL_GPDMA2_TRIGGER_EXTI_LINE2
- LL_GPDMA2_TRIGGER_EXTI_LINE3
- LL_GPDMA2_TRIGGER_EXTI_LINE4
- LL_GPDMA2_TRIGGER_EXTI_LINE5
- LL_GPDMA2_TRIGGER_EXTI_LINE6
- LL_GPDMA2_TRIGGER_EXTI_LINE7
- LL_GPDMA2_TRIGGER_TAMP_TRG1
- LL_GPDMA2_TRIGGER_TAMP_TRG2
- LL_GPDMA2_TRIGGER_TAMP_TRG3 (*)
- LL_GPDMA2_TRIGGER_LPTIM1_CH1
- LL_GPDMA2_TRIGGER_LPTIM1_CH2
- LL_GPDMA2_TRIGGER_LPTIM2_CH1
- LL_GPDMA2_TRIGGER_LPTIM2_CH2
- LL_GPDMA2_TRIGGER_RTC_ALRA_TRG
- LL_GPDMA2_TRIGGER_RTC_ALRB_TRG
- LL_GPDMA2_TRIGGER_RTC_WUT_TRG
- LL_GPDMA2_TRIGGER_GPDMA1_CH0_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH1_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH2_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH3_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH4_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH5_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH6_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH7_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH0_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH1_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH2_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH3_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH4_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH5_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH6_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH7_TCF
- LL_GPDMA2_TRIGGER_TIM2_TRGO
- LL_GPDMA2_TRIGGER_TIM15_TRGO
- LL_GPDMA2_TRIGGER_TIM12_TRGO
- LL_GPDMA2_TRIGGER_LPTIM3_CH1
- LL_GPDMA2_TRIGGER_LPTIM3_CH2
- LL_GPDMA2_TRIGGER_LPTIM4_AIT
- LL_GPDMA2_TRIGGER_LPTIM5_CH1
- LL_GPDMA2_TRIGGER_LPTIM5_CH2
- LL_GPDMA2_TRIGGER_LPTIM6_CH1
- LL_GPDMA2_TRIGGER_LPTIM6_CH2

- LL_GPDMA2_TRIGGER_TIM15_TRGO (*)
- LL_GPDMA2_TRIGGER_TIM12_TRGO (*)
- LL_GPDMA2_TRIGGER_LPTIM3_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM3_CH2 (*)
- LL_GPDMA2_TRIGGER_LPTIM4_AIT (*)
- LL_GPDMA2_TRIGGER_LPTIM5_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM5_CH2 (*)
- LL_GPDMA2_TRIGGER_LPTIM6_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM6_CH2 (*)
- LL_GPDMA2_TRIGGER_COMP1_OUT (*)
- LL_GPDMA2_TRIGGER_EVENTOUT (*)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_SetHWTrigger

LL_DMA_GetHWTrigger

Function name

__STATIC_INLINE uint32_t LL_DMA_GetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware triggers.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPDMA1_TRIGGER_EXTI_LINE0
  - LL_GPDMA1_TRIGGER_EXTI_LINE1
  - LL_GPDMA1_TRIGGER_EXTI_LINE2
  - LL_GPDMA1_TRIGGER_EXTI_LINE3
  - LL_GPDMA1_TRIGGER_EXTI_LINE4
  - LL_GPDMA1_TRIGGER_EXTI_LINE5
  - LL_GPDMA1_TRIGGER_EXTI_LINE6
  - LL_GPDMA1_TRIGGER_EXTI_LINE7
  - LL_GPDMA1_TRIGGER_TAMP_TRG1
  - LL_GPDMA1_TRIGGER_TAMP_TRG2
  - LL_GPDMA1_TRIGGER_TAMP_TRG3 (*)
  - LL_GPDMA1_TRIGGER_LPTIM1_CH1
  - LL_GPDMA1_TRIGGER_LPTIM1_CH2
  - LL_GPDMA1_TRIGGER_LPTIM2_CH1
  - LL_GPDMA1_TRIGGER_LPTIM2_CH2
  - LL_GPDMA1_TRIGGER_RTC_ALRA_TRG
  - LL_GPDMA1_TRIGGER_RTC_ALRB_TRG
  - LL_GPDMA1_TRIGGER_RTC_WUT_TRG
  - LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA2_CH7_TCF
  - LL_GPDMA1_TRIGGER_TIM2_TRGO
  - LL_GPDMA1_TRIGGER_TIM15_TRGO
  - LL_GPDMA1_TRIGGER_TIM12_TRGO
  - LL_GPDMA1_TRIGGER_LPTIM3_CH1
  - LL_GPDMA1_TRIGGER_LPTIM3_CH2
  - LL_GPDMA1_TRIGGER_LPTIM4_AIT
  - LL_GPDMA1_TRIGGER_LPTIM5_CH1
  - LL_GPDMA1_TRIGGER_LPTIM5_CH2
  - LL_GPDMA1_TRIGGER_LPTIM6_CH1
  - LL_GPDMA1_TRIGGER_LPTIM6_CH2
  - LL_GPDMA1_TRIGGER_TIM15_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM12_TRGO (*)
  - LL_GPDMA1_TRIGGER_LPTIM3_CH1 (*)
  - LL_GPDMA1_TRIGGER_LPTIM3_CH2 (*)

- LL_GPDMA1_TRIGGER_LPTIM4_AIT (*)
- LL_GPDMA1_TRIGGER_LPTIM5_CH1 (*)
- LL_GPDMA1_TRIGGER_LPTIM5_CH2 (*)
- LL_GPDMA1_TRIGGER_LPTIM6_CH1 (*)
- LL_GPDMA1_TRIGGER_LPTIM6_CH2 (*)
- LL_GPDMA1_TRIGGER_COMP1_OUT (*)
- LL_GPDMA1_TRIGGER_EVENTOUT (*)
- LL_GPDMA2_TRIGGER_EXTI_LINE0
- LL_GPDMA2_TRIGGER_EXTI_LINE1
- LL_GPDMA2_TRIGGER_EXTI_LINE2
- LL_GPDMA2_TRIGGER_EXTI_LINE3
- LL_GPDMA2_TRIGGER_EXTI_LINE4
- LL_GPDMA2_TRIGGER_EXTI_LINE5
- LL_GPDMA2_TRIGGER_EXTI_LINE6
- LL_GPDMA2_TRIGGER_EXTI_LINE7
- LL_GPDMA2_TRIGGER_TAMP_TRG1
- LL_GPDMA2_TRIGGER_TAMP_TRG2
- LL_GPDMA2_TRIGGER_TAMP_TRG3 (*)
- LL_GPDMA2_TRIGGER_LPTIM1_CH1
- LL_GPDMA2_TRIGGER_LPTIM1_CH2
- LL_GPDMA2_TRIGGER_LPTIM2_CH1
- LL_GPDMA2_TRIGGER_LPTIM2_CH2
- LL_GPDMA2_TRIGGER_RTC_ALRA_TRG
- LL_GPDMA2_TRIGGER_RTC_ALRB_TRG
- LL_GPDMA2_TRIGGER_RTC_WUT_TRG
- LL_GPDMA2_TRIGGER_GPDMA1_CH0_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH1_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH2_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH3_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH4_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH5_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH6_TCF
- LL_GPDMA2_TRIGGER_GPDMA1_CH7_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH0_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH1_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH2_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH3_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH4_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH5_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH6_TCF
- LL_GPDMA2_TRIGGER_GPDMA2_CH7_TCF
- LL_GPDMA2_TRIGGER_TIM2_TRGO
- LL_GPDMA2_TRIGGER_TIM15_TRGO
- LL_GPDMA2_TRIGGER_TIM12_TRGO
- LL_GPDMA2_TRIGGER_LPTIM3_CH1
- LL_GPDMA2_TRIGGER_LPTIM3_CH2
- LL_GPDMA2_TRIGGER_LPTIM4_AIT
- LL_GPDMA2_TRIGGER_LPTIM5_CH1
- LL_GPDMA2_TRIGGER_LPTIM5_CH2
- LL_GPDMA2_TRIGGER_LPTIM6_CH1

- LL_GPDMA2_TRIGGER_LPTIM6_CH2
- LL_GPDMA2_TRIGGER_TIM15_TRGO (*)
- LL_GPDMA2_TRIGGER_TIM12_TRGO (*)
- LL_GPDMA2_TRIGGER_LPTIM3_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM3_CH2 (*)
- LL_GPDMA2_TRIGGER_LPTIM4_AIT (*)
- LL_GPDMA2_TRIGGER_LPTIM5_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM5_CH2 (*)
- LL_GPDMA2_TRIGGER_LPTIM6_CH1 (*)
- LL_GPDMA2_TRIGGER_LPTIM6_CH2 (*)
- LL_GPDMA2_TRIGGER_COMP1_OUT (*)
- LL_GPDMA2_TRIGGER_EVENTOUT (*)

Notes

- This API is used for all available DMA channels.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_GetHWTrigger

LL_DMA_SetTransferMode

Function name

__STATIC_INLINE void LL_DMA_SetTransferMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)

Function description

Set DMA transfer mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_7
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_NORMAL
  - LL_DMA_PFCTRL

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 PFREQ LL_DMA_SetTransferMode

LL_DMA_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTransferMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA transfer mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_NORMAL
  - LL_DMA_PFCTRL

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_GetTransferMode

LL_DMA_ConfigBlkRptAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure addresses update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT or LL_DMA_BLKRPT_DEST_ADDR_DECREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT or LL_DMA_BLKRPT_SRC_ADDR_DECREMENT
  - LL_DMA_BURST_DEST_ADDR_INCREMENT or LL_DMA_BURST_DEST_ADDR_DECREMENT
  - LL_DMA_BURST_SRC_ADDR_INCREMENT or LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 BRSDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 DDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 SDEC LL_DMA_ConfigBlkRptAddrUpdate

LL_DMA_ConfigBlkCounters

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkCounters (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength, uint32_t BlkRptCount)

Function description

Configure DMA Block number of data and repeat Count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkDataLength:** Block transfer length Value between 0 to 0x0000FFFF
- **BlkRptCount:** Block repeat counter Value between 0 to 0x00000EFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_ConfigBlkCounters
- CBR1 BRC LL_DMA_ConfigBlkCounters

LL_DMA_SetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrUpdate)

Function description

Set block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptDestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_SetBlkRptDestAddrUpdate

LL_DMA_GetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_GetBlkRptDestAddrUpdate

LL_DMA_SetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrUpdate)

Function description

Set block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptSrcAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_SetBlkRptSrcAddrUpdate

LL_DMA_GetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_GetBlkRptSrcAddrUpdate

LL_DMA_SetDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrUpdate)

Function description

Set destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BURST_DEST_ADDR_INCREMENT
  - LL_DMA_BURST_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_SetDestAddrUpdate

LL_DMA_GetDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BURST_DEST_ADDR_INCREMENT
  - LL_DMA_BURST_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_GetDestAddrUpdate

LL_DMA_SetSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrUpdate)

Function description

Set source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BURST_SRC_ADDR_INCREMENT
  - LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_SetSrcAddrUpdate

LL_DMA_GetSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BURST_SRC_ADDR_INCREMENT
  - LL_DMA_BURST_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_GetSrcAddrUpdate

LL_DMA_SetBlkRptCount

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptCount)

Function description

Set block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptCount:** Block repeat counter Value between 0 to 0x00000EFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_SetBlkRptCount

LL_DMA_GetBlkRptCount

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x00000EFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_GetBlkRptCount

LL_DMA_SetBlkDataLength

Function name

__STATIC_INLINE void LL_DMA_SetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength)

Function description

Set block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkDataLength:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_SetBlkDataLength

LL_DMA_GetBlkDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_GetBlkDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DestAddress)

Function description

Configure the source and destination addresses.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAddress:** Between 0 to 0xFFFFFFFF
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_ConfigAddresses
- CDAR DA LL_DMA_ConfigAddresses

LL_DMA_SetSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress)

Function description

Set source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_SetSrcAddress

LL_DMA_GetSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_GetSrcAddress

LL_DMA_SetDestAddress

Function name

__STATIC_INLINE void LL_DMA_SetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddress)

Function description

Set destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_SetDestAddress

LL_DMA_GetDestAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_GetDestAddress

LL_DMA_ConfigAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset, uint32_t DestAddrOffset)

Function description

Configure source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestAddrOffset:** Between 0 to 0x00001FFF
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_ConfigAddrUpdateValue
- CTR3 SAO LL_DMA_ConfigAddrUpdateValue

LL_DMA_SetDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrOffset)

Function description

Set destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DestAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_SetDestAddrUpdateValue

LL_DMA_GetDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CDAR DAO LL_DMA_GetDestAddrUpdateValue

LL_DMA_SetSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset)

Function description

Set source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_SetSrcAddrUpdateValue

LL_DMA_GetSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_GetSrcAddrUpdateValue

LL_DMA_ConfigBlkRptAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset, uint32_t BlkRptDestAddrOffset)

Function description

Configure the block repeated source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_ConfigBlkRptAddrUpdateValue
- CBR2 BRSAO LL_DMA_ConfigBlkRptAddrUpdateValue

LL_DMA_SetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrOffset)

Function description

Set block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_SetBlkRptDestAddrUpdateValue

LL_DMA_GetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x0000FFFF.

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_GetBlkRptDestAddrUpdateValue

LL_DMA_SetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset)

Function description

Set block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_SetBlkRptSrcAddrUpdateValue

LL_DMA_GetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_GetBlkRptSrcAddrUpdateValue

LL_DMA_ConfigLinkUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigLinkUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t RegistersUpdate, uint32_t LinkedListAddrOffset)

Function description

Configure registers update and node address offset during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **RegistersUpdate:** This parameter must be a combination of all the following values:
  - LL_DMA_UPDATE_CTR1
  - LL_DMA_UPDATE_CTR2
  - LL_DMA_UPDATE_CBR1
  - LL_DMA_UPDATE_CSAR
  - LL_DMA_UPDATE_CDAR
  - LL_DMA_UPDATE_CTR3 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CBR2 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CLLR
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For linear addressing channels, UT3 and UB2 fields are discarded.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_ConfigLinkUpdate
-
- CLLR UT2 LL_DMA_ConfigLinkUpdate
-
- CLLR UB1 LL_DMA_ConfigLinkUpdate
-
- CLLR USA LL_DMA_ConfigLinkUpdate
-
- CLLR UDA LL_DMA_ConfigLinkUpdate
-
- CLLR UT3 LL_DMA_ConfigLinkUpdate
-
- CLLR UB2 LL_DMA_ConfigLinkUpdate
-
- CLLR ULL LL_DMA_ConfigLinkUpdate

LL_DMA_EnableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_EnableCTR1Update

LL_DMA_DisableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_DisableCTR1Update

LL_DMA_IsEnabledCTR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_IsEnabledCTR1Update

LL_DMA_EnableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_EnableCTR2Update

LL_DMA_DisableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_DisableCTR2Update

LL_DMA_IsEnabledCTR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_IsEnabledCTR2Update

LL_DMA_EnableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_EnableCBR1Update

LL_DMA_DisableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_DisableCBR1Update

LL_DMA_IsEnabledCBR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_IsEnabledCBR1Update

LL_DMA_EnableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_EnableCSARUpdate

LL_DMA_DisableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_DisableCSARUpdate

LL_DMA_IsEnabledCSARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CSAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_IsEnabledCSARUpdate

LL_DMA_EnableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_EnableCDARUpdate

LL_DMA_DisableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_DisableCDARUpdate

LL_DMA_IsEnabledCDARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CDAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_IsEnabledCDARUpdate

LL_DMA_EnableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_EnableCTR3Update

LL_DMA_DisableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_DisableCTR3Update

LL_DMA_IsEnabledCTR3Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR3 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_IsEnabledCTR3Update

LL_DMA_EnableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_EnableCBR2Update

LL_DMA_DisableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_DisableCBR2Update

LL_DMA_IsEnabledCBR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_IsEnabledCBR2Update

LL_DMA_EnableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_EnableCLLRUpdate

LL_DMA_DisableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_DisableCLLRUpdate

LL_DMA_IsEnabledCLLRUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CLLR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_IsEnabledCLLRUpdate

LL_DMA_SetLinkedListAddrOffset

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListAddrOffset)

Function description

Set linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_SetLinkedListAddrOffset

LL_DMA_GetLinkedListAddrOffset

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x0000FFFC.

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_GetLinkedListAddrOffset

LL_DMA_GetFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetFIFOLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get FIFO level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **Between:** 0 to 0x000000FF.

Reference Manual to LL API cross reference:

- CSR FIFOL LL_DMA_GetFIFOLevel

LL_DMA_EnableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_EnableChannelSecure

LL_DMA_DisableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_DisableChannelSecure

LL_DMA_IsEnabledChannelSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel secure is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_IsEnabledChannelSecure

LL_DMA_EnableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_EnableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_EnableChannelPrivilege

LL_DMA_DisableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_DisableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_DisableChannelPrivilege

LL_DMA_IsEnabledChannelPrivilege

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelPrivilege (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA Channel privilege is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_IsEnabledChannelPrivilege

LL_DMA_EnableChannelLockAttribute

Function name

__STATIC_INLINE void LL_DMA_EnableChannelLockAttribute (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel lock attributes.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- RCFGLOCKR LOCKx LL_DMA_EnableChannelLockAttribute

LL_DMA_IsEnabledChannelLockAttribute

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelLockAttribute (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel attributes are locked.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR LOCKx LL_DMA_IsEnabledChannelLockAttribute

LL_DMA_ClearFlag_TO

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TOF LL_DMA_ClearFlag_TO

LL_DMA_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR SUSPF LL_DMA_ClearFlag_SUSP

LL_DMA_ClearFlag_USE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR USEF LL_DMA_ClearFlag_USE

LL_DMA_ClearFlag_ULE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear link transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR ULEF LL_DMA_ClearFlag_ULE

LL_DMA_ClearFlag_DTE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR DTEF LL_DMA_ClearFlag_DTE

LL_DMA_ClearFlag_HT

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR HTF LL_DMA_ClearFlag_HT

LL_DMA_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TCF LL_DMA_ClearFlag_TC

LL_DMA_IsActiveFlag_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TOF LL_DMA_IsActiveFlag_TO

LL_DMA_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR SUSPF LL_DMA_IsActiveFlag_SUSP

LL_DMA_IsActiveFlag_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR USEF LL_DMA_IsActiveFlag_USE

LL_DMA_IsActiveFlag_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR ULEF LL_DMA_IsActiveFlag_ULE

LL_DMA_IsActiveFlag_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR DTEF LL_DMA_IsActiveFlag_DTE

LL_DMA_IsActiveFlag_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR HTF LL_DMA_IsActiveFlag_HT

LL_DMA_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TCF LL_DMA_IsActiveFlag_TC

LL_DMA_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_IDLE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get idle flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR IDLEF LL_DMA_IsActiveFlag_IDLE

LL_DMA_IsActiveFlag_MIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_MIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if nsecure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- MISR MISx LL_DMA_IsActiveFlag_MIS

LL_DMA_IsActiveFlag_SMIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SMIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if secure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SMISR MISx LL_DMA_IsActiveFlag_SMIS

LL_DMA_EnableIT_TO

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_EnableIT_TO

LL_DMA_EnableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_EnableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_EnableIT_SUSP

LL_DMA_EnableIT_USE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_EnableIT_USE

LL_DMA_EnableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_EnableIT_ULE

LL_DMA_EnableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_EnableIT_DTE

LL_DMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_DMA_EnableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_DisableIT_TO

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_DisableIT_TO

LL_DMA_DisableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_DisableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_DisableIT_SUSP

LL_DMA_DisableIT_USE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_DisableIT_USE

LL_DMA_DisableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_DisableIT_ULE

LL_DMA_DisableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_DisableIT_DTE

LL_DMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_DMA_DisableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_IsEnabledIT_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if trigger overrun interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_IsEnabledIT_TO

LL_DMA_IsEnabledIT_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if suspension interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_IsEnabledIT_SUSP

LL_DMA_IsEnabledIT_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if user setting error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_IsEnabledIT_USE

LL_DMA_IsEnabledIT_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if update link transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_IsEnabledIT_ULE

LL_DMA_IsEnabledIT_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if data transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_IsEnabledIT_DTE

LL_DMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if half transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_Init

Function name

uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are initialized.
  - ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- A software request transfer can be done once programming the direction field in memory to memory value.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name

uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_StructInit

Function name

void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Set each LL_DMA_InitTypeDef field to default value.

Parameters

- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values

- **None.:**

LL_DMA_ListStructInit

Function name

void LL_DMA_ListStructInit (LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Set each LL_DMA_InitLinkedListTypeDef field to default value.

Parameters

- **DMA_InitLinkedListStruct:** Pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **None.:**

LL_DMA_NodeStructInit

Function name

void LL_DMA_NodeStructInit (LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct)

Function description

Set each LL_DMA_InitNodeTypeDef field to default value.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure.

Return values

- **None.:**

LL_DMA_List_Init

Function name

uint32_t LL_DMA_List_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Initialize the DMA linked list according to the specified parameters in LL_DMA_InitLinkedListTypeDef.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
- **DMA_InitLinkedListStruct:** pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are initialized.
  - ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_List_DeInit

Function name

uint32_t LL_DMA_List_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA linked list.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_CreateLinkNode

Function name

uint32_t LL_DMA_CreateLinkNode (LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct, LL_DMA_LinkNodeTypeDef * pNode)

Function description

Initializes DMA linked list node according to the specified parameters in the DMA_InitNodeStruct.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure that contains linked list node registers configurations.
- **pNode:** Pointer to linked list node to fill according to LL_DMA_LinkNodeTypeDef parameters.

Return values

- **None:**

LL_DMA_ConnectLinkNode

Function name

void LL_DMA_ConnectLinkNode (LL_DMA_LinkNodeTypeDef * pPrevLinkNode, uint32_t PrevNodeCLLRIdx, LL_DMA_LinkNodeTypeDef * pNewLinkNode, uint32_t NewNodeCLLRIdx)

Function description

Connect Linked list Nodes.

Parameters

- **pPrevLinkNode:** Pointer to previous linked list node to be connected to new Linked list node.
- **PrevNodeCLLRIdx:** Offset of Previous Node CLLR register. This parameter can be a value of CLLR offset.
- **pNewLinkNode:** Pointer to new Linked list.
- **NewNodeCLLRIdx:** Offset of New Node CLLR register. This parameter can be a value of CLLR offset.

Return values

- **None:**

LL_DMA_DisconnectNextLinkNode

Function name

void LL_DMA_DisconnectNextLinkNode (LL_DMA_LinkNodeTypeDef * pLinkNode, uint32_t LinkNodeCLLRIdx)

Function description

Disconnect the next linked list node.

Parameters

- **pLinkNode:** Pointer to linked list node to be disconnected from the next one.
- **LinkNodeCLLRIdx:** Offset of Link Node CLLR register.

Return values

- **None.:**

95.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

95.3.1 DMA

DMA

Block Hardware Request

LL_DMA_HWREQUEST_SINGLEBURST

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a burst level

LL_DMA_HWREQUEST_BLK

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a block level

Block Repeat Destination Address Update Mode

LL_DMA_BLKRPT_DEST_ADDR_INCREMENT

Destination address is incremented after each block transfer by destination update value

LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Destination address is decremented after each block transfer by destination update value

Block Repeat Source Address Update Mode

LL_DMA_BLKRPT_SRC_ADDR_INCREMENT

Source address pointer is incremented after each block transfer by source update value

LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Source address pointer is decremented after each block transfer by source update value

Channel

LL_DMA_CHANNEL_0

LL_DMA_CHANNEL_1

LL_DMA_CHANNEL_2

LL_DMA_CHANNEL_3

LL_DMA_CHANNEL_4

LL_DMA_CHANNEL_5

LL_DMA_CHANNEL_6

LL_DMA_CHANNEL_7

LL_DMA_CHANNEL_8

LL_DMA_CHANNEL_9

LL_DMA_CHANNEL_10

LL_DMA_CHANNEL_11

LL_DMA_CHANNEL_12

LL_DMA_CHANNEL_13

LL_DMA_CHANNEL_14

LL_DMA_CHANNEL_15

LL_DMA_CHANNEL_ALL

CLLR offset

LL_DMA_CLLR_OFFSET0

LL_DMA_CLLR_OFFSET1

LL_DMA_CLLR_OFFSET2

LL_DMA_CLLR_OFFSET3

LL_DMA_CLLR_OFFSET4

LL_DMA_CLLR_OFFSET5

LL_DMA_CLLR_OFFSET6

LL_DMA_CLLR_OFFSET7

Data Alignment

LL_DMA_DATA_ALIGN_ZEROPADD

If src data width < dest data width : => Right Aligned padded with 0 up to destination data width. If src data width > dest data width : => Right Aligned Left Truncated down to destination data width.

LL_DMA_DATA_ALIGN_SIGNEXTPADD

If src data width < dest data width : => Right Aligned padded with sign extended up to destination data width. If src data width > dest data width : => Left Aligned Right Truncated down to the destination data width

LL_DMA_DATA_PACK_UNPACK

If src data width < dest data width : => Packed at the destination data width If src data width > dest data width : => Unpacked at the destination data width

Destination Allocated Port

LL_DMA_DEST_ALLOCATED_PORT0

Destination Allocated Port 0

LL_DMA_DEST_ALLOCATED_PORT1

Destination Allocated Port 1

Destination Data Width

LL_DMA_DEST_DATAWIDTH_BYTE

Destination Data Width : Byte

LL_DMA_DEST_DATAWIDTH_HALFWORD

Destination Data Width : HalfWord

LL_DMA_DEST_DATAWIDTH_WORD

Destination Data Width : Word

Destination Increment Mode

LL_DMA_DEST_FIXED

Destination fixed single/burst

LL_DMA_DEST_INCREMENT

Destination incremented single/burst

Destination Security Attribute

LL_DMA_CHANNEL_DEST_NSEC

NSecure transfer from the destination

LL_DMA_CHANNEL_DEST_SEC

Secure transfer from the destination

Burst Destination Address Update Mode

LL_DMA_BURST_DEST_ADDR_INCREMENT

Destination address pointer is incremented after each burst transfer by destination update value

LL_DMA_BURST_DEST_ADDR_DECREMENT

Destination address pointer is decremented after each burst transfer by destination update value

Destination Byte Exchange

LL_DMA_DEST_BYTE_PRESERVE

No destination Byte exchange when destination data width > Byte

LL_DMA_DEST_BYTE_EXCHANGE

Destination Byte exchange when destination data width > Byte

Destination Half-Word Exchange

LL_DMA_DEST_HALFWORD_PRESERVE

No destination Half-Word exchange when destination data width is word

LL_DMA_DEST_HALFWORD_EXCHANGE

Destination Half-Word exchange when destination data width is word

Linked list node type

LL_DMA_GPDMA_LINEAR_NODE

GPDMA node : linear addressing node

LL_DMA_GPDMA_2D_NODE

GPDMA node : 2 dimension addressing node

Linked list register update

LL_DMA_UPDATE_CTR1

Update CTR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR2

Update CTR2 register from memory : available for all DMA channels

LL_DMA_UPDATE_CBR1

Update CBR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CSAR

Update CSAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CDAR

Update CDAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR3

Update CTR3 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CBR2

Update CBR2 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CLLR

Update CLLR register from memory : available for all DMA channels

Linked List Allocated Port

LL_DMA_LINK_ALLOCATED_PORT0

Linked List Allocated Port 0

LL_DMA_LINK_ALLOCATED_PORT1

Linked List Allocated Port 1

Link Step Mode

LL_DMA_LSM_FULL_EXECUTION

Channel execute the full linked list

LL_DMA_LSM_1LINK_EXECUTION

Channel execute one node of the linked list

Priority Level

LL_DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low Weight

LL_DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid Weight

LL_DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High Weight

LL_DMA_HIGH_PRIORITY

Priority level : High Priority

Request Selection

LL_GPDMA1_REQUEST_ADC1

GPDMA1 HW request is ADC1

LL_GPDMA1_REQUEST_ADC2

GPDMA1 HW request is ADC2

LL_GPDMA1_REQUEST_DAC1_CH1

GPDMA1 HW request is DAC1_CH1

LL_GPDMA1_REQUEST_DAC1_CH2

GPDMA1 HW request is DAC1_CH2

LL_GPDMA1_REQUEST_TIM6_UP

GPDMA1 HW request is TIM6_UP

LL_GPDMA1_REQUEST_TIM7_UP

GPDMA1 HW request is TIM7_UP

LL_GPDMA1_REQUEST_SPI1_RX

GPDMA1 HW request is SPI1_RX

LL_GPDMA1_REQUEST_SPI1_TX

GPDMA1 HW request is SPI1_TX

LL_GPDMA1_REQUEST_SPI2_RX

GPDMA1 HW request is SPI2_RX

LL_GPDMA1_REQUEST_SPI2_TX

GPDMA1 HW request is SPI2_TX

LL_GPDMA1_REQUEST_SPI3_RX

GPDMA1 HW request is SPI3_RX

LL_GPDMA1_REQUEST_SPI3_TX

GPDMA1 HW request is SPI3_TX

LL_GPDMA1_REQUEST_I2C1_RX

GPDMA1 HW request is I2C1_RX

LL_GPDMA1_REQUEST_I2C1_TX

GPDMA1 HW request is I2C1_TX

LL_GPDMA1_REQUEST_I2C2_RX

GPDMA1 HW request is I2C2_RX

LL_GPDMA1_REQUEST_I2C2_TX

GPDMA1 HW request is I2C2_TX

LL_GPDMA1_REQUEST_I2C3_RX

GPDMA1 HW request is I2C3_RX

LL_GPDMA1_REQUEST_I2C3_TX

GPDMA1 HW request is I2C3_TX

LL_GPDMA1_REQUEST_USART1_RX

GPDMA1 HW request is USART1_RX

LL_GPDMA1_REQUEST_USART1_TX

GPDMA1 HW request is USART1_TX

LL_GPDMA1_REQUEST_USART2_RX

GPDMA1 HW request is USART2_RX

LL_GPDMA1_REQUEST_USART2_TX

GPDMA1 HW request is USART2_TX

LL_GPDMA1_REQUEST_USART3_RX

GPDMA1 HW request is USART3_RX

LL_GPDMA1_REQUEST_USART3_TX

GPDMA1 HW request is USART3_TX

LL_GPDMA1_REQUEST_UART4_RX

GPDMA1 HW request is UART4_RX

LL_GPDMA1_REQUEST_UART4_TX

GPDMA1 HW request is UART4_TX

LL_GPDMA1_REQUEST_UART5_RX

GPDMA1 HW request is UART5_RX

LL_GPDMA1_REQUEST_UART5_TX

GPDMA1 HW request is UART5_TX

LL_GPDMA1_REQUEST_USART6_RX

GPDMA1 HW request is USART6_RX

LL_GPDMA1_REQUEST_USART6_TX

GPDMA1 HW request is USART6_TX

LL_GPDMA1_REQUEST_UART7_RX

GPDMA1 HW request is UART7_RX

LL_GPDMA1_REQUEST_UART7_TX

GPDMA1 HW request is UART7_TX

LL_GPDMA1_REQUEST_UART8_RX

GPDMA1 HW request is UART8_RX

LL_GPDMA1_REQUEST_UART8_TX

GPDMA1 HW request is UART8_TX

LL_GPDMA1_REQUEST_UART9_RX

GPDMA1 HW request is UART9_RX

LL_GPDMA1_REQUEST_UART9_TX

GPDMA1 HW request is UART9_TX

LL_GPDMA1_REQUEST_USART10_RX

GPDMA1 HW request is USART10_RX

LL_GPDMA1_REQUEST_USART10_TX

GPDMA1 HW request is USART10_TX

LL_GPDMA1_REQUEST_USART11_RX

GPDMA1 HW request is USART11_RX

LL_GPDMA1_REQUEST_USART11_TX

GPDMA1 HW request is USART11_TX

LL_GPDMA1_REQUEST_UART12_RX

GPDMA1 HW request is UART12_RX

LL_GPDMA1_REQUEST_UART12_TX

GPDMA1 HW request is UART12_TX

LL_GPDMA1_REQUEST_LPUART1_RX

GPDMA1 HW request is LPUART1_RX

LL_GPDMA1_REQUEST_LPUART1_TX

GPDMA1 HW request is LPUART1_TX

LL_GPDMA1_REQUEST_SPI4_RX

GPDMA1 HW request is SPI4_RX

LL_GPDMA1_REQUEST_SPI4_TX

GPDMA1 HW request is SPI4_TX

LL_GPDMA1_REQUEST_SPI5_RX

GPDMA1 HW request is SPI5_RX

LL_GPDMA1_REQUEST_SPI5_TX

GPDMA1 HW request is SPI5_TX

LL_GPDMA1_REQUEST_SPI6_RX

GPDMA1 HW request is SPI6_RX

LL_GPDMA1_REQUEST_SPI6_TX

GPDMA1 HW request is SPI6_TX

LL_GPDMA1_REQUEST_SAI1_A

GPDMA1 HW request is SAI1_A

LL_GPDMA1_REQUEST_SAI1_B

GPDMA1 HW request is SAI1_B

LL_GPDMA1_REQUEST_SAI2_A

GPDMA1 HW request is SAI2_A

LL_GPDMA1_REQUEST_SAI2_B

GPDMA1 HW request is SAI2_B

LL_GPDMA1_REQUEST_OCTOSPI1

GPDMA1 HW request is OCTOSPI1

LL_GPDMA1_REQUEST_TIM1_CH1

GPDMA1 HW request is TIM1_CH1

LL_GPDMA1_REQUEST_TIM1_CH2

GPDMA1 HW request is TIM1_CH2

LL_GPDMA1_REQUEST_TIM1_CH3

GPDMA1 HW request is TIM1_CH3

LL_GPDMA1_REQUEST_TIM1_CH4

GPDMA1 HW request is TIM1_CH4

LL_GPDMA1_REQUEST_TIM1_UP

GPDMA1 HW request is TIM1_UP

LL_GPDMA1_REQUEST_TIM1_TRIG

GPDMA1 HW request is TIM1_TRIG

LL_GPDMA1_REQUEST_TIM1_COM

GPDMA1 HW request is TIM1_COM

LL_GPDMA1_REQUEST_TIM8_CH1

GPDMA1 HW request is TIM8_CH1

LL_GPDMA1_REQUEST_TIM8_CH2

GPDMA1 HW request is TIM8_CH2

LL_GPDMA1_REQUEST_TIM8_CH3

GPDMA1 HW request is TIM8_CH3

LL_GPDMA1_REQUEST_TIM8_CH4

GPDMA1 HW request is TIM8_CH4

LL_GPDMA1_REQUEST_TIM8_UP

GPDMA1 HW request is TIM8_UP

LL_GPDMA1_REQUEST_TIM8_TRIG

GPDMA1 HW request is TIM8_TRIG

LL_GPDMA1_REQUEST_TIM8_COM

GPDMA1 HW request is TIM8_COM

LL_GPDMA1_REQUEST_TIM2_CH1

GPDMA1 HW request is TIM2_CH1

LL_GPDMA1_REQUEST_TIM2_CH2

GPDMA1 HW request is TIM2_CH2

LL_GPDMA1_REQUEST_TIM2_CH3

GPDMA1 HW request is TIM2_CH3

LL_GPDMA1_REQUEST_TIM2_CH4

GPDMA1 HW request is TIM2_CH4

LL_GPDMA1_REQUEST_TIM2_UP

GPDMA1 HW request is TIM2_UP

LL_GPDMA1_REQUEST_TIM3_CH1

GPDMA1 HW request is TIM3_CH1

LL_GPDMA1_REQUEST_TIM3_CH2

GPDMA1 HW request is TIM3_CH2

LL_GPDMA1_REQUEST_TIM3_CH3

GPDMA1 HW request is TIM3_CH3

LL_GPDMA1_REQUEST_TIM3_CH4

GPDMA1 HW request is TIM3_CH4

LL_GPDMA1_REQUEST_TIM3_UP

GPDMA1 HW request is TIM3_UP

LL_GPDMA1_REQUEST_TIM3_TRIG

GPDMA1 HW request is TIM3_TRIG

LL_GPDMA1_REQUEST_TIM4_CH1

GPDMA1 HW request is TIM4_CH1

LL_GPDMA1_REQUEST_TIM4_CH2

GPDMA1 HW request is TIM4_CH2

LL_GPDMA1_REQUEST_TIM4_CH3

GPDMA1 HW request is TIM4_CH3

LL_GPDMA1_REQUEST_TIM4_CH4

GPDMA1 HW request is TIM4_CH4

LL_GPDMA1_REQUEST_TIM4_UP

GPDMA1 HW request is TIM4_UP

LL_GPDMA1_REQUEST_TIM5_CH1

GPDMA1 HW request is TIM5_CH1

LL_GPDMA1_REQUEST_TIM5_CH2

GPDMA1 HW request is TIM5_CH2

LL_GPDMA1_REQUEST_TIM5_CH3

GPDMA1 HW request is TIM5_CH3

LL_GPDMA1_REQUEST_TIM5_CH4

GPDMA1 HW request is TIM5_CH4

LL_GPDMA1_REQUEST_TIM5_UP

GPDMA1 HW request is TIM5_UP

LL_GPDMA1_REQUEST_TIM5_TRIG

GPDMA1 HW request is TIM5_TRIG

LL_GPDMA1_REQUEST_TIM15_CH1

GPDMA1 HW request is TIM15_CH1

LL_GPDMA1_REQUEST_TIM15_UP

GPDMA1 HW request is TIM15_UP

LL_GPDMA1_REQUEST_TIM15_TRIG

GPDMA1 HW request is TIM15_TRIG

LL_GPDMA1_REQUEST_TIM15_COM

GPDMA1 HW request is TIM15_COM

LL_GPDMA1_REQUEST_TIM16_CH1

GPDMA1 HW request is TIM16_CH1

LL_GPDMA1_REQUEST_TIM16_UP

GPDMA1 HW request is TIM16_UP

LL_GPDMA1_REQUEST_TIM17_CH1

GPDMA1 HW request is TIM17_CH1

LL_GPDMA1_REQUEST_TIM17_UP

GPDMA1 HW request is TIM17_UP

LL_GPDMA1_REQUEST_LPTIM1_IC1

GPDMA1 HW request is LPTIM1_IC1

LL_GPDMA1_REQUEST_LPTIM1_IC2

GPDMA1 HW request is LPTIM1_IC2

LL_GPDMA1_REQUEST_LPTIM1_UE

GPDMA1 HW request is LPTIM1_UE

LL_GPDMA1_REQUEST_LPTIM2_IC1

GPDMA1 HW request is LPTIM2_IC1

LL_GPDMA1_REQUEST_LPTIM2_IC2

GPDMA1 HW request is LPTIM2_IC2

LL_GPDMA1_REQUEST_LPTIM2_UE

GPDMA1 HW request is LPTIM2_UE

LL_GPDMA1_REQUEST_DCMI

GPDMA1 HW request is DCMI

LL_GPDMA1_REQUEST_AES_OUT

GPDMA1 HW request is AES_OUT

LL_GPDMA1_REQUEST_AES_IN

GPDMA1 HW request is AES_IN

LL_GPDMA1_REQUEST_HASH_IN

GPDMA1 HW request is HASH_IN

LL_GPDMA1_REQUEST_UCPD1_RX

GPDMA1 HW request is UCPD1_RX

LL_GPDMA1_REQUEST_UCPD1_TX

GPDMA1 HW request is UCPD1_TX

LL_GPDMA1_REQUEST_CORDIC_READ

GPDMA1 HW request is CORDIC_READ

LL_GPDMA1_REQUEST_CORDIC_WRITE

GPDMA1 HW request is CORDIC_WRITE

LL_GPDMA1_REQUEST_FMAC_READ

GPDMA1 HW request is FMAC_READ

LL_GPDMA1_REQUEST_FMAC_WRITE

GPDMA1 HW request is FMAC_WRITE

LL_GPDMA1_REQUEST_SAES_OUT

GPDMA1 HW request is SAES_OUT

LL_GPDMA1_REQUEST_SAES_IN

GPDMA1 HW request is SAES_IN

LL_GPDMA1_REQUEST_I3C1_RX

GPDMA1 HW request is I3C1_RX

LL_GPDMA1_REQUEST_I3C1_TX

GPDMA1 HW request is I3C1_TX

LL_GPDMA1_REQUEST_I3C1_TC

GPDMA1 HW request is I3C1_TC

LL_GPDMA1_REQUEST_I3C1_RS

GPDMA1 HW request is I3C1_RS

LL_GPDMA1_REQUEST_I2C4_RX

GPDMA1 HW request is I2C4_RX

LL_GPDMA1_REQUEST_I2C4_TX

GPDMA1 HW request is I2C4_TX

LL_GPDMA1_REQUEST_LPTIM3_IC1

GPDMA1 HW request is LPTIM3_IC1

LL_GPDMA1_REQUEST_LPTIM3_IC2

GPDMA1 HW request is LPTIM3_IC2

LL_GPDMA1_REQUEST_LPTIM3_UE

GPDMA1 HW request is LPTIM3_UE

LL_GPDMA1_REQUEST_LPTIM5_IC1

GPDMA1 HW request is LPTIM5_IC1

LL_GPDMA1_REQUEST_LPTIM5_IC2

GPDMA1 HW request is LPTIM5_IC2

LL_GPDMA1_REQUEST_LPTIM5_UE

GPDMA1 HW request is LPTIM5_UE

LL_GPDMA1_REQUEST_LPTIM6_IC1

GPDMA1 HW request is LPTIM6_IC1

LL_GPDMA1_REQUEST_LPTIM6_IC2

GPDMA1 HW request is LPTIM6_IC2

LL_GPDMA1_REQUEST_LPTIM6_UE

GPDMA1 HW request is LPTIM6_UE

LL_GPDMA2_REQUEST_ADC1

GPDMA2 HW request is ADC1

LL_GPDMA2_REQUEST_ADC2

GPDMA2 HW request is ADC2

LL_GPDMA2_REQUEST_DAC1_CH1

GPDMA2 HW request is DAC1_CH1

LL_GPDMA2_REQUEST_DAC1_CH2

GPDMA2 HW request is DAC1_CH2

LL_GPDMA2_REQUEST_TIM6_UP

GPDMA2 HW request is TIM6_UP

LL_GPDMA2_REQUEST_TIM7_UP

GPDMA2 HW request is TIM7_UP

LL_GPDMA2_REQUEST_SPI1_RX

GPDMA2 HW request is SPI1_RX

LL_GPDMA2_REQUEST_SPI1_TX

GPDMA2 HW request is SPI1_TX

LL_GPDMA2_REQUEST_SPI2_RX

GPDMA2 HW request is SPI2_RX

LL_GPDMA2_REQUEST_SPI2_TX

GPDMA2 HW request is SPI2_TX

LL_GPDMA2_REQUEST_SPI3_RX

GPDMA2 HW request is SPI3_RX

LL_GPDMA2_REQUEST_SPI3_TX

GPDMA2 HW request is SPI3_TX

LL_GPDMA2_REQUEST_I2C1_RX

GPDMA2 HW request is I2C1_RX

LL_GPDMA2_REQUEST_I2C1_TX

GPDMA2 HW request is I2C1_TX

LL_GPDMA2_REQUEST_I2C2_RX

GPDMA2 HW request is I2C2_RX

LL_GPDMA2_REQUEST_I2C2_TX

GPDMA2 HW request is I2C2_TX

LL_GPDMA2_REQUEST_I2C3_RX

GPDMA2 HW request is I2C3_RX

LL_GPDMA2_REQUEST_I2C3_TX

GPDMA2 HW request is I2C3_TX

LL_GPDMA2_REQUEST_USART1_RX

GPDMA2 HW request is USART1_RX

LL_GPDMA2_REQUEST_USART1_TX

GPDMA2 HW request is USART1_TX

LL_GPDMA2_REQUEST_USART2_RX

GPDMA2 HW request is USART2_RX

LL_GPDMA2_REQUEST_USART2_TX

GPDMA2 HW request is USART2_TX

LL_GPDMA2_REQUEST_USART3_RX

GPDMA2 HW request is USART3_RX

LL_GPDMA2_REQUEST_USART3_TX

GPDMA2 HW request is USART3_TX

LL_GPDMA2_REQUEST_UART4_RX

GPDMA2 HW request is UART4_RX

LL_GPDMA2_REQUEST_UART4_TX

GPDMA2 HW request is UART4_TX

LL_GPDMA2_REQUEST_UART5_RX

GPDMA2 HW request is UART5_RX

LL_GPDMA2_REQUEST_UART5_TX

GPDMA2 HW request is UART5_TX

LL_GPDMA2_REQUEST_USART6_RX

GPDMA2 HW request is USART6_RX

LL_GPDMA2_REQUEST_USART6_TX

GPDMA2 HW request is USART6_TX

LL_GPDMA2_REQUEST_UART7_RX

GPDMA2 HW request is UART7_RX

LL_GPDMA2_REQUEST_UART7_TX

GPDMA2 HW request is UART7_TX

LL_GPDMA2_REQUEST_UART8_RX

GPDMA2 HW request is UART8_RX

LL_GPDMA2_REQUEST_UART8_TX

GPDMA2 HW request is UART8_TX

LL_GPDMA2_REQUEST_UART9_RX

GPDMA2 HW request is UART9_RX

LL_GPDMA2_REQUEST_UART9_TX

GPDMA2 HW request is UART9_TX

LL_GPDMA2_REQUEST_USART10_RX

GPDMA2 HW request is USART10_RX

LL_GPDMA2_REQUEST_USART10_TX

GPDMA2 HW request is USART10_TX

LL_GPDMA2_REQUEST_USART11_RX

GPDMA2 HW request is USART11_RX

LL_GPDMA2_REQUEST_USART11_TX

GPDMA2 HW request is USART11_TX

LL_GPDMA2_REQUEST_UART12_RX

GPDMA2 HW request is UART12_RX

LL_GPDMA2_REQUEST_UART12_TX

GPDMA2 HW request is UART12_TX

LL_GPDMA2_REQUEST_LPUART1_RX

GPDMA2 HW request is LPUART1_RX

LL_GPDMA2_REQUEST_LPUART1_TX

GPDMA2 HW request is LPUART1_TX

LL_GPDMA2_REQUEST_SPI4_RX

GPDMA2 HW request is SPI4_RX

LL_GPDMA2_REQUEST_SPI4_TX

GPDMA2 HW request is SPI4_TX

LL_GPDMA2_REQUEST_SPI5_RX

GPDMA2 HW request is SPI5_RX

LL_GPDMA2_REQUEST_SPI5_TX

GPDMA2 HW request is SPI5_TX

LL_GPDMA2_REQUEST_SPI6_RX

GPDMA2 HW request is SPI6_RX

LL_GPDMA2_REQUEST_SPI6_TX

GPDMA2 HW request is SPI6_TX

LL_GPDMA2_REQUEST_SAI1_A

GPDMA2 HW request is SAI1_A

LL_GPDMA2_REQUEST_SAI1_B

GPDMA2 HW request is SAI1_B

LL_GPDMA2_REQUEST_SAI2_A

GPDMA2 HW request is SAI2_A

LL_GPDMA2_REQUEST_SAI2_B

GPDMA2 HW request is SAI2_B

LL_GPDMA2_REQUEST_OCTOSPI1

GPDMA2 HW request is OCTOSPI1

LL_GPDMA2_REQUEST_TIM1_CH1

GPDMA2 HW request is TIM1_CH1

LL_GPDMA2_REQUEST_TIM1_CH2

GPDMA2 HW request is TIM1_CH2

LL_GPDMA2_REQUEST_TIM1_CH3

GPDMA2 HW request is TIM1_CH3

LL_GPDMA2_REQUEST_TIM1_CH4

GPDMA2 HW request is TIM1_CH4

LL_GPDMA2_REQUEST_TIM1_UP

GPDMA2 HW request is TIM1_UP

LL_GPDMA2_REQUEST_TIM1_TRIG

GPDMA2 HW request is TIM1_TRIG

LL_GPDMA2_REQUEST_TIM1_COM

GPDMA2 HW request is TIM1_COM

LL_GPDMA2_REQUEST_TIM8_CH1

GPDMA2 HW request is TIM8_CH1

LL_GPDMA2_REQUEST_TIM8_CH2

GPDMA2 HW request is TIM8_CH2

LL_GPDMA2_REQUEST_TIM8_CH3

GPDMA2 HW request is TIM8_CH3

LL_GPDMA2_REQUEST_TIM8_CH4

GPDMA2 HW request is TIM8_CH4

LL_GPDMA2_REQUEST_TIM8_UP

GPDMA2 HW request is TIM8_UP

LL_GPDMA2_REQUEST_TIM8_TRIG

GPDMA2 HW request is TIM8_TRIG

LL_GPDMA2_REQUEST_TIM8_COM

GPDMA2 HW request is TIM8_COM

LL_GPDMA2_REQUEST_TIM2_CH1

GPDMA2 HW request is TIM2_CH1

LL_GPDMA2_REQUEST_TIM2_CH2

GPDMA2 HW request is TIM2_CH2

LL_GPDMA2_REQUEST_TIM2_CH3

GPDMA2 HW request is TIM2_CH3

LL_GPDMA2_REQUEST_TIM2_CH4

GPDMA2 HW request is TIM2_CH4

LL_GPDMA2_REQUEST_TIM2_UP

GPDMA2 HW request is TIM2_UP

LL_GPDMA2_REQUEST_TIM3_CH1

GPDMA2 HW request is TIM3_CH1

LL_GPDMA2_REQUEST_TIM3_CH2

GPDMA2 HW request is TIM3_CH2

LL_GPDMA2_REQUEST_TIM3_CH3

GPDMA2 HW request is TIM3_CH3

LL_GPDMA2_REQUEST_TIM3_CH4

GPDMA2 HW request is TIM3_CH4

LL_GPDMA2_REQUEST_TIM3_UP

GPDMA2 HW request is TIM3_UP

LL_GPDMA2_REQUEST_TIM3_TRIG

GPDMA2 HW request is TIM3_TRIG

LL_GPDMA2_REQUEST_TIM4_CH1

GPDMA2 HW request is TIM4_CH1

LL_GPDMA2_REQUEST_TIM4_CH2

GPDMA2 HW request is TIM4_CH2

LL_GPDMA2_REQUEST_TIM4_CH3

GPDMA2 HW request is TIM4_CH3

LL_GPDMA2_REQUEST_TIM4_CH4

GPDMA2 HW request is TIM4_CH4

LL_GPDMA2_REQUEST_TIM4_UP

GPDMA2 HW request is TIM4_UP

LL_GPDMA2_REQUEST_TIM5_CH1

GPDMA2 HW request is TIM5_CH1

LL_GPDMA2_REQUEST_TIM5_CH2

GPDMA2 HW request is TIM5_CH2

LL_GPDMA2_REQUEST_TIM5_CH3

GPDMA2 HW request is TIM5_CH3

LL_GPDMA2_REQUEST_TIM5_CH4

GPDMA2 HW request is TIM5_CH4

LL_GPDMA2_REQUEST_TIM5_UP

GPDMA2 HW request is TIM5_UP

LL_GPDMA2_REQUEST_TIM5_TRIG

GPDMA2 HW request is TIM5_TRIG

LL_GPDMA2_REQUEST_TIM15_CH1

GPDMA2 HW request is TIM15_CH1

LL_GPDMA2_REQUEST_TIM15_UP

GPDMA2 HW request is TIM15_UP

LL_GPDMA2_REQUEST_TIM15_TRIG

GPDMA2 HW request is TIM15_TRIG

LL_GPDMA2_REQUEST_TIM15_COM

GPDMA2 HW request is TIM15_COM

LL_GPDMA2_REQUEST_TIM16_CH1

GPDMA2 HW request is TIM16_CH1

LL_GPDMA2_REQUEST_TIM16_UP

GPDMA2 HW request is TIM16_UP

LL_GPDMA2_REQUEST_TIM17_CH1

GPDMA2 HW request is TIM17_CH1

LL_GPDMA2_REQUEST_TIM17_UP

GPDMA2 HW request is TIM17_UP

LL_GPDMA2_REQUEST_LPTIM1_IC1

GPDMA2 HW request is LPTIM1_IC1

LL_GPDMA2_REQUEST_LPTIM1_IC2

GPDMA2 HW request is LPTIM1_IC2

LL_GPDMA2_REQUEST_LPTIM1_UE

GPDMA2 HW request is LPTIM1_UE

LL_GPDMA2_REQUEST_LPTIM2_IC1

GPDMA2 HW request is LPTIM2_IC1

LL_GPDMA2_REQUEST_LPTIM2_IC2

GPDMA2 HW request is LPTIM2_IC2

LL_GPDMA2_REQUEST_LPTIM2_UE

GPDMA2 HW request is LPTIM2_UE

LL_GPDMA2_REQUEST_DCMI

GPDMA2 HW request is DCMI

LL_GPDMA2_REQUEST_AES_OUT

GPDMA2 HW request is AES_OUT

LL_GPDMA2_REQUEST_AES_IN

GPDMA2 HW request is AES_IN

LL_GPDMA2_REQUEST_HASH_IN

GPDMA2 HW request is HASH_IN

LL_GPDMA2_REQUEST_UCPD1_RX

GPDMA2 HW request is UCPD1_RX

LL_GPDMA2_REQUEST_UCPD1_TX

GPDMA2 HW request is UCPD1_TX

LL_GPDMA2_REQUEST_CORDIC_READ

GPDMA2 HW request is CORDIC_READ

LL_GPDMA2_REQUEST_CORDIC_WRITE

GPDMA2 HW request is CORDIC_WRITE

LL_GPDMA2_REQUEST_FMAC_READ

GPDMA2 HW request is FMAC_READ

LL_GPDMA2_REQUEST_FMAC_WRITE

GPDMA2 HW request is FMAC_WRITE

LL_GPDMA2_REQUEST_SAES_OUT

GPDMA2 HW request is SAES_OUT

LL_GPDMA2_REQUEST_SAES_IN

GPDMA2 HW request is SAES_IN

LL_GPDMA2_REQUEST_I3C1_RX

GPDMA2 HW request is I3C1_RX

LL_GPDMA2_REQUEST_I3C1_TX

GPDMA2 HW request is I3C1_TX

LL_GPDMA2_REQUEST_I3C1_TC

GPDMA2 HW request is I3C1_TC

LL_GPDMA2_REQUEST_I3C1_RS

GPDMA2 HW request is I3C1_RS

LL_GPDMA2_REQUEST_I2C4_RX

GPDMA2 HW request is I2C4_RX

LL_GPDMA2_REQUEST_I2C4_TX

GPDMA2 HW request is I2C4_TX

LL_GPDMA2_REQUEST_LPTIM3_IC1

GPDMA2 HW request is LPTIM3_IC1

LL_GPDMA2_REQUEST_LPTIM3_IC2

GPDMA2 HW request is LPTIM3_IC2

LL_GPDMA2_REQUEST_LPTIM3_UE

GPDMA2 HW request is LPTIM3_UE

LL_GPDMA2_REQUEST_LPTIM5_IC1

GPDMA2 HW request is LPTIM5_IC1

LL_GPDMA2_REQUEST_LPTIM5_IC2

GPDMA2 HW request is LPTIM5_IC2

LL_GPDMA2_REQUEST_LPTIM5_UE

GPDMA2 HW request is LPTIM5_UE

LL_GPDMA2_REQUEST_LPTIM6_IC1

GPDMA2 HW request is LPTIM6_IC1

LL_GPDMA2_REQUEST_LPTIM6_IC2

GPDMA2 HW request is LPTIM6_IC2

LL_GPDMA2_REQUEST_LPTIM6_UE

GPDMA2 HW request is LPTIM6_UE

Source Allocated Port

LL_DMA_SRC_ALLOCATED_PORT0

Source Allocated Port 0

LL_DMA_SRC_ALLOCATED_PORT1

Source Allocated Port 1

Source Data Width

LL_DMA_SRC_DATAWIDTH_BYTE

Source Data Width : Byte

LL_DMA_SRC_DATAWIDTH_HALFWORD

Source Data Width : HalfWord

LL_DMA_SRC_DATAWIDTH_WORD

Source Data Width : Word

Source Increment Mode

LL_DMA_SRC_FIXED

Source fixed single/burst

LL_DMA_SRC_INCREMENT

Source incremented single/burst

Source Security Attribute

LL_DMA_CHANNEL_NSEC

NSecure channel

LL_DMA_CHANNEL_SEC

Secure channel

LL_DMA_CHANNEL_SRC_NSEC

NSecure transfer from the source

LL_DMA_CHANNEL_SRC_SEC

Secure transfer from the source

Burst Source Address Update Mode

LL_DMA_BURST_SRC_ADDR_INCREMENT

Source address pointer is incremented after each burst transfer by source update value

LL_DMA_BURST_SRC_ADDR_DECREMENT

Source address pointer is decremented after each burst transfer by source update value

Source Byte Exchange

LL_DMA_SRC_BYTE_PRESERVE

No source Byte exchange when source data width is word

LL_DMA_SRC_BYTE_EXCHANGE

Source Byte exchange when source data width is word

Transfer Direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

Transfer Event Mode

LL_DMA_TCEM_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each block

LL_DMA_TCEM_RPT_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the repeated block

LL_DMA_TCEM_EACH_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each linked-list item

LL_DMA_TCEM_LAST_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the last linked-list item

Transfer Trigger Mode

LL_DMA_TRIGM_BLK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_RPT_BLK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_SINGLBURST_TRANSFER

A Single/Burst transfer is conditioned by (at least) one hit trigger

Trigger Polarity

LL_DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

LL_DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

LL_DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

Trigger Selection

LL_GPDMA1_TRIGGER_EXTI_LINE0

GPDMA1 HW Trigger signal is EXTI_LINE0

LL_GPDMA1_TRIGGER_EXTI_LINE1

GPDMA1 HW Trigger signal is EXTI_LINE1

LL_GPDMA1_TRIGGER_EXTI_LINE2

GPDMA1 HW Trigger signal is EXTI_LINE2

LL_GPDMA1_TRIGGER_EXTI_LINE3

GPDMA1 HW Trigger signal is EXTI_LINE3

LL_GPDMA1_TRIGGER_EXTI_LINE4

GPDMA1 HW Trigger signal is EXTI_LINE4

LL_GPDMA1_TRIGGER_EXTI_LINE5

GPDMA1 HW Trigger signal is EXTI_LINE5

LL_GPDMA1_TRIGGER_EXTI_LINE6

GPDMA1 HW Trigger signal is EXTI_LINE6

LL_GPDMA1_TRIGGER_EXTI_LINE7

GPDMA1 HW Trigger signal is EXTI_LINE7

LL_GPDMA1_TRIGGER_TAMP_TRG1

GPDMA1 HW Trigger signal is TAMP_TRG1

LL_GPDMA1_TRIGGER_TAMP_TRG2

GPDMA1 HW Trigger signal is TAMP_TRG2

LL_GPDMA1_TRIGGER_TAMP_TRG3

GPDMA1 HW Trigger signal is TAMP_TRG3

LL_GPDMA1_TRIGGER_LPTIM1_CH1

GPDMA1 HW Trigger signal is LPTIM1_CH1

LL_GPDMA1_TRIGGER_LPTIM1_CH2

GPDMA1 HW Trigger signal is LPTIM1_CH2

LL_GPDMA1_TRIGGER_LPTIM2_CH1

GPDMA1 HW Trigger signal is LPTIM2_CH1

LL_GPDMA1_TRIGGER_LPTIM2_CH2

GPDMA1 HW Trigger signal is LPTIM2_CH2

LL_GPDMA1_TRIGGER_RTC_ALRA_TRG

GPDMA1 HW Trigger signal is RTC_ALRA_TRG

LL_GPDMA1_TRIGGER_RTC_ALRB_TRG

GPDMA1 HW Trigger signal is RTC_ALRB_TRG

LL_GPDMA1_TRIGGER_RTC_WUT_TRG

GPDMA1 HW Trigger signal is RTC_WUT_TRG

LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH0_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH0_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH1_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH1_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH2_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH2_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH3_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH3_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH4_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH4_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH5_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH5_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH6_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH6_TCF

LL_GPDMA1_TRIGGER_GPDMA2_CH7_TCF

GPDMA1 HW Trigger signal is GPDMA2_CH7_TCF

LL_GPDMA1_TRIGGER_TIM2_TRGO

GPDMA1 HW Trigger signal is TIM2_TRGO

LL_GPDMA1_TRIGGER_TIM15_TRGO

GPDMA1 HW Trigger signal is TIM15_TRGO

LL_GPDMA1_TRIGGER_TIM12_TRGO

GPDMA1 HW Trigger signal is TIM12_TRGO

LL_GPDMA1_TRIGGER_LPTIM3_CH1

GPDMA1 HW Trigger signal is LPTIM3_CH1

LL_GPDMA1_TRIGGER_LPTIM3_CH2

GPDMA1 HW Trigger signal is LPTIM3_CH2

LL_GPDMA1_TRIGGER_LPTIM4_AIT

GPDMA1 HW Trigger signal is LPTIM4_AIT

LL_GPDMA1_TRIGGER_LPTIM5_CH1

GPDMA1 HW Trigger signal is LPTIM5_CH1

LL_GPDMA1_TRIGGER_LPTIM5_CH2

GPDMA1 HW Trigger signal is LPTIM5_CH2

LL_GPDMA1_TRIGGER_LPTIM6_CH1

GPDMA1 HW Trigger signal is LPTIM6_CH1

LL_GPDMA1_TRIGGER_LPTIM6_CH2

GPDMA1 HW Trigger signal is LPTIM6_CH2

LL_GPDMA2_TRIGGER_EXTI_LINE0

GPDMA2 HW Trigger signal is EXTI_LINE0

LL_GPDMA2_TRIGGER_EXTI_LINE1

GPDMA2 HW Trigger signal is EXTI_LINE1

LL_GPDMA2_TRIGGER_EXTI_LINE2

GPDMA2 HW Trigger signal is EXTI_LINE2

LL_GPDMA2_TRIGGER_EXTI_LINE3

GPDMA2 HW Trigger signal is EXTI_LINE3

LL_GPDMA2_TRIGGER_EXTI_LINE4

GPDMA2 HW Trigger signal is EXTI_LINE4

LL_GPDMA2_TRIGGER_EXTI_LINE5

GPDMA2 HW Trigger signal is EXTI_LINE5

LL_GPDMA2_TRIGGER_EXTI_LINE6

GPDMA2 HW Trigger signal is EXTI_LINE6

LL_GPDMA2_TRIGGER_EXTI_LINE7

GPDMA2 HW Trigger signal is EXTI_LINE7

LL_GPDMA2_TRIGGER_TAMP_TRG1

GPDMA2 HW Trigger signal is TAMP_TRG1

LL_GPDMA2_TRIGGER_TAMP_TRG2

GPDMA2 HW Trigger signal is TAMP_TRG2

LL_GPDMA2_TRIGGER_TAMP_TRG3

GPDMA2 HW Trigger signal is TAMP_TRG3

LL_GPDMA2_TRIGGER_LPTIM1_CH1

GPDMA2 HW Trigger signal is LPTIM1_CH1

LL_GPDMA2_TRIGGER_LPTIM1_CH2

GPDMA2 HW Trigger signal is LPTIM1_CH2

LL_GPDMA2_TRIGGER_LPTIM2_CH1

GPDMA2 HW Trigger signal is LPTIM2_CH1

LL_GPDMA2_TRIGGER_LPTIM2_CH2

GPDMA2 HW Trigger signal is LPTIM2_CH2

LL_GPDMA2_TRIGGER_RTC_ALRA_TRG

GPDMA2 HW Trigger signal is RTC_ALRA_TRG

LL_GPDMA2_TRIGGER_RTC_ALRB_TRG

GPDMA2 HW Trigger signal is RTC_ALRB_TRG

LL_GPDMA2_TRIGGER_RTC_WUT_TRG

GPDMA2 HW Trigger signal is RTC_WUT_TRG

LL_GPDMA2_TRIGGER_GPDMA1_CH0_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH0_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH1_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH1_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH2_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH2_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH3_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH3_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH4_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH4_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH5_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH5_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH6_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH6_TCF

LL_GPDMA2_TRIGGER_GPDMA1_CH7_TCF

GPDMA2 HW Trigger signal is GPDMA1_CH7_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH0_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH0_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH1_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH1_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH2_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH2_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH3_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH3_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH4_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH4_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH5_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH5_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH6_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH6_TCF

LL_GPDMA2_TRIGGER_GPDMA2_CH7_TCF

GPDMA2 HW Trigger signal is GPDMA2_CH7_TCF

LL_GPDMA2_TRIGGER_TIM2_TRGO

GPDMA2 HW Trigger signal is TIM2_TRGO

LL_GPDMA2_TRIGGER_TIM15_TRGO

GPDMA2 HW Trigger signal is TIM15_TRGO

LL_GPDMA2_TRIGGER_TIM12_TRGO

GPDMA2 HW Trigger signal is TIM12_TRGO

LL_GPDMA2_TRIGGER_LPTIM3_CH1

GPDMA2 HW Trigger signal is LPTIM3_CH1

LL_GPDMA2_TRIGGER_LPTIM3_CH2

GPDMA2 HW Trigger signal is LPTIM3_CH2

LL_GPDMA2_TRIGGER_LPTIM4_AIT

GPDMA2 HW Trigger signal is LPTIM4_AIT

LL_GPDMA2_TRIGGER_LPTIM5_CH1

GPDMA2 HW Trigger signal is LPTIM5_CH1

LL_GPDMA2_TRIGGER_LPTIM5_CH2

GPDMA2 HW Trigger signal is LPTIM5_CH2

LL_GPDMA2_TRIGGER_LPTIM6_CH1

GPDMA2 HW Trigger signal is LPTIM6_CH1

LL_GPDMA2_TRIGGER_LPTIM6_CH2

GPDMA2 HW Trigger signal is LPTIM6_CH2

Common Write and Read Registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be written.
- __VALUE__: Value to be written in the register.

Return value:

- None.

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be read.

Return value:

- Register: value.

Convert DMAxChannely

LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Channely into DMAx.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- DMAx.

LL_DMA_GET_CHANNEL

Description:

- Convert DMAx_Channely into LL_DMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- LL_DMA_CHANNEL_y.

LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx.
- __CHANNEL__: LL_DMA_CHANNEL_y.

Return value:

- DMAx_Channely.

Transfer Mode

LL_DMA_NORMAL

Normal DMA transfer

LL_DMA_PFCTRL

HW request peripheral flow control mode

96 LL EXTI Generic Driver

96.1 EXTI Firmware driver registers structures

96.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32h5xx_ll_exti.h

Data Fields

- ***uint32_t Line_0_31***
- ***uint32_t Line_32_63***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_32_63***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 32 to 63 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

96.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

96.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_EnableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_EnableIT_32_63

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_DisableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_DisableIT_32_63

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_IsEnabledIT_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_IsEnabledIT_32_63

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_EnableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_EnableEvent_32_63

LL_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_DisableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_DisableEvent_32_63

LL_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_IsEnabledEvent_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_IsEnabledEvent_32_63

LL_EXTI_EnableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_EnableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_EnableRisingTrig_32_63

LL_EXTI_DisableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_DisableRisingTrig_32_63

LL_EXTI_IsEnabledRisingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_IsEnabledRisingTrig_32_63

LL_EXTI_EnableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_EnableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_EnableFallingTrig_32_63

LL_EXTI_DisableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_DisableFallingTrig_32_63

LL_EXTI_IsEnabledFallingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_IsEnabledFallingTrig_32_63

LL_EXTI_GenerateSWI_0_31

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_C1IMR1, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR1 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR1 register (by writing a 1 into the bit)
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_GenerateSWI_32_63

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_32_63 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR2 register (by writing a 1 into the bit)

Reference Manual to LL API cross reference:

- SWIER2 SWIx LL_EXTI_GenerateSWI_32_63

LL_EXTI_IsActiveFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Falling Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_IsActiveFallingFlag_0_31

LL_EXTI_IsActiveFallingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Falling Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_IsActiveFallingFlag_32_63

LL_EXTI_ReadFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Falling Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ReadFallingFlag_0_31

LL_EXTI_ReadFallingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Falling Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_ReadFallingFlag_32_63

LL_EXTI_ClearFallingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Falling Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ClearFallingFlag_0_31

LL_EXTI_ClearFallingFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Falling Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **None:**

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR2 FPIFx LL_EXTI_ClearFallingFlag_32_63

LL_EXTI_IsActiveRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_IsActiveRisingFlag_0_31

LL_EXTI_IsActiveRisingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_IsActiveRisingFlag_32_63

LL_EXTI_ReadRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ReadRisingFlag_0_31

LL_EXTI_ReadRisingFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_ReadRisingFlag_32_63

LL_EXTI_ClearRisingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16

Return values

- **None:**

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ClearRisingFlag_0_31

LL_EXTI_ClearRisingFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_53

Return values

- **None:**

Notes

- This bit is set when the rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR2 RPIFx LL_EXTI_ClearRisingFlag_32_63

LL_EXTI_SetEXTISource

Function name

__STATIC_INLINE void LL_EXTI_SetEXTISource (uint32_t Port, uint32_t Line)

Function description

Configure source input for the EXTI external interrupt.

Parameters

- **Port:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_PORTA
  - LL_EXTI_EXTI_PORTB
  - LL_EXTI_EXTI_PORTC
  - LL_EXTI_EXTI_PORTD
  - LL_EXTI_EXTI_PORTE
  - LL_EXTI_EXTI_PORTF
  - LL_EXTI_EXTI_PORTG
  - LL_EXTI_EXTI_PORTH
  - LL_EXTI_EXTI_PORTI

  (*) value not defined in all devices
- **Line:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_LINE0
  - LL_EXTI_EXTI_LINE1
  - LL_EXTI_EXTI_LINE2
  - LL_EXTI_EXTI_LINE3
  - LL_EXTI_EXTI_LINE4
  - LL_EXTI_EXTI_LINE5
  - LL_EXTI_EXTI_LINE6
  - LL_EXTI_EXTI_LINE7
  - LL_EXTI_EXTI_LINE8
  - LL_EXTI_EXTI_LINE9
  - LL_EXTI_EXTI_LINE10
  - LL_EXTI_EXTI_LINE11
  - LL_EXTI_EXTI_LINE12
  - LL_EXTI_EXTI_LINE13
  - LL_EXTI_EXTI_LINE14
  - LL_EXTI_EXTI_LINE15

Return values

- **None:**

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_SetEXTISource

LL_EXTI_GetEXTISource

Function name

__STATIC_INLINE uint32_t LL_EXTI_GetEXTISource (uint32_t Line)

Function description

Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_LINE0
  - LL_EXTI_EXTI_LINE1
  - LL_EXTI_EXTI_LINE2
  - LL_EXTI_EXTI_LINE3
  - LL_EXTI_EXTI_LINE4
  - LL_EXTI_EXTI_LINE5
  - LL_EXTI_EXTI_LINE6
  - LL_EXTI_EXTI_LINE7
  - LL_EXTI_EXTI_LINE8
  - LL_EXTI_EXTI_LINE9
  - LL_EXTI_EXTI_LINE10
  - LL_EXTI_EXTI_LINE11
  - LL_EXTI_EXTI_LINE12
  - LL_EXTI_EXTI_LINE13
  - LL_EXTI_EXTI_LINE14
  - LL_EXTI_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_EXTI_PORTA
  - LL_EXTI_EXTI_PORTB
  - LL_EXTI_EXTI_PORTC
  - LL_EXTI_EXTI_PORTD
  - LL_EXTI_EXTI_PORTE
  - LL_EXTI_EXTI_PORTF
  - LL_EXTI_EXTI_PORTG
  - LL_EXTI_EXTI_PORTH
  - LL_EXTI_EXTI_PORTI

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_GetEXTISource

LL_EXTI_EnableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_EnableSecure_0_31

LL_EXTI_EnableSecure_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_EnableSecure_32_63

LL_EXTI_DisableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_DisableSecure_0_31

LL_EXTI_DisableSecure_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_DisableSecure_32_63

LL_EXTI_IsEnabledSecure_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_IsEnabledSecure_0_31

LL_EXTI_IsEnabledSecure_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR2 SECx LL_EXTI_IsEnabledSecure_32_63

LL_EXTI_EnablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_EnablePrivilege_0_31

LL_EXTI_EnablePrivilege_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_EnablePrivilege_32_63

LL_EXTI_DisablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_DisablePrivilege_0_31

LL_EXTI_DisablePrivilege_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_EnablePrivilege_32_63

LL_EXTI_IsEnabledPrivilege_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_IsEnabledPrivilege_0_31

LL_EXTI_IsEnabledPrivilege_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR2 PRIVx LL_EXTI_IsEnabledPrivilege_32_63

LL_EXTI_EnableLockAttribute

Function name

__STATIC_INLINE void LL_EXTI_EnableLockAttribute (void)

Function description

Enable the EXTI lock attributes.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_EnableLockAttribute

LL_EXTI_IsEnabledLockAttribute

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledLockAttribute (void)

Function description

Check if EXTI attributes are locked.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_IsEnabledLockAttribute

LL_EXTI_Init

Function name

ErrorStatus LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters

- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are initialized
  - ERROR: not applicable

LL_EXTI_DeInit

Function name

ErrorStatus LL_EXTI_DeInit (void)

Function description

De-initialize the EXTI registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are de-initialized
  - ERROR: not applicable

LL_EXTI_StructInit

Function name

void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Set each LL_EXTI_InitTypeDef field to default value.

Parameters

- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **None:**

96.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

96.3.1 EXTI

EXTI

LINE

LL_EXTI_LINE_0

Extended line 0

LL_EXTI_LINE_1

Extended line 1

LL_EXTI_LINE_2

Extended line 2

LL_EXTI_LINE_3

Extended line 3

LL_EXTI_LINE_4

Extended line 4

LL_EXTI_LINE_5

Extended line 5

LL_EXTI_LINE_6

Extended line 6

LL_EXTI_LINE_7

Extended line 7

LL_EXTI_LINE_8

Extended line 8

LL_EXTI_LINE_9

Extended line 9

LL_EXTI_LINE_10

Extended line 10

LL_EXTI_LINE_11

Extended line 11

LL_EXTI_LINE_12

Extended line 12

LL_EXTI_LINE_13

Extended line 13

LL_EXTI_LINE_14

Extended line 14

LL_EXTI_LINE_15

Extended line 15

LL_EXTI_LINE_16

Extended line 16

LL_EXTI_LINE_17

Extended line 17

LL_EXTI_LINE_18

Extended line 18

LL_EXTI_LINE_19

Extended line 19

LL_EXTI_LINE_20

Extended line 20

LL_EXTI_LINE_21

Extended line 21

LL_EXTI_LINE_22

Extended line 22

LL_EXTI_LINE_23

Extended line 23

LL_EXTI_LINE_24

Extended line 24

LL_EXTI_LINE_25

Extended line 25

LL_EXTI_LINE_26

Extended line 26

LL_EXTI_LINE_27

Extended line 27

LL_EXTI_LINE_28

Extended line 28

LL_EXTI_LINE_29

Extended line 29

LL_EXTI_LINE_30

Extended line 30

LL_EXTI_LINE_31

Extended line 31

LL_EXTI_LINE_ALL_0_31

All Extended line not reserved

LL_EXTI_LINE_32

Extended line 32

LL_EXTI_LINE_33

Extended line 33

LL_EXTI_LINE_34

Extended line 34

LL_EXTI_LINE_35

Extended line 35

LL_EXTI_LINE_36

Extended line 36

LL_EXTI_LINE_37

Extended line 37

LL_EXTI_LINE_38

Extended line 38

LL_EXTI_LINE_39

Extended line 39

LL_EXTI_LINE_40

Extended line 40

LL_EXTI_LINE_41

Extended line 41

LL_EXTI_LINE_42

Extended line 42

LL_EXTI_LINE_43

Extended line 43

LL_EXTI_LINE_44

Extended line 44

LL_EXTI_LINE_45

Extended line 45

LL_EXTI_LINE_46

Extended line 46

LL_EXTI_LINE_47

Extended line 47

LL_EXTI_LINE_48

Extended line 48

LL_EXTI_LINE_49

Extended line 49

LL_EXTI_LINE_50

Extended line 50

LL_EXTI_LINE_51

Extended line 51

LL_EXTI_LINE_52

Extended line 52

LL_EXTI_LINE_53

Extended line 53

LL_EXTI_LINE_54

Extended line 54

LL_EXTI_LINE_55

Extended line 55

LL_EXTI_LINE_56

Extended line 56

LL_EXTI_LINE_57

Extended line 57

LL_EXTI_LINE_ALL_32_63

ALL Extended line

LL_EXTI_LINE_ALL

All Extended line

LL_EXTI_LINE_NONE

None Extended line

Mode

LL_EXTI_MODE_IT

Interrupt Mode

LL_EXTI_MODE_EVENT

Event Mode

LL_EXTI_MODE_IT_EVENT

Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE

No Trigger Mode

LL_EXTI_TRIGGER_RISING

Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING

Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING

Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

97 LL FMAC Generic Driver

97.1 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

97.1.1 Detailed description of functions

LL_FMAC_SetX1FullWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetX1FullWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure X1 full watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_SetX1FullWatermark

LL_FMAC_GetX1FullWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetX1FullWatermark (const FMAC_TypeDef * FMACx)

Function description

Return X1 full watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_GetX1FullWatermark

LL_FMAC_SetX1BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX1BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X1 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_SetX1BufferSize

LL_FMAC_GetX1BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1BufferSize (const FMAC_TypeDef * FMACx)

Function description

Return X1 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the input buffer (including the optional "headroom") (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_GetX1BufferSize

LL_FMAC_SetX1Base

Function name

__STATIC_INLINE void LL_FMAC_SetX1Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X1 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_SetX1Base

LL_FMAC_GetX1Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1Base (const FMAC_TypeDef * FMACx)

Function description

Return X1 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the input buffer (X1) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_GetX1Base

LL_FMAC_SetX2BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX2BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X2 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_SetX2BufferSize

LL_FMAC_GetX2BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2BufferSize (const FMAC_TypeDef * FMACx)

Function description

Return X2 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the coefficient buffer (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_GetX2BufferSize

LL_FMAC_SetX2Base

Function name

__STATIC_INLINE void LL_FMAC_SetX2Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X2 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_SetX2Base

LL_FMAC_GetX2Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2Base (const FMAC_TypeDef * FMACx)

Function description

Return X2 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the coefficient buffer (X2) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_GetX2Base

LL_FMAC_SetYEmptyWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetYEmptyWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure Y empty watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_SetYEmptyWatermark

LL_FMAC_GetYEmptyWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetYEmptyWatermark (const FMAC_TypeDef * FMACx)

Function description

Return Y empty watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_GetYEmptyWatermark

LL_FMAC_SetYBufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetYBufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure Y buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_SetYBufferSize

LL_FMAC_GetYBufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBufferSize (const FMAC_TypeDef * FMACx)

Function description

Return Y buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the output buffer (including the optional "headroom" - value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_GetYBufferSize

LL_FMAC_SetYBase

Function name

__STATIC_INLINE void LL_FMAC_SetYBase (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure Y base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_SetYBase

LL_FMAC_GetYBase

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBase (const FMAC_TypeDef * FMACx)

Function description

Return Y base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the output buffer (Y) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_GetYBase

LL_FMAC_EnableStart

Function name

__STATIC_INLINE void LL_FMAC_EnableStart (FMAC_TypeDef * FMACx)

Function description

Start FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_EnableStart

LL_FMAC_DisableStart

Function name

__STATIC_INLINE void LL_FMAC_DisableStart (FMAC_TypeDef * FMACx)

Function description

Stop FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_DisableStart

LL_FMAC_IsEnabledStart

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledStart (const FMAC_TypeDef * FMACx)

Function description

Check the state of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_IsEnabledStart

LL_FMAC_SetFunction

Function name

__STATIC_INLINE void LL_FMAC_SetFunction (FMAC_TypeDef * FMACx, uint32_t Function)

Function description

Configure function.

Parameters

- **FMACx:** FMAC instance
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_SetFunction

LL_FMAC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetFunction (const FMAC_TypeDef * FMACx)

Function description

Return function.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_GetFunction

LL_FMAC_SetParamR

Function name

__STATIC_INLINE void LL_FMAC_SetParamR (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter R.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_SetParamR

LL_FMAC_GetParamR

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamR (const FMAC_TypeDef * FMACx)

Function description

Return input parameter R.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter R (gain, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_GetParamR

LL_FMAC_SetParamQ

Function name

__STATIC_INLINE void LL_FMAC_SetParamQ (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter Q.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_SetParamQ

LL_FMAC_GetParamQ

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamQ (const FMAC_TypeDef * FMACx)

Function description

Return input parameter Q.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter Q (vector length, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_GetParamQ

LL_FMAC_SetParamP

Function name

__STATIC_INLINE void LL_FMAC_SetParamP (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter P.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_SetParamP

LL_FMAC_GetParamP

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamP (const FMAC_TypeDef * FMACx)

Function description

Return input parameter P.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter P (vector length, number of filter taps, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_GetParamP

LL_FMAC_EnableReset

Function name

__STATIC_INLINE void LL_FMAC_EnableReset (FMAC_TypeDef * FMACx)

Function description

Start the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_EnableReset

LL_FMAC_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledReset (const FMAC_TypeDef * FMACx)

Function description

Check the state of the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_IsEnabledReset

LL_FMAC_EnableClipping

Function name

__STATIC_INLINE void LL_FMAC_EnableClipping (FMAC_TypeDef * FMACx)

Function description

Enable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_EnableClipping

LL_FMAC_DisableClipping

Function name

__STATIC_INLINE void LL_FMAC_DisableClipping (FMAC_TypeDef * FMACx)

Function description

Disable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_DisableClipping

LL_FMAC_IsEnabledClipping

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledClipping (const FMAC_TypeDef * FMACx)

Function description

Check Clipping State.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_IsEnabledClipping

LL_FMAC_EnableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_EnableDMAReq_WRITE

LL_FMAC_DisableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_DisableDMAReq_WRITE

LL_FMAC_IsEnabledDMAReq_WRITE

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_WRITE (const FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA write channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_IsEnabledDMAReq_WRITE

LL_FMAC_EnableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_EnableDMAReq_READ

LL_FMAC_DisableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_DisableDMAReq_READ

LL_FMAC_IsEnabledDMAReq_READ

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_READ (const FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA read channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_IsEnabledDMAReq_READ

LL_FMAC_EnableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Enable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_EnableIT_SAT

LL_FMAC_DisableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Disable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_DisableIT_SAT

LL_FMAC_IsEnabledIT_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_SAT (const FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_IsEnabledIT_SAT

LL_FMAC_EnableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_EnableIT_UNFL

LL_FMAC_DisableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_DisableIT_UNFL

LL_FMAC_IsEnabledIT_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_UNFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_IsEnabledIT_UNFL

LL_FMAC_EnableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_EnableIT_OVFL

LL_FMAC_DisableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_DisableIT_OVFL

LL_FMAC_IsEnabledIT_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_OVFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_IsEnabledIT_OVFL

LL_FMAC_EnableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_WR (FMAC_TypeDef * FMACx)

Function description

Enable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_EnableIT_WR

LL_FMAC_DisableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_WR (FMAC_TypeDef * FMACx)

Function description

Disable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_DisableIT_WR

LL_FMAC_IsEnabledIT_WR

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_WR (const FMAC_TypeDef * FMACx)

Function description

Check FMAC write interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_IsEnabledIT_WR

LL_FMAC_EnableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_RD (FMAC_TypeDef * FMACx)

Function description

Enable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_EnableIT_RD

LL_FMAC_DisableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_RD (FMAC_TypeDef * FMACx)

Function description

Disable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_DisableIT_RD

LL_FMAC_IsEnabledIT_RD

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_RD (const FMAC_TypeDef * FMACx)

Function description

Check FMAC read interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_IsEnabledIT_RD

LL_FMAC_IsActiveFlag_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_SAT (const FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SAT LL_FMAC_IsActiveFlag_SAT

LL_FMAC_IsActiveFlag_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_UNFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UNFL LL_FMAC_IsActiveFlag_UNFL

LL_FMAC_IsActiveFlag_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_OVFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVFL LL_FMAC_IsActiveFlag_OVFL

LL_FMAC_IsActiveFlag_X1FULL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_X1FULL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC X1 buffer full flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR X1FULL LL_FMAC_IsActiveFlag_X1FULL

LL_FMAC_IsActiveFlag_YEMPTY

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_YEMPTY (const FMAC_TypeDef * FMACx)

Function description

Check FMAC Y buffer empty flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR YEMPTY LL_FMAC_IsActiveFlag_YEMPTY

LL_FMAC_WriteData

Function name

__STATIC_INLINE void LL_FMAC_WriteData (FMAC_TypeDef * FMACx, uint16_t InData)

Function description

Write 16-bit input data for the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **InData:** 16-bit value to be provided as input data for FMAC processing. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA WDATA LL_FMAC_WriteData

LL_FMAC_ReadData

Function name

__STATIC_INLINE uint16_t LL_FMAC_ReadData (const FMAC_TypeDef * FMACx)

Function description

Return 16-bit output data of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint16_t:** 16-bit output data of FMAC processing (value between Min_Data=0x0000 and Max_Data=0xFFFF).

Reference Manual to LL API cross reference:

- RDATA RDATA LL_FMAC_ReadData

LL_FMAC_ConfigX1

Function name

__STATIC_INLINE void LL_FMAC_ConfigX1 (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X1 buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_ConfigX1
- X1BUFCFG X1_BASE LL_FMAC_ConfigX1
- X1BUFCFG X1_BUF_SIZE LL_FMAC_ConfigX1

LL_FMAC_ConfigX2

Function name

__STATIC_INLINE void LL_FMAC_ConfigX2 (FMAC_TypeDef * FMACx, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X2 buffer.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_ConfigX2
- X2BUFCFG X2_BUF_SIZE LL_FMAC_ConfigX2

LL_FMAC_ConfigY

Function name

__STATIC_INLINE void LL_FMAC_ConfigY (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for Y buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_ConfigY
- YBUFCFG Y_BASE LL_FMAC_ConfigY
- YBUFCFG Y_BUF_SIZE LL_FMAC_ConfigY

LL_FMAC_ConfigFunc

Function name

__STATIC_INLINE void LL_FMAC_ConfigFunc (FMAC_TypeDef * FMACx, uint8_t Start, uint32_t Function, uint8_t ParamP, uint8_t ParamQ, uint8_t ParamR)

Function description

Configure the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **Start:** This parameter can be one of the following values:
  - LL_FMAC_PROCESSING_STOP
  - LL_FMAC_PROCESSING_START
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1
- **ParamP:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamQ:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamR:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_ConfigFunc
- PARAM FUNC LL_FMAC_ConfigFunc
- PARAM P LL_FMAC_ConfigFunc
- PARAM Q LL_FMAC_ConfigFunc
- PARAM R LL_FMAC_ConfigFunc

LL_FMAC_Init

Function name

ErrorStatus LL_FMAC_Init (FMAC_TypeDef * FMACx)

Function description

Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **ErrorStatus:** enumeration value:
  - SUCCESS: FMAC registers are initialized
  - ERROR: FMAC registers are not initialized

LL_FMAC_DeInit

Function name

ErrorStatus LL_FMAC_DeInit (const FMAC_TypeDef * FMACx)

Function description

De-Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: FMAC registers are de-initialized
  - ERROR: FMAC registers are not de-initialized

97.2 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

97.2.1 FMAC

FMAC

FMAC functions

LL_FMAC_FUNC_LOAD_X1

Load X1 buffer

LL_FMAC_FUNC_LOAD_X2

Load X2 buffer

LL_FMAC_FUNC_LOAD_Y

Load Y buffer

LL_FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

LL_FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

Get Flag Defines

LL_FMAC_SR_SAT

Saturation Error Flag (this helps in debugging a filter)

LL_FMAC_SR_UNFL

Underflow Error Flag

LL_FMAC_SR_OVFL

Overflow Error Flag

LL_FMAC_SR_X1FULL

X1 Buffer Full Flag

LL_FMAC_SR_YEMPTY

Y Buffer Empty Flag

IT Defines

LL_FMAC_CR_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

LL_FMAC_CR_UNFLIEN

Underflow Error Interrupt Enable

LL_FMAC_CR_OVFLIEN

Overflow Error Interrupt Enable

LL_FMAC_CR_WIEN

Write Interrupt Enable

LL_FMAC_CR_RIEN

Read Interrupt Enable

FMAC processing

LL_FMAC_PROCESSING_STOP

Stop FMAC Processing

LL_FMAC_PROCESSING_START

Start FMAC Processing

FMAC watermarks

LL_FMAC_WM_0_THRESHOLD_1

Buffer full/empty flag set if there is less than 1 free/unread space.

LL_FMAC_WM_1_THRESHOLD_2

Buffer full/empty flag set if there are less than 2 free/unread spaces.

LL_FMAC_WM_2_THRESHOLD_4

Buffer full/empty flag set if there are less than 4 free/unread spaces.

LL_FMAC_WM_3_THRESHOLD_8

Buffer full/empty flag set if there are less than 8 free/empty spaces.

Common Write and read registers Macros

LL_FMAC_WriteReg

Description:

- Write a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_FMAC_ReadReg

Description:

- Read a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be read

Return value:

- Register: value

98 LL GPIO Generic Driver

98.1 GPIO Firmware driver registers structures

98.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32h5xx_ll_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***
  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***
  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***
  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

98.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

98.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name

__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name

__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)

Function description

Configure gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Return values

- **None:**

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name

__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description

Configure gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values

- **None:**

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name

__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description

Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Notes

- Possible values are from AF0 to AF15 depending on target.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (const GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **GPIOx:** GPIO Port
- **PortValue:** Level value for each pin of the port

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name

__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_TogglePin

LL_GPIO_EnableHighSPeedLowVoltage

Function name

__STATIC_INLINE void LL_GPIO_EnableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_EnableHighSPeedLowVoltage

LL_GPIO_DisableHighSPeedLowVoltage

Function name

__STATIC_INLINE void LL_GPIO_DisableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_DisableHighSPeedLowVoltage

LL_GPIO_IsEnabledHighSPeedLowVoltage

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledHighSPeedLowVoltage (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if speed optimization for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **State:** of bit (1 or 0).

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_IsEnabledHighSPeedLowVoltage

LL_GPIO_EnablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_EnablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_EnablePinSecure

LL_GPIO_DisablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_DisablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_DisablePinSecure

LL_GPIO_IsEnabledPinSecure

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledPinSecure (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if secure write only access for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_IsEnabledPinSecure

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (const GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  - ERROR: Not applicable

LL_GPIO_StructInit

Function name

void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Set each LL_GPIO_InitTypeDef field to default value.

Parameters

- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

98.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

98.3.1 GPIO

GPIO

Alternate Function

LL_GPIO_AF_0

Select alternate function 0

LL_GPIO_AF_1

Select alternate function 1

LL_GPIO_AF_2

Select alternate function 2

LL_GPIO_AF_3

Select alternate function 3

LL_GPIO_AF_4

Select alternate function 4

LL_GPIO_AF_5

Select alternate function 5

LL_GPIO_AF_6

Select alternate function 6

LL_GPIO_AF_7

Select alternate function 7

LL_GPIO_AF_8

Select alternate function 8

LL_GPIO_AF_9

Select alternate function 9

LL_GPIO_AF_10

Select alternate function 10

LL_GPIO_AF_11

Select alternate function 11

LL_GPIO_AF_12

Select alternate function 12

LL_GPIO_AF_13

Select alternate function 13

LL_GPIO_AF_14

Select alternate function 14

LL_GPIO_AF_15

Select alternate function 15

Mode

LL_GPIO_MODE_INPUT

Select input mode

LL_GPIO_MODE_OUTPUT

Select output mode

LL_GPIO_MODE_ALTERNATE

Select alternate function mode

LL_GPIO_MODE_ANALOG

Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL

Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN

Select open-drain as output type

PIN

LL_GPIO_PIN_0

Select pin 0

LL_GPIO_PIN_1

Select pin 1

LL_GPIO_PIN_2

Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6

Select pin 6

LL_GPIO_PIN_7

Select pin 7

LL_GPIO_PIN_8

Select pin 8

LL_GPIO_PIN_9

Select pin 9

LL_GPIO_PIN_10

Select pin 10

LL_GPIO_PIN_11

Select pin 11

LL_GPIO_PIN_12

Select pin 12

LL_GPIO_PIN_13

Select pin 13

LL_GPIO_PIN_14

Select pin 14

LL_GPIO_PIN_15

Select pin 15

LL_GPIO_PIN_ALL

Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO

Select I/O no pull

LL_GPIO_PULL_UP

Select I/O pull up

LL_GPIO_PULL_DOWN

Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW

Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM

Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH

Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH

Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

99 LL I2C Generic Driver

99.1 I2C Firmware driver registers structures

99.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32h5xx_ll_i2c.h

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***
  Specifies the peripheral mode. This parameter can be a value of ***I2C_LL_EC_PERIPHERAL_MODE***.This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***
  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**.This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***
  Enables or disables analog noise filter. This parameter can be a value of ***I2C_LL_EC_ANALOGFILTER_SELECTION***.This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***
  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0F.This feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FF.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of ***I2C_LL_EC_I2C_ACKNOWLEDGE***.This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of ***I2C_LL_EC_OWNADDRESS1***.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

99.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

99.2.1 Detailed description of functions

LL_I2C_Enable

Function name

__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)

Function description

Enable I2C peripheral (PE = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name

__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)

Function description

Disable I2C peripheral (PE = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C peripheral is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name

__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)

Function description

Configure Noise Filters (Analog and Digital).

Parameters

- **I2Cx:** I2C Instance.
- **AnalogFilter:** This parameter can be one of the following values:
  - LL_I2C_ANALOGFILTER_ENABLE
  - LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_ConfigFilters
- CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name

__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)

Function description

Configure Digital Noise Filter.

Parameters

- **I2Cx:** I2C Instance.
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (const I2C_TypeDef * I2Cx)

Function description

Get the current Digital Noise Filter configuration.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Enable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Disable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (const I2C_TypeDef * I2Cx)

Function description

Check if Analog Noise Filter is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Enable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Disable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA transmission requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Enable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Disable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA reception requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (const I2C_TypeDef * I2Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I2Cx:** I2C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I2C_DMA_REG_DATA_TRANSMIT
  - LL_I2C_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_DMA_GetRegAddr
- RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name

__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)

Function description

Enable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name

__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)

Function description

Disable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (const I2C_TypeDef * I2Cx)

Function description

Check if Clock stretching is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Enable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Disable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (const I2C_TypeDef * I2Cx)

Function description

Check if hardware byte control in slave mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Enable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
- This bit can only be programmed when Digital Filter is disabled.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Disable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (const I2C_TypeDef * I2Cx)

Function description

Check if Wakeup from STOP is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Enable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Disable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (const I2C_TypeDef * I2Cx)

Function description

Check if General Call is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_EnableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_EnableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Enable I2C Fast Mode Plus (FMP = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive enable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_EnableFastModePlus

LL_I2C_DisableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_DisableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Disable I2C Fast Mode Plus (FMP = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive disable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_DisableFastModePlus

LL_I2C_IsEnabledFastModePlus

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledFastModePlus (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C Fast Mode Plus is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_IsEnabledFastModePlus

LL_I2C_EnableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_EnableAutoClearFlag_ADDR

LL_I2C_DisableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_DisableAutoClearFlag_ADDR

LL_I2C_IsEnabledAutoClearFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of ADDR flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_IsEnabledAutoClearFlag_ADDR

LL_I2C_EnableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_EnableAutoClearFlag_STOP

LL_I2C_DisableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_DisableAutoClearFlag_STOP

LL_I2C_IsEnabledAutoClearFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of STOP flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_IsEnabledAutoClearFlag_STOP

LL_I2C_SetMasterAddressingMode

Function name

__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)

Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

Parameters

- **I2Cx:** I2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Return values

- **None:**

Notes

- Changing this bit is not allowed, when the START bit is set.

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (const I2C_TypeDef * I2Cx)

Function description

Get the Master addressing mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)

Function description

Set the Own Address1.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF.
- **OwnAddrSize:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS1_7BIT
  - LL_I2C_OWNADDRESS1_10BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1 LL_I2C_SetOwnAddress1
- OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)

Function description

Set the 7bits Own Address2.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F.
- **OwnAddrMask:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS2_NOMASK
  - LL_I2C_OWNADDRESS2_MASK01
  - LL_I2C_OWNADDRESS2_MASK02
  - LL_I2C_OWNADDRESS2_MASK03
  - LL_I2C_OWNADDRESS2_MASK04
  - LL_I2C_OWNADDRESS2_MASK05
  - LL_I2C_OWNADDRESS2_MASK06
  - LL_I2C_OWNADDRESS2_MASK07

Return values

- **None:**

Notes

- This action has no effect if own address2 is enabled.

Reference Manual to LL API cross reference:

- OAR2 OA2 LL_I2C_SetOwnAddress2
- OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name

__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)

Function description

Configure the SDA setup, hold time and the SCL high, low period.

Parameters

- **I2Cx:** I2C Instance.
- **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (const I2C_TypeDef * I2Cx)

Function description

Get the Timing Prescaler setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL low period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL high period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA hold time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA setup time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name

__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I2Cx:** I2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_SetMode
- CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMode (const I2C_TypeDef * I2Cx)

Function description

Get peripheral mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_GetMode
- CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Enable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Disable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Enable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Disable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)

Function description

Configure the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
- **TimeoutB:**

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/ orTimeoutB).

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)

Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)

Function description

Set the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)

Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Extended Cumulative Clock TimeoutB setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Enable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Disable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (const I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Check if the SMBus Clock Timeout is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)

Function description

Enable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)

Function description

Disable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (const I2C_TypeDef * I2Cx)

Function description

Check if the TXIS Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)

Function description

Enable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)

Function description

Disable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (const I2C_TypeDef * I2Cx)

Function description

Check if the RXNE Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if Address match interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Enable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Disable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (const I2C_TypeDef * I2Cx)

Function description

Check if Not acknowledge received interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Enable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Disable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if STOP detection interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)

Function description

Enable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)

Function description

Disable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (const I2C_TypeDef * I2Cx)

Function description

Check if Transfer Complete interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Enable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Disable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (const I2C_TypeDef * I2Cx)

Function description

Check if Error interrupts are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit data register empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit interrupt flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Receive data register not empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.

Reference Manual to LL API cross reference:

- ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Address matched flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.

Reference Manual to LL API cross reference:

- ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Not Acknowledge received flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a NACK is received after a byte transmission.

Reference Manual to LL API cross reference:

- ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When arbitration lost.

Reference Manual to LL API cross reference:

- ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Overrun/Underrun flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).

Reference Manual to LL API cross reference:

- ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus PEC error flag in reception.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.

Reference Manual to LL API cross reference:

- ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.

Reference Manual to LL API cross reference:

- ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.

Reference Manual to LL API cross reference:

- ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus Busy flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Start condition is detected.

Reference Manual to LL API cross reference:

- ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Clear Address Matched flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Clear Not Acknowledge flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Clear Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Clear Transmit data register empty flag (TXE).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can be clear by software in order to flush the transmit data register (TXDR).

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Clear Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Clear Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Clear Overrun/Underrun flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Clear SMBus PEC error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Enable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Disable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (const I2C_TypeDef * I2Cx)

Function description

Check if automatic STOP condition is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name

__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)

Function description

Enable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name

__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)

Function description

Disable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (const I2C_TypeDef * I2Cx)

Function description

Check if reload mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name

__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)

Function description

Configure the number of bytes for transfer.

Parameters

- **I2Cx:** I2C Instance.
- **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (const I2C_TypeDef * I2Cx)

Function description

Get the number of bytes configured for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name

__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)

Function description

Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.

Parameters

- **I2Cx:** I2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  - LL_I2C_ACK
  - LL_I2C_NACK

Return values

- **None:**

Notes

- Usage in Slave mode only.

Reference Manual to LL API cross reference:

- CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)

Function description

Generate a START or RESTART condition.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.

Reference Manual to LL API cross reference:

- CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)

Function description

Generate a STOP condition after the current byte transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master only sends the first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (const I2C_TypeDef * I2Cx)

Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name

__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)

Function description

Configure the transfer direction (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (const I2C_TypeDef * I2Cx)

Function description

Get the transfer direction requested (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name

__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)

Function description

Configure the slave address for transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (const I2C_TypeDef * I2Cx)

Function description

Get the slave address programmed for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name

__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)

Function description

Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  - LL_I2C_ADDRSLAVE_7BIT
  - LL_I2C_ADDRSLAVE_10BIT
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255.
- **EndMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_RELOAD
  - LL_I2C_MODE_AUTOEND
  - LL_I2C_MODE_SOFTEND
  - LL_I2C_MODE_SMBUS_RELOAD
  - LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC
  - LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC
- **Request:** This parameter can be one of the following values:
  - LL_I2C_GENERATE_NOSTARTSTOP
  - LL_I2C_GENERATE_STOP
  - LL_I2C_GENERATE_START_READ
  - LL_I2C_GENERATE_START_WRITE
  - LL_I2C_GENERATE_RESTART_7BIT_READ
  - LL_I2C_GENERATE_RESTART_7BIT_WRITE
  - LL_I2C_GENERATE_RESTART_10BIT_READ
  - LL_I2C_GENERATE_RESTART_10BIT_WRITE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_HandleTransfer
- CR2 ADD10 LL_I2C_HandleTransfer
- CR2 RD_WRN LL_I2C_HandleTransfer
- CR2 START LL_I2C_HandleTransfer
- CR2 STOP LL_I2C_HandleTransfer
- CR2 RELOAD LL_I2C_HandleTransfer
- CR2 NBYTES LL_I2C_HandleTransfer
- CR2 AUTOEND LL_I2C_HandleTransfer
- CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (const I2C_TypeDef * I2Cx)

Function description

Indicate the value of transfer direction (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_DIRECTION_WRITE
  - LL_I2C_DIRECTION_READ

Notes

- RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.

Reference Manual to LL API cross reference:

- ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (const I2C_TypeDef * I2Cx)

Function description

Return the slave matched address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (const I2C_TypeDef * I2Cx)

Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Packet Error byte calculated.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (const I2C_TypeDef * I2Cx)

Function description

Read Receive Data register.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name

__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)

Function description

Write in Transmit Data Register .

Parameters

- **I2Cx:** I2C Instance.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name

ErrorStatus LL_I2C_Init (I2C_TypeDef * I2Cx, const LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Initialize the I2C registers according to the specified parameters in I2C_InitStruct.

Parameters

- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are initialized
  - ERROR: Not applicable

LL_I2C_DeInit

Function name

ErrorStatus LL_I2C_DeInit (const I2C_TypeDef * I2Cx)

Function description

De-initialize the I2C registers to their default reset values.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are de-initialized
  - ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name

void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Set each LL_I2C_InitTypeDef field to default value.

Parameters

- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values

- **None:**

99.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

99.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT

Master operates in 7-bit addressing mode.

LL_I2C_ADDRESSING_MODE_10BIT

Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT

Slave Address in 7-bit.

LL_I2C_ADDRSLAVE_10BIT

Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE

Analog filter is enabled.

LL_I2C_ANALOGFILTER_DISABLE

Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF

Address Matched flag

LL_I2C_ICR_NACKCF

Not Acknowledge flag

LL_I2C_ICR_STOPCF

Stop detection flag

LL_I2C_ICR_BERRCF

Bus error flag

LL_I2C_ICR_ARLOCF

Arbitration Lost flag

LL_I2C_ICR_OVRCF

Overrun/Underrun flag

LL_I2C_ICR_PECCF

PEC error flag

LL_I2C_ICR_TIMOUTCF

Timeout detection flag

LL_I2C_ICR_ALERTCF

Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE

Write transfer request by master, slave enters receiver mode.

LL_I2C_DIRECTION_READ

Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_I2C_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP

Don't Generate Stop and Start condition.

LL_I2C_GENERATE_STOP

Generate Stop condition (Size should be set to 0).

LL_I2C_GENERATE_START_READ

Generate Start for read request.

LL_I2C_GENERATE_START_WRITE

Generate Start for write request.

LL_I2C_GENERATE_RESTART_7BIT_READ

Generate Restart for read request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_7BIT_WRITE

Generate Restart for write request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_10BIT_READ

Generate Restart for read request, slave 10Bit address.

LL_I2C_GENERATE_RESTART_10BIT_WRITE

Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE

Transmit data register empty

LL_I2C_ISR_TXIS

Transmit interrupt status

LL_I2C_ISR_RXNE

Receive data register not empty

LL_I2C_ISR_ADDR

Address matched (slave mode)

LL_I2C_ISR_NACKF

Not Acknowledge received flag

LL_I2C_ISR_STOPF

Stop detection flag

LL_I2C_ISR_TC

Transfer Complete (master mode)

LL_I2C_ISR_TCR

Transfer Complete Reload

LL_I2C_ISR_BERR

Bus error

LL_I2C_ISR_ARLO

Arbitration lost

LL_I2C_ISR_OVR

Overrun/Underrun (slave mode)

LL_I2C_ISR_PECERR

PEC Error in reception (SMBus mode)

LL_I2C_ISR_TIMEOUT

Timeout detection flag (SMBus mode)

LL_I2C_ISR_ALERT

SMBus alert (SMBus mode)

LL_I2C_ISR_BUSY

Bus busy

Acknowledge Generation

LL_I2C_ACK

ACK is sent after current received byte.

LL_I2C_NACK

NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE

TX Interrupt enable

LL_I2C_CR1_RXIE

RX Interrupt enable

LL_I2C_CR1_ADDRIE

Address match Interrupt enable (slave only)

LL_I2C_CR1_NACKIE

Not acknowledge received Interrupt enable

LL_I2C_CR1_STOPIE

STOP detection Interrupt enable

LL_I2C_CR1_TCIE

Transfer Complete interrupt enable

LL_I2C_CR1_ERRIE

Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD

Enable I2C Reload mode.

LL_I2C_MODE_AUTOEND

Enable I2C Automatic end mode with no HW PEC comparison.

LL_I2C_MODE_SOFTEND

Enable I2C Software end mode with no HW PEC comparison.

LL_I2C_MODE_SMBUS_RELOAD

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC

Enable SMBUS Software end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC

Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT

Own address 1 is a 7-bit address.

LL_I2C_OWNADDRESS1_10BIT

Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK

Own Address2 No mask.

LL_I2C_OWNADDRESS2_MASK01

Only Address2 bits[7:2] are compared.

LL_I2C_OWNADDRESS2_MASK02

Only Address2 bits[7:3] are compared.

LL_I2C_OWNADDRESS2_MASK03

Only Address2 bits[7:4] are compared.

LL_I2C_OWNADDRESS2_MASK04

Only Address2 bits[7:5] are compared.

LL_I2C_OWNADDRESS2_MASK05

Only Address2 bits[7:6] are compared.

LL_I2C_OWNADDRESS2_MASK06

Only Address2 bits[7] are compared.

LL_I2C_OWNADDRESS2_MASK07

No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C

I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HOST

SMBus Host address acknowledge

LL_I2C_MODE_SMBUS_DEVICE

SMBus Device default mode (Default address not acknowledge)

LL_I2C_MODE_SMBUS_DEVICE_ARP

SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE

Master request a write transfer.

LL_I2C_REQUEST_READ

Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW

TimeoutA is used to detect SCL low level timeout.

LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA

TimeoutA enable bit

LL_I2C_SMBUS_TIMEOUTB

TimeoutB (extended clock) enable bit

LL_I2C_SMBUS_ALL_TIMEOUT

TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __SCLH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __SCLL_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

100 LL I2S Generic Driver

100.1 I2S Firmware driver registers structures

100.1.1 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32h5xx_ll_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

100.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

100.2.1 Detailed description of functions

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataLength)

Function description

Set I2S Data frame format.

Parameters

- **SPIx:** SPI Handle
- **DataLength:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat
- I2SCFGR DATFMT LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (const SPI_TypeDef * SPIx)

Function description

Get I2S Data frame format.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat
- I2SCFGR DATFMT LL_I2S_GetDataFormat

LL_I2S_SetChannelLengthType

Function name

__STATIC_INLINE void LL_I2S_SetChannelLengthType (SPI_TypeDef * SPIx, uint32_t ChannelLengthType)

Function description

Set I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle
- **ChannelLengthType:** This parameter can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Return values

- **None:**

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_SetChannelLengthType

LL_I2S_GetChannelLengthType

Function name

__STATIC_INLINE uint32_t LL_I2S_GetChannelLengthType (const SPI_TypeDef * SPIx)

Function description

Get I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_GetChannelLengthType

LL_I2S_EnableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_EnableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Invert the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_EnableWordSelectInversion

LL_I2S_DisableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_DisableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Use the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_DisableWordSelectInversion

LL_I2S_IsEnabledWordSelectInversion

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledWordSelectInversion (const SPI_TypeDef * SPIx)

Function description

Check if polarity of WS signal is inverted.

Parameters

- **SPIx:** SPI Handle

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_IsEnabledWordSelectInversion

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get I2S standard.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S config.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (const SPI_TypeDef * SPIx)

Function description

Get I2S config.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- CR1 SPE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral and disable I2S mode.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_I2S_Disable
- I2SCFGR I2SMOD LL_I2S_Disable

LL_I2S_EnableIOSwap

Function name

__STATIC_INLINE void LL_I2S_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_EnableIOSwap

LL_I2S_DisableIOSwap

Function name

__STATIC_INLINE void LL_I2S_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_DisableIOSwap

LL_I2S_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOSwap (const SPI_TypeDef * SPIx)

Function description

Check if SDO and SDI pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_IsEnabledIOSwap

LL_I2S_EnableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_EnableGPIOControl

LL_I2S_DisableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_DisableGPIOControl

LL_I2S_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledGPIOControl (const SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_IsEnabledGPIOControl

LL_I2S_EnableIOLock

Function name

__STATIC_INLINE void LL_I2S_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the SPI_CFG2 register content can not be modified until a hardware reset occurs. The reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_I2S_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOLock (const SPI_TypeDef * SPIx)

Function description

Check if the the SPI_CFG2 register is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_I2S_IsEnabledIOLock

LL_I2S_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_I2S_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_SetTransferBitOrder

LL_I2S_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferBitOrder (const SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_GetTransferBitOrder

LL_I2S_StartTransfer

Function name

__STATIC_INLINE void LL_I2S_StartTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_StartTransfer

LL_I2S_IsActiveTransfer

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveTransfer (const SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_IsActiveTransfer

LL_I2S_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_I2S_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_SetFIFOThreshold

LL_I2S_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I2S_GetFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_GetFIFOThreshold

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint32_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (const SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (const SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (const SPI_TypeDef * SPIx)

Function description

Check if the master clock output (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_I2S_IsActiveFlag_RXP

LL_I2S_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_I2S_IsActiveFlag_TXP

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_I2S_IsActiveFlag_FRE

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_EnableIT_RXP

LL_I2S_EnableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_EnableIT_TXP

LL_I2S_EnableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_EnableIT_UDR

LL_I2S_EnableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_EnableIT_OVR

LL_I2S_EnableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_EnableIT_FRE

LL_I2S_DisableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_DisableIT_RXP

LL_I2S_DisableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_DisableIT_TXP

LL_I2S_DisableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_DisableIT_UDR

LL_I2S_DisableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_DisableIT_OVR

LL_I2S_DisableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_DisableIT_FRE

LL_I2S_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXP (const SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_IsEnabledIT_RXP

LL_I2S_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXP (const SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_IsEnabledIT_TXP

LL_I2S_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_UDR (const SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_IsEnabledIT_UDR

LL_I2S_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_OVR (const SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_IsEnabledIT_OVR

LL_I2S_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_FRE (const SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_IsEnabledIT_FRE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData16

LL_I2S_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_I2S_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData32

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData16

LL_I2S_TransmitData32

Function name

__STATIC_INLINE void LL_I2S_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData32

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in I2S configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- I2S (SPI) source clock must be ready before calling this function. Otherwise will results in wrong programming.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).
- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

100.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

100.3.1 I2S

I2S

Audio Frequency

LL_I2S_AUDIOFREQ_192K

Audio Frequency configuration 192000 Hz

LL_I2S_AUDIOFREQ_96K

Audio Frequency configuration 96000 Hz

LL_I2S_AUDIOFREQ_48K

Audio Frequency configuration 48000 Hz

LL_I2S_AUDIOFREQ_44K

Audio Frequency configuration 44100 Hz

LL_I2S_AUDIOFREQ_32K

Audio Frequency configuration 32000 Hz

LL_I2S_AUDIOFREQ_22K

Audio Frequency configuration 22050 Hz

LL_I2S_AUDIOFREQ_16K

Audio Frequency configuration 16000 Hz

LL_I2S_AUDIOFREQ_11K

Audio Frequency configuration 11025 Hz

LL_I2S_AUDIOFREQ_8K

Audio Frequency configuration 8000 Hz

LL_I2S_AUDIOFREQ_DEFAULT

Audio Freq not specified. Register I2SDIV = 0

Transmission Bit Order

LL_I2S_LSB_FIRST

LL_I2S_MSB_FIRST

Type of Channel Length

LL_I2S_SLAVE_VARIABLE_CH_LENGTH

LL_I2S_SLAVE_FIXED_CH_LENGTH

Data Format

LL_I2S_DATAFORMAT_16B

LL_I2S_DATAFORMAT_16B_EXTENDED

LL_I2S_DATAFORMAT_24B

LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED

LL_I2S_DATAFORMAT_32B

FIFO Threshold Level

LL_I2S_FIFO_TH_01DATA

LL_I2S_FIFO_TH_02DATA

LL_I2S_FIFO_TH_03DATA

LL_I2S_FIFO_TH_04DATA

LL_I2S_FIFO_TH_05DATA

LL_I2S_FIFO_TH_06DATA

LL_I2S_FIFO_TH_07DATA

LL_I2S_FIFO_TH_08DATA

MCLK Output

LL_I2S_MCLK_OUTPUT_DISABLE

LL_I2S_MCLK_OUTPUT_ENABLE

Operation Mode

LL_I2S_MODE_SLAVE_TX

LL_I2S_MODE_SLAVE_RX

LL_I2S_MODE_SLAVE_FULL_DUPLEX

LL_I2S_MODE_MASTER_TX

LL_I2S_MODE_MASTER_RX

LL_I2S_MODE_MASTER_FULL_DUPLEX

Clock Polarity

LL_I2S_POLARITY_LOW

LL_I2S_POLARITY_HIGH

Prescaler Factor

LL_I2S_PRESCALER_PARITY_EVEN

Odd factor: Real divider value is = I2SDIV * 2

LL_I2S_PRESCALER_PARITY_ODD

Odd factor: Real divider value is = (I2SDIV * 2)+1

I2S Standard

LL_I2S_STANDARD_PHILIPS

LL_I2S_STANDARD_MSB

LL_I2S_STANDARD_LSB

LL_I2S_STANDARD_PCM_SHORT

LL_I2S_STANDARD_PCM_LONG

Common Write and read registers Macros

LL_I2S_WriteReg

Description:

- Write a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2S_ReadReg

Description:

- Read a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be read

Return value:

- Register: value

101 LL I3C Generic Driver

101.1 I3C Firmware driver registers structures

101.1.1 LL_I3C_CtrlBusConfTypeDef

LL_I3C_CtrlBusConfTypeDef is defined in the stm32h5xx_ll_i3c.h

Data Fields

- ***uint32_t SDAHoldTime***
- ***uint32_t WaitTime***
- ***uint8_t SCLPPLowDuration***
- ***uint8_t SCLI3CHighDuration***
- ***uint8_t SCLODLowDuration***
- ***uint8_t SCLI2CHighDuration***
- ***uint8_t BusFreeDuration***
- ***uint8_t BusIdleDuration***

Field Documentation

- ***uint32_t LL_I3C_CtrlBusConfTypeDef::SDAHoldTime***
  Specifies the I3C SDA hold time. This parameter must be a value of ***I3C_LL_EC_SDA_HOLD_TIME***
- ***uint32_t LL_I3C_CtrlBusConfTypeDef::WaitTime***
  Specifies the time that the main and the new controllers should wait before issuing a start. This parameter must be a value of ***I3C_LL_EC_OWN_ACTIVITY_STATE***
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLPPLowDuration***
  Specifies the I3C SCL low duration in number of kernel clock cycles in I3C push-pull phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLI3CHighDuration***
  Specifies the I3C SCL high duration in number of kernel clock cycles, used for I3C messages for I3C open-drain and push pull phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLODLowDuration***
  Specifies the I3C SCL low duration in number of kernel clock cycles in open-drain phases, used for legacy I2C commands and for I3C open-drain phases. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::SCLI2CHighDuration***
  Specifies the I3C SCL high duration in number of kernel clock cycles, used for legacy I2C commands. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::BusFreeDuration***
  Specifies the I3C controller duration in number of kernel clock cycles, after a stop and before a start. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.
- ***uint8_t LL_I3C_CtrlBusConfTypeDef::BusIdleDuration***
  Specifies the I3C controller duration in number of kernel clock cycles to be elapsed, after that both SDA and SCL are continuously high and stable before issuing a hot-join event. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.

101.1.2 LL_I3C_TgtBusConfTypeDef

LL_I3C_TgtBusConfTypeDef is defined in the stm32h5xx_ll_i3c.h

Data Fields

- ***uint8_t BusAvailableDuration***

Field Documentation

- ***uint8_t LL_I3C_TgtBusConfTypeDef::BusAvailableDuration***
  Specifies the I3C target duration in number of kernel clock cycles, when the SDA and the SCL are high for at least taval. This parameter must be a number between Min_Data=0 and Max_Data=0xFF.

101.1.3 LL_I3C_InitTypeDef

LL_I3C_InitTypeDef is defined in the stm32h5xx_ll_i3c.h

Data Fields

- ***LL_I3C_CtrlBusConfTypeDef CtrlBusCharacteristic***
- ***LL_I3C_TgtBusConfTypeDef TgtBusCharacteristic***

Field Documentation

- ***LL_I3C_CtrlBusConfTypeDef LL_I3C_InitTypeDef::CtrlBusCharacteristic***
  Specifies the I3C controller bus characteristic configuration when Controller mode
- ***LL_I3C_TgtBusConfTypeDef LL_I3C_InitTypeDef::TgtBusCharacteristic***
  Specifies the I3C target bus characteristic configuration when Target mode

101.2 I3C Firmware driver API description

The following section lists the various functions of the I3C library.

101.2.1 Detailed description of functions

LL_I3C_Enable

Function name

__STATIC_INLINE void LL_I3C_Enable (I3C_TypeDef * I3Cx)

Function description

Enable I3C peripheral (EN = 1).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_Enable

LL_I3C_Disable

Function name

__STATIC_INLINE void LL_I3C_Disable (I3C_TypeDef * I3Cx)

Function description

Disable I3C peripheral (EN = 0).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Controller mode: before clearing EN, all possible target requests must be disabled using DISEC CCC.
  Target mode: software is not expected clearing EN unless a partial reset of the IP is needed

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_Disable

LL_I3C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabled (const I3C_TypeDef * I3Cx)

Function description

Check if the I3C peripheral is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR EN LL_I3C_IsEnabled

LL_I3C_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledReset (const I3C_TypeDef * I3Cx)

Function description

Check if Reset action is required or not required.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This bit indicates if Reset Action field has been updated by HW upon reception of RSTACT during current frame.

Reference Manual to LL API cross reference:

- DEVR0 RSTVAL LL_I3C_IsEnabledReset

LL_I3C_SetMode

Function name

__STATIC_INLINE void LL_I3C_SetMode (I3C_TypeDef * I3Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I3Cx:** I3C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I3C_MODE_CONTROLLER
  - LL_I3C_MODE_TARGET

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR CRINIT LL_I3C_SetMode

LL_I3C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMode (const I3C_TypeDef * I3Cx)

Function description

Get peripheral mode.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MODE_CONTROLLER
  - LL_I3C_MODE_TARGET

Reference Manual to LL API cross reference:

- CFGR CRINIT LL_I3C_GetMode

LL_I3C_EnableArbitrationHeader

Function name

__STATIC_INLINE void LL_I3C_EnableArbitrationHeader (I3C_TypeDef * I3Cx)

Function description

An arbitration header (7'h7E) is sent after Start in case of legacy I2C or I3C private transfers.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_EnableArbitrationHeader

LL_I3C_DisableArbitrationHeader

Function name

__STATIC_INLINE void LL_I3C_DisableArbitrationHeader (I3C_TypeDef * I3Cx)

Function description

Target address is sent directly after a Start in case of legacy I2C or I3C private transfers.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_DisableArbitrationHeader

LL_I3C_IsEnabledArbitrationHeader

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledArbitrationHeader (const I3C_TypeDef * I3Cx)

Function description

Check if the arbitration header is enabled of disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR NOARBH LL_I3C_IsEnabledArbitrationHeader

LL_I3C_EnableResetPattern

Function name

__STATIC_INLINE void LL_I3C_EnableResetPattern (I3C_TypeDef * I3Cx)

Function description

A Reset Pattern is inserted before the STOP at the end of a frame when the last CCC of the frame was RSTACT CCC.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_EnableResetPattern

LL_I3C_DisableResetPattern

Function name

__STATIC_INLINE void LL_I3C_DisableResetPattern (I3C_TypeDef * I3Cx)

Function description

A single STOP is emitted at the end of a frame.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_DisableResetPattern

LL_I3C_IsEnabledResetPattern

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledResetPattern (const I3C_TypeDef * I3Cx)

Function description

Check if Reset Pattern is enabled of disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR RSTPTRN LL_I3C_IsEnabledResetPattern

LL_I3C_EnableExitPattern

Function name

__STATIC_INLINE void LL_I3C_EnableExitPattern (I3C_TypeDef * I3Cx)

Function description

An Exit Pattern is sent after header (MTYPE = header) to program an escalation fault.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_EnableExitPattern

LL_I3C_DisableExitPattern

Function name

__STATIC_INLINE void LL_I3C_DisableExitPattern (I3C_TypeDef * I3Cx)

Function description

An Exit Pattern is not sent after header (MTYPE = header).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be modified only when there is no frame ongoing

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_DisableExitPattern

LL_I3C_IsEnabledExitPattern

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledExitPattern (const I3C_TypeDef * I3Cx)

Function description

Check if Exit Pattern is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR EXITPTRN LL_I3C_IsEnabledExitPattern

LL_I3C_EnableHighKeeperSDA

Function name

__STATIC_INLINE void LL_I3C_EnableHighKeeperSDA (I3C_TypeDef * I3Cx)

Function description

High Keeper is enabled and will be used in place of standard Open drain Pull Up device during handoff procedures.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_EnableHighKeeperSDA

LL_I3C_DisableHighKeeperSDA

Function name

__STATIC_INLINE void LL_I3C_DisableHighKeeperSDA (I3C_TypeDef * I3Cx)

Function description

High Keeper is disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_DisableHighKeeperSDA

LL_I3C_IsEnabledHighKeeperSDA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHighKeeperSDA (const I3C_TypeDef * I3Cx)

Function description

Check if High Keeper is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR HKSDAEN LL_I3C_IsEnabledHighKeeperSDA

LL_I3C_EnableHJAck

Function name

__STATIC_INLINE void LL_I3C_EnableHJAck (I3C_TypeDef * I3Cx)

Function description

Hot Join Request is Acked.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be used when I3C is acting as a Controller.

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_EnableHJAck

LL_I3C_DisableHJAck

Function name

__STATIC_INLINE void LL_I3C_DisableHJAck (I3C_TypeDef * I3Cx)

Function description

Hot Join Request is Nacked.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be used when I3C is acting as a Controller.

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_DisableHJAck

LL_I3C_IsEnabledHJAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHJAck (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join Request Acknowledgement is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR HJACK LL_I3C_IsEnabledHJAck

LL_I3C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I3C_DMA_GetRegAddr (const I3C_TypeDef * I3Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I3Cx:** I3C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I3C_DMA_REG_DATA_TRANSMIT_BYTE
  - LL_I3C_DMA_REG_DATA_RECEIVE_BYTE
  - LL_I3C_DMA_REG_DATA_TRANSMIT_WORD
  - LL_I3C_DMA_REG_DATA_RECEIVE_WORD
  - LL_I3C_DMA_REG_STATUS
  - LL_I3C_DMA_REG_CONTROL

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TDR TDB0 LL_I3C_DMA_GetRegAddr
- TDWR TDWR LL_I3C_DMA_GetRegAddr
- RDR RXRB0 LL_I3C_DMA_GetRegAddr
- RDWR RDWR LL_I3C_DMA_GetRegAddr
- SR SR LL_I3C_DMA_GetRegAddr
- CR CR LL_I3C_DMA_GetRegAddr

LL_I3C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_RX (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO reception requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_EnableDMAReq_RX

LL_I3C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_RX (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO reception requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_DisableDMAReq_RX

LL_I3C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_RX (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO reception requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR RXDMAEN LL_I3C_IsEnabledDMAReq_RX

LL_I3C_SetRxFIFOThreshold

Function name

__STATIC_INLINE void LL_I3C_SetRxFIFOThreshold (I3C_TypeDef * I3Cx, uint32_t RxFIFOThreshold)

Function description

Set the Receive FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.
- **RxFIFOThreshold:** This parameter can be one of the following values:
  - LL_I3C_RXFIFO_THRESHOLD_1_4
  - LL_I3C_RXFIFO_THRESHOLD_4_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXTHRES LL_I3C_SetRxFIFOThreshold

LL_I3C_GetRxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I3C_GetRxFIFOThreshold (const I3C_TypeDef * I3Cx)

Function description

Get the Receive FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_RXFIFO_THRESHOLD_1_4
  - LL_I3C_RXFIFO_THRESHOLD_4_4

Reference Manual to LL API cross reference:

- CFGR RXTHRES LL_I3C_GetRxFIFOThreshold

LL_I3C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_TX (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO transmission requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_EnableDMAReq_TX

LL_I3C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_TX (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO transmission requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_DisableDMAReq_TX

LL_I3C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_TX (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO transmission requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TXDMAEN LL_I3C_IsEnabledDMAReq_TX

LL_I3C_SetTxFIFOThreshold

Function name

__STATIC_INLINE void LL_I3C_SetTxFIFOThreshold (I3C_TypeDef * I3Cx, uint32_t TxFIFOThreshold)

Function description

Set the Transmit FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.
- **TxFIFOThreshold:** This parameter can be one of the following values:
  - LL_I3C_TXFIFO_THRESHOLD_1_4
  - LL_I3C_TXFIFO_THRESHOLD_4_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXTHRES LL_I3C_SetTxFIFOThreshold

LL_I3C_GetTxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTxFIFOThreshold (const I3C_TypeDef * I3Cx)

Function description

Get the Transmit FIFO Threshold level.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_TXFIFO_THRESHOLD_1_4
  - LL_I3C_TXFIFO_THRESHOLD_4_4
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TXTHRES LL_I3C_GetTxFIFOThreshold

LL_I3C_EnableDMAReq_Status

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_Status (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO Status requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_EnableDMAReq_Status

LL_I3C_DisableDMAReq_Status

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_Status (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO Status requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_DisableDMAReq_Status

LL_I3C_IsEnabledDMAReq_Status

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_Status (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO Status requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR SDMAEN LL_I3C_IsEnabledDMAReq_Status

LL_I3C_EnableStatusFIFO

Function name

__STATIC_INLINE void LL_I3C_EnableStatusFIFO (I3C_TypeDef * I3Cx)

Function description

Enable the Status FIFO.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Not applicable in target mode. Status FIFO always disabled in target mode.

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_EnableStatusFIFO

LL_I3C_DisableStatusFIFO

Function name

__STATIC_INLINE void LL_I3C_DisableStatusFIFO (I3C_TypeDef * I3Cx)

Function description

Disable the Status FIFO Threshold.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_DisableStatusFIFO

LL_I3C_IsEnabledStatusFIFO

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStatusFIFO (const I3C_TypeDef * I3Cx)

Function description

Check if the Status FIFO Threshold is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR SMODE LL_I3C_IsEnabledStatusFIFO

LL_I3C_EnableControlFIFO

Function name

__STATIC_INLINE void LL_I3C_EnableControlFIFO (I3C_TypeDef * I3Cx)

Function description

Enable the Control and Transmit FIFO preloaded before starting a transfer on I3C bus.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- Not applicable in target mode. Control FIFO always disabled in target mode.

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_EnableControlFIFO

LL_I3C_DisableControlFIFO

Function name

__STATIC_INLINE void LL_I3C_DisableControlFIFO (I3C_TypeDef * I3Cx)

Function description

Disable the Control and Transmit FIFO preloaded before starting a transfer on I3C bus.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_DisableControlFIFO

LL_I3C_IsEnabledControlFIFO

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledControlFIFO (const I3C_TypeDef * I3Cx)

Function description

Check if the Control and Transmit FIFO preloaded is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TMODE LL_I3C_IsEnabledControlFIFO

LL_I3C_EnableDMAReq_Control

Function name

__STATIC_INLINE void LL_I3C_EnableDMAReq_Control (I3C_TypeDef * I3Cx)

Function description

Enable DMA FIFO Control word transfer requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_EnableDMAReq_Control

LL_I3C_DisableDMAReq_Control

Function name

__STATIC_INLINE void LL_I3C_DisableDMAReq_Control (I3C_TypeDef * I3Cx)

Function description

Disable DMA FIFO Control word transfer requests.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_DisableDMAReq_Control

LL_I3C_IsEnabledDMAReq_Control

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledDMAReq_Control (const I3C_TypeDef * I3Cx)

Function description

Check if DMA FIFO Control word transfer requests are enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR CDMAEN LL_I3C_IsEnabledDMAReq_Control

LL_I3C_EnableOwnDynAddress

Function name

__STATIC_INLINE void LL_I3C_EnableOwnDynAddress (I3C_TypeDef * I3Cx)

Function description

Set Own Dynamic Address as Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_EnableOwnDynAddress

LL_I3C_DisableOwnDynAddress

Function name

__STATIC_INLINE void LL_I3C_DisableOwnDynAddress (I3C_TypeDef * I3Cx)

Function description

Set Own Dynamic Address as Not-Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_DisableOwnDynAddress

LL_I3C_IsEnabledOwnDynAddress

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledOwnDynAddress (const I3C_TypeDef * I3Cx)

Function description

Check if Own Dynamic address is Valid or Not-Valid.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 DAVAL LL_I3C_IsEnabledOwnDynAddress

LL_I3C_SetOwnDynamicAddress

Function name

__STATIC_INLINE void LL_I3C_SetOwnDynamicAddress (I3C_TypeDef * I3Cx, uint32_t OwnDynamicAddress)

Function description

Configure Own Dynamic Address.

Parameters

- **I3Cx:** I3C Instance.
- **OwnDynamicAddress:** This parameter must be a value between Min_Data=0 and Max_Data=0x7F

Return values

- **None:**

Notes

- This bit can be programmed in controller mode or during Dynamic Address procedure from current controller.

Reference Manual to LL API cross reference:

- DEVR0 DA LL_I3C_SetOwnDynamicAddress

LL_I3C_GetOwnDynamicAddress

Function name

__STATIC_INLINE uint8_t LL_I3C_GetOwnDynamicAddress (const I3C_TypeDef * I3Cx)

Function description

Get Own Dynamic Address.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x7F

Reference Manual to LL API cross reference:

- DEVR0 DA LL_I3C_GetOwnDynamicAddress

LL_I3C_EnableIBI

Function name

__STATIC_INLINE void LL_I3C_EnableIBI (I3C_TypeDef * I3Cx)

Function description

Set IBI procedure allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_EnableIBI

LL_I3C_DisableIBI

Function name

__STATIC_INLINE void LL_I3C_DisableIBI (I3C_TypeDef * I3Cx)

Function description

Set IBI procedure not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_DisableIBI

LL_I3C_IsEnabledIBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIBI (const I3C_TypeDef * I3Cx)

Function description

Check if IBI procedure is allowed or not allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 IBIEN LL_I3C_IsEnabledIBI

LL_I3C_EnableControllerRoleReq

Function name

__STATIC_INLINE void LL_I3C_EnableControllerRoleReq (I3C_TypeDef * I3Cx)

Function description

Set Controller-role Request allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_EnableControllerRoleReq

LL_I3C_DisableControllerRoleReq

Function name

__STATIC_INLINE void LL_I3C_DisableControllerRoleReq (I3C_TypeDef * I3Cx)

Function description

Set Controller-role Request as not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_DisableControllerRoleReq

LL_I3C_IsEnabledControllerRoleReq

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledControllerRoleReq (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role Request is allowed or not-allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 CREN LL_I3C_IsEnabledControllerRoleReq

LL_I3C_EnableHotJoin

Function name

__STATIC_INLINE void LL_I3C_EnableHotJoin (I3C_TypeDef * I3Cx)

Function description

Set Hot Join allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_EnableHotJoin

LL_I3C_DisableHotJoin

Function name

__STATIC_INLINE void LL_I3C_DisableHotJoin (I3C_TypeDef * I3Cx)

Function description

Set Hot Join as not-allowed (when the I3C is acting as target).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0) or updated by HW upon reception of DISEC CCC.

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_DisableHotJoin

LL_I3C_IsEnabledHotJoin

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledHotJoin (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join is allowed or not-allowed.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVR0 HJEN LL_I3C_IsEnabledHotJoin

LL_I3C_SetMaxReadLength

Function name

__STATIC_INLINE void LL_I3C_SetMaxReadLength (I3C_TypeDef * I3Cx, uint16_t MaxReadLength)

Function description

Configure Maximum Read Length (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **MaxReadLength:** This parameter must be a value between Min_Data=0x0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Those bits can be updated by HW upon reception of GETMRL CCC.

Reference Manual to LL API cross reference:

- MAXRLR MRL LL_I3C_SetMaxReadLength

LL_I3C_GetMaxReadLength

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxReadLength (const I3C_TypeDef * I3Cx)

Function description

Return Maximum Read Length (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFFFFF

Reference Manual to LL API cross reference:

- MAXRLR MRL LL_I3C_GetMaxReadLength

LL_I3C_ConfigNbIBIAddData

Function name

__STATIC_INLINE void LL_I3C_ConfigNbIBIAddData (I3C_TypeDef * I3Cx, uint32_t NbIBIAddData)

Function description

Configure the number of additional Mandatory Data Byte (MDB) sent to the controller after an acknowledge of the IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **NbIBIAddData:** This parameter can be one of the following values:
  - LL_I3C_PAYLOAD_EMPTY
  - LL_I3C_PAYLOAD_1_BYTE
  - LL_I3C_PAYLOAD_2_BYTES
  - LL_I3C_PAYLOAD_3_BYTES
  - LL_I3C_PAYLOAD_4_BYTES

Return values

- **None:**

Reference Manual to LL API cross reference:

- MAXRLR IBIP LL_I3C_ConfigNbIBIAddData

LL_I3C_GetConfigNbIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_GetConfigNbIBIAddData (const I3C_TypeDef * I3Cx)

Function description

Return the number of additional Mandatory Data Byte (MDB) sent to the controller after an acknowledge of the IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_PAYLOAD_EMPTY
  - LL_I3C_PAYLOAD_1_BYTE
  - LL_I3C_PAYLOAD_2_BYTES
  - LL_I3C_PAYLOAD_3_BYTES
  - LL_I3C_PAYLOAD_4_BYTES

Reference Manual to LL API cross reference:

- MAXRLR IBIP LL_I3C_GetConfigNbIBIAddData

LL_I3C_SetMaxWriteLength

Function name

__STATIC_INLINE void LL_I3C_SetMaxWriteLength (I3C_TypeDef * I3Cx, uint16_t MaxWriteLength)

Function description

Configure Maximum Write Length (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **MaxWriteLength:** This parameter must be a value between Min_Data=0x0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Those bits can be updated by HW upon reception of GETMWL CCC.

Reference Manual to LL API cross reference:

- MAXWLR MWL LL_I3C_SetMaxWriteLength

LL_I3C_GetMaxWriteLength

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxWriteLength (const I3C_TypeDef * I3Cx)

Function description

Return Maximum Write Length (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFFFFF

Reference Manual to LL API cross reference:

- MAXWLR MWL LL_I3C_GetMaxWriteLength

LL_I3C_ConfigClockWaveForm

Function name

__STATIC_INLINE void LL_I3C_ConfigClockWaveForm (I3C_TypeDef * I3Cx, uint32_t ClockWaveForm)

Function description

Configure the SCL clock signal waveform.

Parameters

- **I3Cx:** I3C Instance.
- **ClockWaveForm:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 TIMINGR0 LL_I3C_ConfigClockWaveForm

LL_I3C_GetClockWaveForm

Function name

__STATIC_INLINE uint32_t LL_I3C_GetClockWaveForm (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock signal waveform.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF.

Reference Manual to LL API cross reference:

- TIMINGR0 TIMINGR0 LL_I3C_GetClockWaveForm

LL_I3C_SetPeriodClockLowPP

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockLowPP (I3C_TypeDef * I3Cx, uint32_t PeriodClockLowPP)

Function description

Configure the SCL clock low period during I3C push-pull phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockLowPP:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_PP LL_I3C_SetPeriodClockLowPP

LL_I3C_GetPeriodClockLowPP

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockLowPP (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock low period during I3C push-pull phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_PP LL_I3C_GetPeriodClockLowPP

LL_I3C_SetPeriodClockHighI3C

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockHighI3C (I3C_TypeDef * I3Cx, uint32_t PeriodClockHighI3C)

Function description

Configure the SCL clock High period during I3C open drain and push-pull phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockHighI3C:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I3C LL_I3C_SetPeriodClockHighI3C

LL_I3C_GetPeriodClockHighI3C

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockHighI3C (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock high period during I3C open drain and push-pull phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I3C LL_I3C_GetPeriodClockHighI3C

LL_I3C_SetPeriodClockLowOD

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockLowOD (I3C_TypeDef * I3Cx, uint32_t PeriodClockLowOD)

Function description

Configure the SCL clock low period during I3C open drain phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockLowOD:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_OD LL_I3C_SetPeriodClockLowOD

LL_I3C_GetPeriodClockLowOD

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockLowOD (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock low period during I3C open phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLL_OD LL_I3C_GetPeriodClockLowOD

LL_I3C_SetPeriodClockHighI2C

Function name

__STATIC_INLINE void LL_I3C_SetPeriodClockHighI2C (I3C_TypeDef * I3Cx, uint32_t PeriodClockHighI2C)

Function description

Configure the SCL clock High period during I2C open drain phases.

Parameters

- **I3Cx:** I3C Instance.
- **PeriodClockHighI2C:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I2C LL_I3C_SetPeriodClockHighI2C

LL_I3C_GetPeriodClockHighI2C

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPeriodClockHighI2C (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock high period during I2C open drain phases.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR0 SCLH_I2C LL_I3C_GetPeriodClockHighI2C

LL_I3C_SetDataHoldTime

Function name

__STATIC_INLINE void LL_I3C_SetDataHoldTime (I3C_TypeDef * I3Cx, uint32_t DataHoldTime)

Function description

Configure the Controller additional hold time on SDA line.

Parameters

- **I3Cx:** I3C Instance.
- **DataHoldTime:** This parameter can be one of the following values:
  - LL_I3C_SDA_HOLD_TIME_0_5
  - LL_I3C_SDA_HOLD_TIME_1_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_SetDataHoldTime

LL_I3C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataHoldTime (const I3C_TypeDef * I3Cx)

Function description

Get the Controller additional hold time on SDA line.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_SDA_HOLD_TIME_0_5
  - LL_I3C_SDA_HOLD_TIME_1_5

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_GetDataHoldTime

LL_I3C_SetAvalTiming

Function name

__STATIC_INLINE void LL_I3C_SetAvalTiming (I3C_TypeDef * I3Cx, uint32_t AvalTiming)

Function description

Configure the Idle, Available state.

Parameters

- **I3Cx:** I3C Instance.
- **AvalTiming:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 AVAL LL_I3C_SetAvalTiming

LL_I3C_GetAvalTiming

Function name

__STATIC_INLINE uint32_t LL_I3C_GetAvalTiming (const I3C_TypeDef * I3Cx)

Function description

Get the Idle, Available integer value state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR1 AVAL LL_I3C_GetAvalTiming

LL_I3C_SetFreeTiming

Function name

__STATIC_INLINE void LL_I3C_SetFreeTiming (I3C_TypeDef * I3Cx, uint32_t FreeTiming)

Function description

Configure the Free state.

Parameters

- **I3Cx:** I3C Instance.
- **FreeTiming:** This parameter must be a value between Min_Data=0 and Max_Data=0x3F.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 FREE LL_I3C_SetFreeTiming

LL_I3C_GetFreeTiming

Function name

__STATIC_INLINE uint32_t LL_I3C_GetFreeTiming (const I3C_TypeDef * I3Cx)

Function description

Get the Free integeter value state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x3F.

Reference Manual to LL API cross reference:

- TIMINGR1 FREE LL_I3C_GetFreeTiming

LL_I3C_SetControllerActivityState

Function name

__STATIC_INLINE void LL_I3C_SetControllerActivityState (I3C_TypeDef * I3Cx, uint32_t ControllerActivityState)

Function description

Configure the activity state of the new controller.

Parameters

- **I3Cx:** I3C Instance.
- **ControllerActivityState:** This parameter can be one of the following values:
  - LL_I3C_OWN_ACTIVITY_STATE_0
  - LL_I3C_OWN_ACTIVITY_STATE_1
  - LL_I3C_OWN_ACTIVITY_STATE_2
  - LL_I3C_OWN_ACTIVITY_STATE_3

Return values

- **None:**

Notes

- Refer to MIPI I3C specification (https:__www.mipi.org_specifications) for more details related to Activity State.

Reference Manual to LL API cross reference:

- TIMINGR1 ASNCR LL_I3C_SetControllerActivityState

LL_I3C_GetControllerActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetControllerActivityState (const I3C_TypeDef * I3Cx)

Function description

Get the activity state of the new controller.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_OWN_ACTIVITY_STATE_0
  - LL_I3C_OWN_ACTIVITY_STATE_1
  - LL_I3C_OWN_ACTIVITY_STATE_2
  - LL_I3C_OWN_ACTIVITY_STATE_3

Notes

- Refer to MIPI I3C specification (https:__www.mipi.org_specifications) for more details related to Activity State.

Reference Manual to LL API cross reference:

- TIMINGR1 ASNCR LL_I3C_GetControllerActivityState

LL_I3C_SetCtrlBusCharacteristic

Function name

__STATIC_INLINE void LL_I3C_SetCtrlBusCharacteristic (I3C_TypeDef * I3Cx, uint32_t CtrlBusCharacteristic)

Function description

Configure the Controller SDA Hold time, Bus Free, Activity state, Idle state.

Parameters

- **I3Cx:** I3C Instance.
- **CtrlBusCharacteristic:** This parameter must be a value between Min_Data=0 and Max_Data=0x107F03FF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 FREE LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 ASNCR LL_I3C_SetCtrlBusCharacteristic
- TIMINGR1 IDLE LL_I3C_SetCtrlBusCharacteristic

LL_I3C_GetCtrlBusCharacteristic

Function name

__STATIC_INLINE uint32_t LL_I3C_GetCtrlBusCharacteristic (const I3C_TypeDef * I3Cx)

Function description

Get the Controller SDA Hold time, Bus Free, Activity state, Idle state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x107F03FF.

Reference Manual to LL API cross reference:

- TIMINGR1 SDA_HD LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 FREE LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 ASNCR LL_I3C_GetCtrlBusCharacteristic
- TIMINGR1 IDLE LL_I3C_GetCtrlBusCharacteristic

LL_I3C_SetTgtBusCharacteristic

Function name

__STATIC_INLINE void LL_I3C_SetTgtBusCharacteristic (I3C_TypeDef * I3Cx, uint32_t TgtBusCharacteristic)

Function description

Configure the Target Available state.

Parameters

- **I3Cx:** I3C Instance.
- **TgtBusCharacteristic:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR1 IDLE LL_I3C_SetTgtBusCharacteristic

LL_I3C_GetTgtBusCharacteristic

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTgtBusCharacteristic (const I3C_TypeDef * I3Cx)

Function description

Get the Target Available state.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR1 IDLE LL_I3C_GetTgtBusCharacteristic

LL_I3C_SetStallTime

Function name

__STATIC_INLINE void LL_I3C_SetStallTime (I3C_TypeDef * I3Cx, uint32_t ControllerStallTime)

Function description

Configure the SCL clock stalling time on I3C Bus (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **ControllerStallTime:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR2 STALL LL_I3C_SetStallTime

LL_I3C_GetStallTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetStallTime (const I3C_TypeDef * I3Cx)

Function description

Get the SCL clock stalling time on I3C Bus (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Reference Manual to LL API cross reference:

- TIMINGR2 STALL LL_I3C_GetStallTime

LL_I3C_EnableStallACK

Function name

__STATIC_INLINE void LL_I3C_EnableStallACK (I3C_TypeDef * I3Cx)

Function description

Set stall on ACK bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_EnableStallACK

LL_I3C_DisableStallACK

Function name

__STATIC_INLINE void LL_I3C_DisableStallACK (I3C_TypeDef * I3Cx)

Function description

Disable stall on ACK bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_DisableStallACK

LL_I3C_IsEnabledStallACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallACK (const I3C_TypeDef * I3Cx)

Function description

Check if stall on ACK bit is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLA LL_I3C_IsEnabledStallACK

LL_I3C_EnableStallParityCCC

Function name

__STATIC_INLINE void LL_I3C_EnableStallParityCCC (I3C_TypeDef * I3Cx)

Function description

Set stall on Parity bit of Command Code byte (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_EnableStallParityCCC

LL_I3C_DisableStallParityCCC

Function name

__STATIC_INLINE void LL_I3C_DisableStallParityCCC (I3C_TypeDef * I3Cx)

Function description

Disable stall on Parity bit of Command Code byte (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_DisableStallParityCCC

LL_I3C_IsEnabledStallParityCCC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallParityCCC (const I3C_TypeDef * I3Cx)

Function description

Check if stall on Parity bit of Command Code byte is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLC LL_I3C_IsEnabledStallParityCCC

LL_I3C_EnableStallParityData

Function name

__STATIC_INLINE void LL_I3C_EnableStallParityData (I3C_TypeDef * I3Cx)

Function description

Set stall on Parity bit of Data bytes (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_EnableStallParityData

LL_I3C_DisableStallParityData

Function name

__STATIC_INLINE void LL_I3C_DisableStallParityData (I3C_TypeDef * I3Cx)

Function description

Disable stall on Parity bit of Data bytes (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_DisableStallParityData

LL_I3C_IsEnabledStallParityData

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallParityData (const I3C_TypeDef * I3Cx)

Function description

Check if stall on Parity bit of Data bytes is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLD LL_I3C_IsEnabledStallParityData

LL_I3C_EnableStallTbit

Function name

__STATIC_INLINE void LL_I3C_EnableStallTbit (I3C_TypeDef * I3Cx)

Function description

Set stall on T bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_EnableStallTbit

LL_I3C_DisableStallTbit

Function name

__STATIC_INLINE void LL_I3C_DisableStallTbit (I3C_TypeDef * I3Cx)

Function description

Disable stall on T bit (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- This bit can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_DisableStallTbit

LL_I3C_IsEnabledStallTbit

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledStallTbit (const I3C_TypeDef * I3Cx)

Function description

Check if stall on T bit is enabled or disabled (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TIMINGR2 STALLT LL_I3C_IsEnabledStallTbit

LL_I3C_SetDeviceCapabilityOnBus

Function name

__STATIC_INLINE void LL_I3C_SetDeviceCapabilityOnBus (I3C_TypeDef * I3Cx, uint32_t DeviceCapabilityOnBus)

Function description

Configure the Device Capability on Bus as Target or Controller (MIPI Bus Characteristics Register BCR6).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceCapabilityOnBus:** This parameter can be one of the following values:
  - LL_I3C_DEVICE_ROLE_AS_TARGET
  - LL_I3C_DEVICE_ROLE_AS_CONTROLLER

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR6 LL_I3C_SetDeviceCapabilityOnBus

LL_I3C_GetDeviceCapabilityOnBus

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceCapabilityOnBus (const I3C_TypeDef * I3Cx)

Function description

Get the Device Capability on Bus as Target or Controller (MIPI Bus Characteristics Register BCR6).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_DEVICE_ROLE_AS_TARGET
  - LL_I3C_DEVICE_ROLE_AS_CONTROLLER

Reference Manual to LL API cross reference:

- BCR BCR6 LL_I3C_GetDeviceCapabilityOnBus

LL_I3C_SetDeviceIBIPayload

Function name

__STATIC_INLINE void LL_I3C_SetDeviceIBIPayload (I3C_TypeDef * I3Cx, uint32_t DeviceIBIPayload)

Function description

Configure the Device IBI Payload (MIPI Bus Characteristics Register BCR2).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceIBIPayload:** This parameter can be one of the following values:
  - LL_I3C_IBI_NO_ADDITIONAL_DATA
  - LL_I3C_IBI_ADDITIONAL_DATA

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR2 LL_I3C_SetDeviceIBIPayload

LL_I3C_GetDeviceIBIPayload

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceIBIPayload (const I3C_TypeDef * I3Cx)

Function description

Get the Device IBI Payload (MIPI Bus Characteristics Register BCR2).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_IBI_NO_ADDITIONAL_DATA
  - LL_I3C_IBI_ADDITIONAL_DATA

Reference Manual to LL API cross reference:

- BCR BCR2 LL_I3C_GetDeviceIBIPayload

LL_I3C_SetDataSpeedLimitation

Function name

__STATIC_INLINE void LL_I3C_SetDataSpeedLimitation (I3C_TypeDef * I3Cx, uint32_t DataSpeedLimitation)

Function description

Configure the Data Speed Limitation (limitation, as described by I3C_GETMXDSR).

Parameters

- **I3Cx:** I3C Instance.
- **DataSpeedLimitation:** This parameter can be one of the following values:
  - LL_I3C_NO_DATA_SPEED_LIMITATION
  - LL_I3C_MAX_DATA_SPEED_LIMITATION

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- BCR BCR0 LL_I3C_SetDataSpeedLimitation

LL_I3C_GetDataSpeedLimitation

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataSpeedLimitation (const I3C_TypeDef * I3Cx)

Function description

Get the Data Speed Limitation (limitation, as described by I3C_GETMXDSR).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_NO_DATA_SPEED_LIMITATION
  - LL_I3C_MAX_DATA_SPEED_LIMITATION

Reference Manual to LL API cross reference:

- BCR BCR0 LL_I3C_GetDataSpeedLimitation

LL_I3C_SetDeviceCharacteristics

Function name

__STATIC_INLINE void LL_I3C_SetDeviceCharacteristics (I3C_TypeDef * I3Cx, uint32_t DeviceCharacteristics)

Function description

Configure the Device Characteristics Register (DCR).

Parameters

- **I3Cx:** I3C Instance.
- **DeviceCharacteristics:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I3C is disabled (EN = 0).
- Refer MIPI web site for the list of device code available.

Reference Manual to LL API cross reference:

- DCR DC LL_I3C_SetDeviceCharacteristics

LL_I3C_GetDeviceCharacteristics

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDeviceCharacteristics (const I3C_TypeDef * I3Cx)

Function description

Get the Device Characteristics Register (DCR).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFF.

Notes

- Refer MIPI web site to associated value with the list of device code available.

Reference Manual to LL API cross reference:

- DCR DCR LL_I3C_GetDeviceCharacteristics

LL_I3C_SetPendingReadMDB

Function name

__STATIC_INLINE void LL_I3C_SetPendingReadMDB (I3C_TypeDef * I3Cx, uint32_t PendingReadMDB)

Function description

Configure IBI MDB support for pending read notification.

Parameters

- **I3Cx:** I3C Instance.
- **PendingReadMDB:** This parameter can be one of the following values:
  - LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION
  - LL_I3C_MDB_PENDING_READ_NOTIFICATION

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETCAPR CAPPEND LL_I3C_SetPendingReadMDB

LL_I3C_GetPendingReadMDB

Function name

__STATIC_INLINE uint32_t LL_I3C_GetPendingReadMDB (const I3C_TypeDef * I3Cx)

Function description

Get IBI MDB support for pending read notification value.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION
  - LL_I3C_MDB_PENDING_READ_NOTIFICATION

Reference Manual to LL API cross reference:

- GETCAPR CAPPEND LL_I3C_GetPendingReadMDB

LL_I3C_SetGrpAddrHandoffSupport

Function name

__STATIC_INLINE void LL_I3C_SetGrpAddrHandoffSupport (I3C_TypeDef * I3Cx, uint32_t GrpAddrHandoffSupport)

Function description

Configure the Group Management Support bit of MSTCAP1.

Parameters

- **I3Cx:** I3C Instance.
- **GrpAddrHandoffSupport:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED
  - LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CRCAPR CAPGRP LL_I3C_SetGrpAddrHandoffSupport

LL_I3C_GetGrpAddrHandoffSupport

Function name

__STATIC_INLINE uint32_t LL_I3C_GetGrpAddrHandoffSupport (const I3C_TypeDef * I3Cx)

Function description

Get the Group Management Support bit of MSTCAP1.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED
  - LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Reference Manual to LL API cross reference:

- CRCAPR CAPGRP LL_I3C_GetGrpAddrHandoffSupport

LL_I3C_SetControllerHandoffDelayed

Function name

__STATIC_INLINE void LL_I3C_SetControllerHandoffDelayed (I3C_TypeDef * I3Cx, uint32_t ControllerHandoffDelayed)

Function description

Configure the Delayed Controller Handoff bit in MSTCAP2.

Parameters

- **I3Cx:** I3C Instance.
- **ControllerHandoffDelayed:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_NOT_DELAYED
  - LL_I3C_HANDOFF_DELAYED

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- CRCAPR CAPDHOFF LL_I3C_SetControllerHandoffDelayed

LL_I3C_GetControllerHandoffDelayed

Function name

__STATIC_INLINE uint32_t LL_I3C_GetControllerHandoffDelayed (const I3C_TypeDef * I3Cx)

Function description

Get the Delayed Controller Handoff bit in MSTCAP2.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_NOT_DELAYED
  - LL_I3C_HANDOFF_DELAYED

Reference Manual to LL API cross reference:

- CRCAPR CAPDHOFF LL_I3C_GetControllerHandoffDelayed

LL_I3C_SetHandoffActivityState

Function name

__STATIC_INLINE void LL_I3C_SetHandoffActivityState (I3C_TypeDef * I3Cx, uint32_t HandoffActivityState)

Function description

Configure the Activity State after controllership handoff.

Parameters

- **I3Cx:** I3C Instance.
- **HandoffActivityState:** This parameter can be one of the following values:
  - LL_I3C_HANDOFF_ACTIVITY_STATE_0
  - LL_I3C_HANDOFF_ACTIVITY_STATE_1
  - LL_I3C_HANDOFF_ACTIVITY_STATE_2
  - LL_I3C_HANDOFF_ACTIVITY_STATE_3

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR HOFFAS LL_I3C_SetHandoffActivityState

LL_I3C_GetHandoffActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetHandoffActivityState (const I3C_TypeDef * I3Cx)

Function description

Get the Activity State after controllership handoff.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_HANDOFF_ACTIVITY_STATE_0
  - LL_I3C_HANDOFF_ACTIVITY_STATE_1
  - LL_I3C_HANDOFF_ACTIVITY_STATE_2
  - LL_I3C_HANDOFF_ACTIVITY_STATE_3

Reference Manual to LL API cross reference:

- GETMXDSR HOFFAS LL_I3C_GetHandoffActivityState

LL_I3C_SetMaxDataSpeedFormat

Function name

__STATIC_INLINE void LL_I3C_SetMaxDataSpeedFormat (I3C_TypeDef * I3Cx, uint32_t MaxDataSpeedFormat)

Function description

Configure the Max Data Speed Format response for GETMXDS CCC.

Parameters

- **I3Cx:** I3C Instance.
- **MaxDataSpeedFormat:** This parameter can be one of the following values:
  - LL_I3C_GETMXDS_FORMAT_1
  - LL_I3C_GETMXDS_FORMAT_2_LSB
  - LL_I3C_GETMXDS_FORMAT_2_MID
  - LL_I3C_GETMXDS_FORMAT_2_MSB

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR FMT LL_I3C_SetMaxDataSpeedFormat

LL_I3C_GetMaxDataSpeedFormat

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMaxDataSpeedFormat (const I3C_TypeDef * I3Cx)

Function description

Get the Max Data Speed Format response for GETMXDS CCC.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_GETMXDS_FORMAT_1
  - LL_I3C_GETMXDS_FORMAT_2_LSB
  - LL_I3C_GETMXDS_FORMAT_2_MID
  - LL_I3C_GETMXDS_FORMAT_2_MSB

Reference Manual to LL API cross reference:

- GETMXDSR FMT LL_I3C_GetMaxDataSpeedFormat

LL_I3C_SetMiddleByteTurnAround

Function name

__STATIC_INLINE void LL_I3C_SetMiddleByteTurnAround (I3C_TypeDef * I3Cx, uint32_t MiddleByteTurnAround)

Function description

Configure the Middle byte of MaxRdTurn field of GETMXDS CCC Format 2 with turnaround.

Parameters

- **I3Cx:** I3C Instance.
- **MiddleByteTurnAround:** This parameter must be a value between Min_Data=0 and Max_Data=0xF.

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR RDTURN LL_I3C_SetMiddleByteTurnAround

LL_I3C_GetMiddleByteTurnAround

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMiddleByteTurnAround (const I3C_TypeDef * I3Cx)

Function description

Get the value of Middle byte of MaxRdTurn field of GETMXDS CCC Format 2 with turnaround.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xF.

Reference Manual to LL API cross reference:

- GETMXDSR RDTURN LL_I3C_GetMiddleByteTurnAround

LL_I3C_SetDataTurnAroundTime

Function name

__STATIC_INLINE void LL_I3C_SetDataTurnAroundTime (I3C_TypeDef * I3Cx, uint32_t DataTurnAroundTime)

Function description

Configure clock-to-data turnaround time.

Parameters

- **I3Cx:** I3C Instance.
- **DataTurnAroundTime:** This parameter can be one of the following values:
  - LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS
  - LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- GETMXDSR TSCO LL_I3C_SetDataTurnAroundTime

LL_I3C_GetDataTurnAroundTime

Function name

__STATIC_INLINE uint32_t LL_I3C_GetDataTurnAroundTime (const I3C_TypeDef * I3Cx)

Function description

Get clock-to-data turnaround time.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS
  - LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

Reference Manual to LL API cross reference:

- GETMXDSR TSCO LL_I3C_GetDataTurnAroundTime

LL_I3C_SetMIPIInstanceID

Function name

__STATIC_INLINE void LL_I3C_SetMIPIInstanceID (I3C_TypeDef * I3Cx, uint32_t MIPIInstanceID)

Function description

Configure the MIPI Instance ID.

Parameters

- **I3Cx:** I3C Instance.
- **MIPIInstanceID:** This parameter must be a value between Min_Data=0 and Max_Data=0xF.

Return values

- **None:**

Notes

- Those bits can be programmed when the I3C is disabled (EN = 0).

Reference Manual to LL API cross reference:

- EPIDR MIPIID LL_I3C_SetMIPIInstanceID

LL_I3C_GetMIPIInstanceID

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMIPIInstanceID (const I3C_TypeDef * I3Cx)

Function description

Get the MIPI Instance ID.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xF.

Reference Manual to LL API cross reference:

- EPIDR MIPIID LL_I3C_GetMIPIInstanceID

LL_I3C_GetIDTypeSelector

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIDTypeSelector (const I3C_TypeDef * I3Cx)

Function description

Get the ID type selector.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x1

Reference Manual to LL API cross reference:

- EPIDR IDTSEL LL_I3C_GetIDTypeSelector

LL_I3C_GetMIPIManufacturerID

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMIPIManufacturerID (const I3C_TypeDef * I3Cx)

Function description

Get the MIPI Manufacturer ID.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0x7FFF.

Reference Manual to LL API cross reference:

- EPIDR MIPIMID LL_I3C_GetMIPIManufacturerID

LL_I3C_RequestRxFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestRxFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a reception Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RXFLUSH LL_I3C_RequestRxFIFOFlush

LL_I3C_RequestTxFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestTxFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a transmission Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TXFLUSH LL_I3C_RequestTxFIFOFlush

LL_I3C_RequestStatusFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestStatusFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a Status Data FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SFLUSH LL_I3C_RequestStatusFIFOFlush

LL_I3C_GetActivityState

Function name

__STATIC_INLINE uint32_t LL_I3C_GetActivityState (const I3C_TypeDef * I3Cx)

Function description

Get Activity state of Controller on the I3C Bus (Target only).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_BUS_ACTIVITY_STATE_0
  - LL_I3C_BUS_ACTIVITY_STATE_1
  - LL_I3C_BUS_ACTIVITY_STATE_2
  - LL_I3C_BUS_ACTIVITY_STATE_3

Reference Manual to LL API cross reference:

- DEVR0 AS LL_I3C_GetActivityState

LL_I3C_GetResetAction

Function name

__STATIC_INLINE uint32_t LL_I3C_GetResetAction (const I3C_TypeDef * I3Cx)

Function description

Get Reset Action (Target only).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_RESET_ACTION_NONE
  - LL_I3C_RESET_ACTION_PARTIAL
  - LL_I3C_RESET_ACTION_FULL

Reference Manual to LL API cross reference:

- DEVR0 RSTACT LL_I3C_GetResetAction

LL_I3C_RequestControlFIFOFlush

Function name

__STATIC_INLINE void LL_I3C_RequestControlFIFOFlush (I3C_TypeDef * I3Cx)

Function description

Request a Control word FIFO Flush.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CFLUSH LL_I3C_RequestControlFIFOFlush

LL_I3C_RequestTransfer

Function name

__STATIC_INLINE void LL_I3C_RequestTransfer (I3C_TypeDef * I3Cx)

Function description

Request a Transfer start.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Notes

- After request, the current instruction in Control Register is executed on I3C Bus.

Reference Manual to LL API cross reference:

- CFGR TSFSET LL_I3C_RequestTransfer

LL_I3C_ControllerHandleMessage

Function name

__STATIC_INLINE void LL_I3C_ControllerHandleMessage (I3C_TypeDef * I3Cx, uint32_t TargetAddr, uint32_t TransferSize, uint32_t Direction, uint32_t MessageType, uint32_t EndMode)

Function description

Handles I3C Message content on the I3C Bus as Controller.

Parameters

- **I3Cx:** I3C Instance.
- **TargetAddr:** Specifies the target address to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=0xFF.
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=65535.
- **Direction:** This parameter can be one of the following values:
  - LL_I3C_DIRECTION_WRITE
  - LL_I3C_DIRECTION_READ
- **MessageType:** This parameter can be one of the following values:
  - LL_I3C_CONTROLLER_MTYPE_RELEASE
  - LL_I3C_CONTROLLER_MTYPE_HEADER
  - LL_I3C_CONTROLLER_MTYPE_PRIVATE
  - LL_I3C_CONTROLLER_MTYPE_DIRECT
  - LL_I3C_CONTROLLER_MTYPE_LEGACY_I2C
- **EndMode:** This parameter can be one of the following values:
  - LL_I3C_GENERATE_STOP
  - LL_I3C_GENERATE_RESTART

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADD LL_I3C_ControllerHandleMessage
- CR DCNT LL_I3C_ControllerHandleMessage
- CR RNW LL_I3C_ControllerHandleMessage
- CR MTYPE LL_I3C_ControllerHandleMessage
- CR MEND LL_I3C_ControllerHandleMessage

LL_I3C_ControllerHandleCCC

Function name

__STATIC_INLINE void LL_I3C_ControllerHandleCCC (I3C_TypeDef * I3Cx, uint32_t CCCValue, uint32_t AddByteSize, uint32_t EndMode)

Function description

Handles I3C Common Command Code content on the I3C Bus as Controller.

Parameters

- **I3Cx:** I3C Instance.
- **CCCValue:** Specifies the Command Code to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=0x1FF.
- **AddByteSize:** Specifies the number of CCC additional bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=65535.
- **EndMode:** This parameter can be one of the following values:
  - LL_I3C_GENERATE_STOP
  - LL_I3C_GENERATE_RESTART

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CCC LL_I3C_ControllerHandleCCC
- CR DCNT LL_I3C_ControllerHandleCCC
- CR MTYPE LL_I3C_ControllerHandleCCC
- CR MEND LL_I3C_ControllerHandleCCC

LL_I3C_TargetHandleMessage

Function name

__STATIC_INLINE void LL_I3C_TargetHandleMessage (I3C_TypeDef * I3Cx, uint32_t MessageType, uint32_t IBISize)

Function description

Handles I3C Message content on the I3C Bus as Target.

Parameters

- **I3Cx:** I3C Instance.
- **MessageType:** This parameter can be one of the following values:
  - LL_I3C_TARGET_MTYPE_HOT_JOIN
  - LL_I3C_TARGET_MTYPE_CONTROLLER_ROLE_REQ
  - LL_I3C_TARGET_MTYPE_IBI
- **IBISize:** Specifies the number of IBI bytes. This parameter must be a value between Min_Data=0 and Max_Data=65535.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MTYPE LL_I3C_TargetHandleMessage
- CR DCNT LL_I3C_TargetHandleMessage

LL_I3C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I3C_ReceiveData8 (const I3C_TypeDef * I3Cx)

Function description

Read Receive Data Byte register.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDB0 LL_I3C_ReceiveData8

LL_I3C_TransmitData8

Function name

__STATIC_INLINE void LL_I3C_TransmitData8 (I3C_TypeDef * I3Cx, uint8_t Data)

Function description

Write in Transmit Data Byte Register.

Parameters

- **I3Cx:** I3C Instance.
- **Data:** This parameter must be a value between Min_Data=0 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDB0 LL_I3C_TransmitData8

LL_I3C_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_I3C_ReceiveData32 (const I3C_TypeDef * I3Cx)

Function description

Read Receive Data Word register.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFFFFFF

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last data byte received, LSB correspond to first data byte received.

Reference Manual to LL API cross reference:

- RDWR RDWR LL_I3C_ReceiveData32

LL_I3C_TransmitData32

Function name

__STATIC_INLINE void LL_I3C_TransmitData32 (I3C_TypeDef * I3Cx, uint32_t Data)

Function description

Write in Transmit Data Word Register.

Parameters

- **I3Cx:** I3C Instance.
- **Data:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last data byte transmitted, LSB correspond to first data byte transmitted.

Reference Manual to LL API cross reference:

- TDWR TDWR LL_I3C_TransmitData32

LL_I3C_SetIBIPayload

Function name

__STATIC_INLINE void LL_I3C_SetIBIPayload (I3C_TypeDef * I3Cx, uint32_t OwnIBIPayload)

Function description

Configure the IBI data payload to be sent during IBI (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **OwnIBIPayload:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last IBI data byte, LSB correspond to first IBI data byte.

Reference Manual to LL API cross reference:

- IBIDR IBIDR LL_I3C_SetIBIPayload

LL_I3C_GetIBIPayload

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIBIPayload (const I3C_TypeDef * I3Cx)

Function description

Get the own IBI data payload (target mode), or get the Target IBI received (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFFFFFF

Notes

- Content of register is filled in Little Endian. Mean MSB correspond to last IBI data byte, LSB correspond to first IBI data byte.

Reference Manual to LL API cross reference:

- IBIDR IBIDR LL_I3C_GetIBIPayload

LL_I3C_GetNbIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_GetNbIBIAddData (const I3C_TypeDef * I3Cx)

Function description

Get the number of data bytes received when reading IBI data (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0x7

Reference Manual to LL API cross reference:

- RMR IBIRDCNT LL_I3C_GetNbIBIAddData

LL_I3C_GetIBITargetAddr

Function name

__STATIC_INLINE uint32_t LL_I3C_GetIBITargetAddr (const I3C_TypeDef * I3Cx)

Function description

Get the target address received during accepted IBI or Controller-role request.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0x3F

Reference Manual to LL API cross reference:

- RMR RADD LL_I3C_GetIBITargetAddr

LL_I3C_ConfigTxPreload

Function name

__STATIC_INLINE void LL_I3C_ConfigTxPreload (I3C_TypeDef * I3Cx, uint16_t TxDataCount)

Function description

Set TX FIFO Preload (target mode).

Parameters

- **I3Cx:** I3C Instance.
- **TxDataCount:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Set high by Software, cleared by hardware when all the bytes to transmit have been loaded to TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR PRELOAD LL_I3C_ConfigTxPreload
- TGTTDR TDCNT LL_I3C_ConfigTxPreload

LL_I3C_IsActiveTxPreload

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveTxPreload (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of TX FIFO preload (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- Set high by Software, cleared by hardware when all the bytes to transmit have been loaded to TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR PRELOAD LL_I3C_IsActiveTxPreload

LL_I3C_GetTxPreloadDataCount

Function name

__STATIC_INLINE uint16_t LL_I3C_GetTxPreloadDataCount (const I3C_TypeDef * I3Cx)

Function description

Get the number of bytes to transmit (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFF

Notes

- The return value correspond to the remaining number of bytes to load in TX FIFO.

Reference Manual to LL API cross reference:

- TGTTDR TDCNT LL_I3C_GetTxPreloadDataCount

LL_I3C_GetXferDataCount

Function name

__STATIC_INLINE uint32_t LL_I3C_GetXferDataCount (const I3C_TypeDef * I3Cx)

Function description

Get the number of data during a Transfer.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFFFF

Notes

- The return value correspond to number of transmitted bytes reported during Address Assignment process in Target mode. The return value correspond to number of target detected during Address Assignment process in Controller mode. The return value correspond to number of data bytes read from or sent to the I3C bus during the message link to MID current value.

Reference Manual to LL API cross reference:

- SR XDCNT LL_I3C_GetXferDataCount

LL_I3C_IsTargetAbortPrivateRead

Function name

__STATIC_INLINE uint32_t LL_I3C_IsTargetAbortPrivateRead (const I3C_TypeDef * I3Cx)

Function description

Indicates if a Target abort a private read command.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ABT LL_I3C_IsTargetAbortPrivateRead

LL_I3C_GetMessageDirection

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageDirection (const I3C_TypeDef * I3Cx)

Function description

Get Direction of the Message.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_MESSAGE_DIRECTION_WRITE
  - LL_I3C_MESSAGE_DIRECTION_READ

Reference Manual to LL API cross reference:

- SR DIR LL_I3C_GetMessageDirection

LL_I3C_GetMessageIdentifier

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageIdentifier (const I3C_TypeDef * I3Cx)

Function description

Get Message identifier.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF, representing the internal hardware counter value.

Reference Manual to LL API cross reference:

- SR MID LL_I3C_GetMessageIdentifier

LL_I3C_GetMessageErrorCode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetMessageErrorCode (const I3C_TypeDef * I3Cx)

Function description

Get Message error code.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I3C_CONTROLLER_ERROR_CE0
  - LL_I3C_CONTROLLER_ERROR_CE1
  - LL_I3C_CONTROLLER_ERROR_CE2
  - LL_I3C_CONTROLLER_ERROR_CE3
  - LL_I3C_TARGET_ERROR_TE0
  - LL_I3C_TARGET_ERROR_TE1
  - LL_I3C_TARGET_ERROR_TE2
  - LL_I3C_TARGET_ERROR_TE3
  - LL_I3C_TARGET_ERROR_TE4
  - LL_I3C_TARGET_ERROR_TE5
  - LL_I3C_TARGET_ERROR_TE6

Reference Manual to LL API cross reference:

- SER CODERR LL_I3C_GetMessageErrorCode

LL_I3C_GetReceiveCommandCode

Function name

__STATIC_INLINE uint32_t LL_I3C_GetReceiveCommandCode (const I3C_TypeDef * I3Cx)

Function description

Get CCC code of received command.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **Value:** between Min_Data=0 to Max_Data=0xFF.

Reference Manual to LL API cross reference:

- RMR RCODE LL_I3C_GetReceiveCommandCode

LL_I3C_SetTargetDynamicAddress

Function name

__STATIC_INLINE void LL_I3C_SetTargetDynamicAddress (I3C_TypeDef * I3Cx, uint32_t TargetId, uint32_t DynamicAddr)

Function description

Set Dynamic Address assigned to target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4
- **DynamicAddr:** Value between Min_Data=0 to Max_Data=0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_SetTargetDynamicAddress

LL_I3C_GetTargetDynamicAddress

Function name

__STATIC_INLINE uint32_t LL_I3C_GetTargetDynamicAddress (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Get Dynamic Address assigned to target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **Value:** between Min_Data=0 to Max_Data=0x7F

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_GetTargetDynamicAddress

LL_I3C_EnableTargetIBIAck

Function name

__STATIC_INLINE void LL_I3C_EnableTargetIBIAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable IBI Acknowledgement from target x(controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- The bit DIS is automatically set when CRACK or IBIACK are set. This mean DEVRX register access is not allowed. Reset CRACK and IBIACK will reset DIS bit.

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_EnableTargetIBIAck

LL_I3C_DisableTargetIBIAck

Function name

__STATIC_INLINE void LL_I3C_DisableTargetIBIAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable IBI Acknowledgement from target x (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_DisableTargetIBIAck

LL_I3C_IsEnabledTargetIBIAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledTargetIBIAck (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if IBI from target x will be Acknowledged or Not Acknowledged (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX IBIACK LL_I3C_IsEnabledTargetIBIAck

LL_I3C_EnableTargetCRAck

Function name

__STATIC_INLINE void LL_I3C_EnableTargetCRAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable Controller-role Request Acknowledgement from target x(controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- The bit DIS is automatically set when CRACK or IBIACK are set. This mean DEVRX register access is not allowed. Reset CRACK and IBIACK will reset DIS bit.

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_EnableTargetCRAck

LL_I3C_DisableTargetCRAck

Function name

__STATIC_INLINE void LL_I3C_DisableTargetCRAck (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable Controller-role Request Acknowledgement from target x (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_DisableTargetCRAck

LL_I3C_IsEnabledTargetCRAck

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledTargetCRAck (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if Controller-role Request from target x will be Acknowledged or Not Acknowledged (controller mode).

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX CRACK LL_I3C_IsEnabledTargetCRAck

LL_I3C_EnableIBIAddData

Function name

__STATIC_INLINE void LL_I3C_EnableIBIAddData (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_EnableIBIAddData

LL_I3C_DisableIBIAddData

Function name

__STATIC_INLINE void LL_I3C_DisableIBIAddData (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_DisableIBIAddData

LL_I3C_IsEnabledIBIAddData

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIBIAddData (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if additional Mandatory Data Byte (MDB) follows the accepted IBI from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX IBIDEN LL_I3C_IsEnabledIBIAddData

LL_I3C_EnableFrameSuspend

Function name

__STATIC_INLINE void LL_I3C_EnableFrameSuspend (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Enable Suspension of Current transfer during IBI treatment.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- When set, this feature will allow controller to send a Stop condition and CR FIFO is flushed after IBI treatment. Software has to rewrite instructions in Control Register to start a new transfer.

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_EnableFrameSuspend

LL_I3C_DisableFrameSuspend

Function name

__STATIC_INLINE void LL_I3C_DisableFrameSuspend (I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Disable Suspension of Current transfer during IBI treatment.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **None:**

Notes

- When set, this feature will allow controller to continue CR FIFO treatment after IBI treatment.

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_DisableFrameSuspend

LL_I3C_IsFrameMustBeSuspended

Function name

__STATIC_INLINE uint32_t LL_I3C_IsFrameMustBeSuspended (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if I3C transfer must be Suspended or not Suspended during IBI treatment from target x.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DEVRX SUSP LL_I3C_IsFrameMustBeSuspended

LL_I3C_IsAllowedPayloadUpdate

Function name

__STATIC_INLINE uint32_t LL_I3C_IsAllowedPayloadUpdate (const I3C_TypeDef * I3Cx, uint32_t TargetId)

Function description

Indicates if update of the Device Characteristics Register is Allowed or Not Allowed.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4

Return values

- **State:** of bit (1 or 0).

Notes

- Used to prevent software writing during reception of an IBI or Controller-role Request from target x.

Reference Manual to LL API cross reference:

- DEVRX DIS LL_I3C_IsAllowedPayloadUpdate

LL_I3C_ConfigDeviceCapabilities

Function name

__STATIC_INLINE void LL_I3C_ConfigDeviceCapabilities (I3C_TypeDef * I3Cx, uint32_t TargetId, uint32_t DynamicAddr, uint32_t IBIAck, uint32_t IBIAddData, uint32_t CRAck)

Function description

Set I3C bus devices configuration.

Parameters

- **I3Cx:** I3C Instance.
- **TargetId:** This parameter must be a value between Min_Data=1 and Max_Data=4
- **DynamicAddr:** Value between Min_Data=0 to Max_Data=0x7F
- **IBIAck:** Value This parameter can be one of the following values:
  - LL_I3C_IBI_CAPABILITY
  - LL_I3C_IBI_NO_CAPABILITY
- **IBIAddData:** This parameter can be one of the following values:
  - LL_I3C_IBI_DATA_ENABLE
  - LL_I3C_IBI_DATA_DISABLE
- **CRAck:** This parameter can be one of the following values:
  - LL_I3C_CR_CAPABILITY
  - LL_I3C_CR_NO_CAPABILITY

Return values

- **None:**

Notes

- This function is called only when the I3C instance is initialized as controller. This function can be called by the controller application to help the automatic treatment when target have capability of IBI and/or Control-Role.

Reference Manual to LL API cross reference:

- DEVRX DA LL_I3C_ConfigDeviceCapabilities
- DEVRX IBIACK LL_I3C_ConfigDeviceCapabilities
- DEVRX IBIDEN LL_I3C_ConfigDeviceCapabilities
- DEVRX CRACK LL_I3C_ConfigDeviceCapabilities

LL_I3C_IsActiveFlag_CFE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CFE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control FIFO Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR CFEF LL_I3C_IsActiveFlag_CFE

LL_I3C_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXFE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Transmit FIFO Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR TXFEF LL_I3C_IsActiveFlag_TXFE

LL_I3C_IsActiveFlag_CFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CFNF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- When a transfer is ongoing, the Control FIFO shall not be written unless this flag is set.

Reference Manual to LL API cross reference:

- EVR CFNFF LL_I3C_IsActiveFlag_CFNF

LL_I3C_IsActiveFlag_SFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_SFNE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Status FIFO Not Empty flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This flag is updated only when the FIFO is used, mean SMODE = 1.

Reference Manual to LL API cross reference:

- EVR SFNEF LL_I3C_IsActiveFlag_SFNE

LL_I3C_IsActiveFlag_TXFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXFNF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Transmit FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- When a transfer is ongoing, the Transmit FIFO shall not be written unless this flag is set.

Reference Manual to LL API cross reference:

- EVR TXFNFF LL_I3C_IsActiveFlag_TXFNF

LL_I3C_IsActiveFlag_RXFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXFNE (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Receive FIFO Not Full flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RXFNEF LL_I3C_IsActiveFlag_RXFNE

LL_I3C_IsActiveFlag_RXLAST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXLAST (const I3C_TypeDef * I3Cx)

Function description

Indicates that the last Receive byte is available.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RXLASTF LL_I3C_IsActiveFlag_RXLAST

LL_I3C_IsActiveFlag_TXLAST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_TXLAST (const I3C_TypeDef * I3Cx)

Function description

Indicates that the last Transmit byte is written in FIFO.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR TXLASTF LL_I3C_IsActiveFlag_TXLAST

LL_I3C_IsActiveFlag_FC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_FC (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Frame Complete flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR FCF LL_I3C_IsActiveFlag_FC

LL_I3C_IsActiveFlag_RXTGTEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RXTGTEND (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Reception Target End flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- This flag is set only when status FIFO is not used, mean SMODE = 0.

Reference Manual to LL API cross reference:

- EVR RXTGTENDF LL_I3C_IsActiveFlag_RXTGTEND

LL_I3C_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Error flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR ERRF LL_I3C_IsActiveFlag_ERR

LL_I3C_IsActiveFlag_IBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_IBI (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of IBI flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR IBIF LL_I3C_IsActiveFlag_IBI

LL_I3C_IsActiveFlag_IBIEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_IBIEND (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of IBI End flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR IBIENDF LL_I3C_IsActiveFlag_IBIEND

LL_I3C_IsActiveFlag_CR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Controller-role Request flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR CRF LL_I3C_IsActiveFlag_CR

LL_I3C_IsActiveFlag_CRUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_CRUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Controller-role Request Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR BCUPDF LL_I3C_IsActiveFlag_CRUPD

LL_I3C_IsActiveFlag_HJ

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_HJ (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Hot Join flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR HJF LL_I3C_IsActiveFlag_HJ

LL_I3C_IsActiveFlag_WKP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_WKP (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Wake Up flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR WKPF LL_I3C_IsActiveFlag_WKP

LL_I3C_IsActiveFlag_GET

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_GET (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Get flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR GETF LL_I3C_IsActiveFlag_GET

LL_I3C_IsActiveFlag_STA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_STA (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Get Status flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR STAF LL_I3C_IsActiveFlag_STA

LL_I3C_IsActiveFlag_DAUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DAUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Dynamic Address Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR DAUPDF LL_I3C_IsActiveFlag_DAUPD

LL_I3C_IsActiveFlag_MWLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_MWLUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Max Write Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR MWLUPDF LL_I3C_IsActiveFlag_MWLUPD

LL_I3C_IsActiveFlag_MRLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_MRLUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Max Read Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR MRLUPDF LL_I3C_IsActiveFlag_MRLUPD

LL_I3C_IsActiveFlag_RST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_RST (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Reset flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR RSTF LL_I3C_IsActiveFlag_RST

LL_I3C_IsActiveFlag_ASUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ASUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Active State flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR ASUPDF LL_I3C_IsActiveFlag_ASUPD

LL_I3C_IsActiveFlag_INTUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_INTUPD (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Interrupt Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR INTUPDF LL_I3C_IsActiveFlag_INTUPD

LL_I3C_IsActiveFlag_DEF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DEF (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Define List Targets flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR DEFF LL_I3C_IsActiveFlag_DEF

LL_I3C_IsActiveFlag_GRP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_GRP (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Define List Group Addresses flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EVR GRPF LL_I3C_IsActiveFlag_GRP

LL_I3C_IsActiveFlag_PERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_PERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Protocol Error flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER PERR LL_I3C_IsActiveFlag_PERR

LL_I3C_IsActiveFlag_STALL

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_STALL (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of SCL Stall Error flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER STALL LL_I3C_IsActiveFlag_STALL

LL_I3C_IsActiveFlag_DOVR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DOVR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of RX or TX FIFO Overrun flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DOVR LL_I3C_IsActiveFlag_DOVR

LL_I3C_IsActiveFlag_COVR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_COVR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Control or Status FIFO Overrun flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER COVR LL_I3C_IsActiveFlag_COVR

LL_I3C_IsActiveFlag_ANACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_ANACK (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Address not acknowledged flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER ANACK LL_I3C_IsActiveFlag_ANACK

LL_I3C_IsActiveFlag_DNACK

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DNACK (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Data not acknowledged flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DNACK LL_I3C_IsActiveFlag_DNACK

LL_I3C_IsActiveFlag_DERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsActiveFlag_DERR (const I3C_TypeDef * I3Cx)

Function description

Indicates the status of Data error flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SER DERR LL_I3C_IsActiveFlag_DERR

LL_I3C_EnableIT_CFNF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CFNF (I3C_TypeDef * I3Cx)

Function description

Enable Control FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_EnableIT_CFNF

LL_I3C_DisableIT_CFNF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CFNF (I3C_TypeDef * I3Cx)

Function description

Disable Control FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_DisableIT_CFNF

LL_I3C_IsEnabledIT_CFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CFNF (const I3C_TypeDef * I3Cx)

Function description

Check if Control FIFO Not Full interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CFNFIE LL_I3C_IsEnabledIT_CFNF

LL_I3C_EnableIT_SFNE

Function name

__STATIC_INLINE void LL_I3C_EnableIT_SFNE (I3C_TypeDef * I3Cx)

Function description

Enable Status FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_EnableIT_SFNE

LL_I3C_DisableIT_SFNE

Function name

__STATIC_INLINE void LL_I3C_DisableIT_SFNE (I3C_TypeDef * I3Cx)

Function description

Disable Status FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_DisableIT_SFNE

LL_I3C_IsEnabledIT_SFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_SFNE (const I3C_TypeDef * I3Cx)

Function description

Check if Status FIFO Not Empty interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER SFNEIE LL_I3C_IsEnabledIT_SFNE

LL_I3C_EnableIT_TXFNF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_TXFNF (I3C_TypeDef * I3Cx)

Function description

Enable Transmit FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_EnableIT_TXFNF

LL_I3C_DisableIT_TXFNF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_TXFNF (I3C_TypeDef * I3Cx)

Function description

Disable Transmit FIFO Not Full interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_DisableIT_TXFNF

LL_I3C_IsEnabledIT_TXFNF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_TXFNF (const I3C_TypeDef * I3Cx)

Function description

Check if Transmit FIFO Not Full interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TXFNFIE LL_I3C_IsEnabledIT_TXFNF

LL_I3C_EnableIT_RXFNE

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RXFNE (I3C_TypeDef * I3Cx)

Function description

Enable Receive FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_EnableIT_RXFNE

LL_I3C_DisableIT_RXFNE

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RXFNE (I3C_TypeDef * I3Cx)

Function description

Disable Receive FIFO Not Empty interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_DisableIT_RXFNE

LL_I3C_IsEnabledIT_RXFNE

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RXFNE (const I3C_TypeDef * I3Cx)

Function description

Check if Receive FIFO Not Empty interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXFNEIE LL_I3C_IsEnabledIT_RXFNE

LL_I3C_EnableIT_FC

Function name

__STATIC_INLINE void LL_I3C_EnableIT_FC (I3C_TypeDef * I3Cx)

Function description

Enable Frame Complete interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_EnableIT_FC

LL_I3C_DisableIT_FC

Function name

__STATIC_INLINE void LL_I3C_DisableIT_FC (I3C_TypeDef * I3Cx)

Function description

Disable Frame Complete interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_DisableIT_FC

LL_I3C_IsEnabledIT_FC

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_FC (const I3C_TypeDef * I3Cx)

Function description

Check if Frame Complete interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER FCIE LL_I3C_IsEnabledIT_FC

LL_I3C_EnableIT_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Enable Reception Target End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_EnableIT_RXTGTEND

LL_I3C_DisableIT_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Disable Reception Target End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_DisableIT_RXTGTEND

LL_I3C_IsEnabledIT_RXTGTEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RXTGTEND (const I3C_TypeDef * I3Cx)

Function description

Check if Reception Target End interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXTGTENDIE LL_I3C_IsEnabledIT_RXTGTEND

LL_I3C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I3C_EnableIT_ERR (I3C_TypeDef * I3Cx)

Function description

Enable Error interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_EnableIT_ERR

LL_I3C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I3C_DisableIT_ERR (I3C_TypeDef * I3Cx)

Function description

Disable Error interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_DisableIT_ERR

LL_I3C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_ERR (const I3C_TypeDef * I3Cx)

Function description

Check if Error interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_I3C_IsEnabledIT_ERR

LL_I3C_EnableIT_IBI

Function name

__STATIC_INLINE void LL_I3C_EnableIT_IBI (I3C_TypeDef * I3Cx)

Function description

Enable IBI interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_EnableIT_IBI

LL_I3C_DisableIT_IBI

Function name

__STATIC_INLINE void LL_I3C_DisableIT_IBI (I3C_TypeDef * I3Cx)

Function description

Disable IBI interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_DisableIT_IBI

LL_I3C_IsEnabledIT_IBI

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_IBI (const I3C_TypeDef * I3Cx)

Function description

Check if IBI interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER IBIIE LL_I3C_IsEnabledIT_IBI

LL_I3C_EnableIT_IBIEND

Function name

__STATIC_INLINE void LL_I3C_EnableIT_IBIEND (I3C_TypeDef * I3Cx)

Function description

Enable IBI End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_EnableIT_IBIEND

LL_I3C_DisableIT_IBIEND

Function name

__STATIC_INLINE void LL_I3C_DisableIT_IBIEND (I3C_TypeDef * I3Cx)

Function description

Disable IBI End interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_DisableIT_IBIEND

LL_I3C_IsEnabledIT_IBIEND

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_IBIEND (const I3C_TypeDef * I3Cx)

Function description

Check if IBI End interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER IBIENDIE LL_I3C_IsEnabledIT_IBIEND

LL_I3C_EnableIT_CR

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CR (I3C_TypeDef * I3Cx)

Function description

Enable Controller-role interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_EnableIT_CR

LL_I3C_DisableIT_CR

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CR (I3C_TypeDef * I3Cx)

Function description

Disable Controller-role interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_DisableIT_CR

LL_I3C_IsEnabledIT_CR

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CR (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CRIE LL_I3C_IsEnabledIT_CR

LL_I3C_EnableIT_CRUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_CRUPD (I3C_TypeDef * I3Cx)

Function description

Enable Controller-role Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_EnableIT_CRUPD

LL_I3C_DisableIT_CRUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_CRUPD (I3C_TypeDef * I3Cx)

Function description

Disable Controller-role Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_DisableIT_CRUPD

LL_I3C_IsEnabledIT_CRUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_CRUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Controller-role Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CRUPDIE LL_I3C_IsEnabledIT_CRUPD

LL_I3C_EnableIT_HJ

Function name

__STATIC_INLINE void LL_I3C_EnableIT_HJ (I3C_TypeDef * I3Cx)

Function description

Enable Hot Join interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_EnableIT_HJ

LL_I3C_DisableIT_HJ

Function name

__STATIC_INLINE void LL_I3C_DisableIT_HJ (I3C_TypeDef * I3Cx)

Function description

Disable Hot Join interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_DisableIT_HJ

LL_I3C_IsEnabledIT_HJ

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_HJ (const I3C_TypeDef * I3Cx)

Function description

Check if Hot Join interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER HJIE LL_I3C_IsEnabledIT_HJ

LL_I3C_EnableIT_WKP

Function name

__STATIC_INLINE void LL_I3C_EnableIT_WKP (I3C_TypeDef * I3Cx)

Function description

Enable Wake Up interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_EnableIT_WKP

LL_I3C_DisableIT_WKP

Function name

__STATIC_INLINE void LL_I3C_DisableIT_WKP (I3C_TypeDef * I3Cx)

Function description

Disable Wake Up interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_DisableIT_WKP

LL_I3C_IsEnabledIT_WKP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_WKP (const I3C_TypeDef * I3Cx)

Function description

Check if Wake Up is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER WKPIE LL_I3C_IsEnabledIT_WKP

LL_I3C_EnableIT_GET

Function name

__STATIC_INLINE void LL_I3C_EnableIT_GET (I3C_TypeDef * I3Cx)

Function description

Enable Get Command interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_EnableIT_GET

LL_I3C_DisableIT_GET

Function name

__STATIC_INLINE void LL_I3C_DisableIT_GET (I3C_TypeDef * I3Cx)

Function description

Disable Get Command interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_DisableIT_GET

LL_I3C_IsEnabledIT_GET

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_GET (const I3C_TypeDef * I3Cx)

Function description

Check if Get Command is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER GETIE LL_I3C_IsEnabledIT_GET

LL_I3C_EnableIT_STA

Function name

__STATIC_INLINE void LL_I3C_EnableIT_STA (I3C_TypeDef * I3Cx)

Function description

Enable Get Status interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_EnableIT_STA

LL_I3C_DisableIT_STA

Function name

__STATIC_INLINE void LL_I3C_DisableIT_STA (I3C_TypeDef * I3Cx)

Function description

Disable Get Status interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_DisableIT_STA

LL_I3C_IsEnabledIT_STA

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_STA (const I3C_TypeDef * I3Cx)

Function description

Check if Get Status interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER STAIE LL_I3C_IsEnabledIT_STA

LL_I3C_EnableIT_DAUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_DAUPD (I3C_TypeDef * I3Cx)

Function description

Enable Dynamic Address Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_EnableIT_DAUPD

LL_I3C_DisableIT_DAUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_DAUPD (I3C_TypeDef * I3Cx)

Function description

Disable Dynamic Address Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_DisableIT_DAUPD

LL_I3C_IsEnabledIT_DAUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_DAUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Dynamic Address Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER DAUPDIE LL_I3C_IsEnabledIT_DAUPD

LL_I3C_EnableIT_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Enable Max Write Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_EnableIT_MWLUPD

LL_I3C_DisableIT_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Disable Max Write Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_DisableIT_MWLUPD

LL_I3C_IsEnabledIT_MWLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_MWLUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Max Write Length Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER MWLUPDIE LL_I3C_IsEnabledIT_MWLUPD

LL_I3C_EnableIT_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Enable Max Read Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_EnableIT_MRLUPD

LL_I3C_DisableIT_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Disable Max Read Length Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_DisableIT_MRLUPD

LL_I3C_IsEnabledIT_MRLUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_MRLUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Max Read Length Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER MRLUPDIE LL_I3C_IsEnabledIT_MRLUPD

LL_I3C_EnableIT_RST

Function name

__STATIC_INLINE void LL_I3C_EnableIT_RST (I3C_TypeDef * I3Cx)

Function description

Enable Reset interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_EnableIT_RST

LL_I3C_DisableIT_RST

Function name

__STATIC_INLINE void LL_I3C_DisableIT_RST (I3C_TypeDef * I3Cx)

Function description

Disable Reset interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_DisableIT_RST

LL_I3C_IsEnabledIT_RST

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_RST (const I3C_TypeDef * I3Cx)

Function description

Check if Reset interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RSTIE LL_I3C_IsEnabledIT_RST

LL_I3C_EnableIT_ASUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_ASUPD (I3C_TypeDef * I3Cx)

Function description

Enable Activity State Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_EnableIT_ASUPD

LL_I3C_DisableIT_ASUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_ASUPD (I3C_TypeDef * I3Cx)

Function description

Disable Activity State Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_DisableIT_ASUPD

LL_I3C_IsEnabledIT_ASUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_ASUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Activity State Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ASUPDIE LL_I3C_IsEnabledIT_ASUPD

LL_I3C_EnableIT_INTUPD

Function name

__STATIC_INLINE void LL_I3C_EnableIT_INTUPD (I3C_TypeDef * I3Cx)

Function description

Enable Interrupt Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_EnableIT_INTUPD

LL_I3C_DisableIT_INTUPD

Function name

__STATIC_INLINE void LL_I3C_DisableIT_INTUPD (I3C_TypeDef * I3Cx)

Function description

Disable Interrupt Update interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_DisableIT_INTUPD

LL_I3C_IsEnabledIT_INTUPD

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_INTUPD (const I3C_TypeDef * I3Cx)

Function description

Check if Interrupt Update interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER INTUPDIE LL_I3C_IsEnabledIT_INTUPD

LL_I3C_EnableIT_DEF

Function name

__STATIC_INLINE void LL_I3C_EnableIT_DEF (I3C_TypeDef * I3Cx)

Function description

Enable Define List Target interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_EnableIT_DEF

LL_I3C_DisableIT_DEF

Function name

__STATIC_INLINE void LL_I3C_DisableIT_DEF (I3C_TypeDef * I3Cx)

Function description

Disable Define List Target interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_DisableIT_DEF

LL_I3C_IsEnabledIT_DEF

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_DEF (const I3C_TypeDef * I3Cx)

Function description

Check if Define List Target interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER DEFIE LL_I3C_IsEnabledIT_DEF

LL_I3C_EnableIT_GRP

Function name

__STATIC_INLINE void LL_I3C_EnableIT_GRP (I3C_TypeDef * I3Cx)

Function description

Enable Define List Group Addresses interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_EnableIT_GRP

LL_I3C_DisableIT_GRP

Function name

__STATIC_INLINE void LL_I3C_DisableIT_GRP (I3C_TypeDef * I3Cx)

Function description

Disable Define List Group Addresses interrupt.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_DisableIT_GRP

LL_I3C_IsEnabledIT_GRP

Function name

__STATIC_INLINE uint32_t LL_I3C_IsEnabledIT_GRP (const I3C_TypeDef * I3Cx)

Function description

Check if Define List Group Addresses interrupt is enabled or disabled.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER GRPIE LL_I3C_IsEnabledIT_GRP

LL_I3C_ClearFlag_FC

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_FC (I3C_TypeDef * I3Cx)

Function description

Clear Frame Complete flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CFCF LL_I3C_ClearFlag_FC

LL_I3C_ClearFlag_RXTGTEND

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_RXTGTEND (I3C_TypeDef * I3Cx)

Function description

Clear Reception Target End flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CRXTGTENDF LL_I3C_ClearFlag_RXTGTEND

LL_I3C_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_ERR (I3C_TypeDef * I3Cx)

Function description

Clear Error flag (controller and target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CERRF LL_I3C_ClearFlag_ERR

LL_I3C_ClearFlag_IBI

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_IBI (I3C_TypeDef * I3Cx)

Function description

Clear IBI flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CIBIF LL_I3C_ClearFlag_IBI

LL_I3C_ClearFlag_IBIEND

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_IBIEND (I3C_TypeDef * I3Cx)

Function description

Clear IBI End flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CIBIENDF LL_I3C_ClearFlag_IBIEND

LL_I3C_ClearFlag_CR

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_CR (I3C_TypeDef * I3Cx)

Function description

Clear Controller-role Request flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CCRF LL_I3C_ClearFlag_CR

LL_I3C_ClearFlag_CRUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_CRUPD (I3C_TypeDef * I3Cx)

Function description

Clear Controller-role Request Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CCRUPDF LL_I3C_ClearFlag_CRUPD

LL_I3C_ClearFlag_HJ

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_HJ (I3C_TypeDef * I3Cx)

Function description

Clear Hot Join flag (controller mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CHJF LL_I3C_ClearFlag_HJ

LL_I3C_ClearFlag_WKP

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_WKP (I3C_TypeDef * I3Cx)

Function description

Clear Wake Up flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CWKPF LL_I3C_ClearFlag_WKP

LL_I3C_ClearFlag_GET

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_GET (I3C_TypeDef * I3Cx)

Function description

Clear Get flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CGETF LL_I3C_ClearFlag_GET

LL_I3C_ClearFlag_STA

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_STA (I3C_TypeDef * I3Cx)

Function description

Clear Get Status flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CSTAF LL_I3C_ClearFlag_STA

LL_I3C_ClearFlag_DAUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_DAUPD (I3C_TypeDef * I3Cx)

Function description

Clear Dynamic Address Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CDAUPDF LL_I3C_ClearFlag_DAUPD

LL_I3C_ClearFlag_MWLUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_MWLUPD (I3C_TypeDef * I3Cx)

Function description

Clear Max Write Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CMWLUPDF LL_I3C_ClearFlag_MWLUPD

LL_I3C_ClearFlag_MRLUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_MRLUPD (I3C_TypeDef * I3Cx)

Function description

Clear Max Read Length flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CMRLUPDF LL_I3C_ClearFlag_MRLUPD

LL_I3C_ClearFlag_RST

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_RST (I3C_TypeDef * I3Cx)

Function description

Clear Reset flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CRSTF LL_I3C_ClearFlag_RST

LL_I3C_ClearFlag_ASUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_ASUPD (I3C_TypeDef * I3Cx)

Function description

Clear Active State flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CASUPDF LL_I3C_ClearFlag_ASUPD

LL_I3C_ClearFlag_INTUPD

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_INTUPD (I3C_TypeDef * I3Cx)

Function description

Clear Interrupt Update flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CINTUPDF LL_I3C_ClearFlag_INTUPD

LL_I3C_ClearFlag_DEF

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_DEF (I3C_TypeDef * I3Cx)

Function description

Clear Define List Targets flag (target mode).

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CDEFF LL_I3C_ClearFlag_DEF

LL_I3C_ClearFlag_GRP

Function name

__STATIC_INLINE void LL_I3C_ClearFlag_GRP (I3C_TypeDef * I3Cx)

Function description

Clear Define List Group Addresses flag.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CEVR CGRPF LL_I3C_ClearFlag_GRP

LL_I3C_Init

Function name

ErrorStatus LL_I3C_Init (I3C_TypeDef * I3Cx, LL_I3C_InitTypeDef * I3C_InitStruct, uint32_t Mode)

Function description

Initialize the I3C registers according to the specified parameters in I3C_InitStruct.

Parameters

- **I3Cx:** I3C Instance.
- **I3C_InitStruct:** pointer to a LL_I3C_InitTypeDef structure.
- **Mode:** I3C peripheral mode. This parameter can be a value of MODE.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I3C registers are initialized
  - ERROR: Not applicable

LL_I3C_DeInit

Function name

ErrorStatus LL_I3C_DeInit (const I3C_TypeDef * I3Cx)

Function description

De-initialize the I3C registers to their default reset values.

Parameters

- **I3Cx:** I3C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I3C registers are de-initialized
  - ERROR: I3C registers are not de-initialized

LL_I3C_StructInit

Function name

void LL_I3C_StructInit (LL_I3C_InitTypeDef * I3C_InitStruct)

Function description

Set each LL_I3C_InitTypeDef field to default value.

Parameters

- **I3C_InitStruct:** Pointer to a LL_I3C_InitTypeDef structure.

Return values

- **None:**

101.3 I3C Firmware driver defines

The following section lists the various define and macros of the module.

101.3.1 I3C

I3C

BUS ACTIVITY STATE

LL_I3C_BUS_ACTIVITY_STATE_0

Controller on the Bus Activity State 0

LL_I3C_BUS_ACTIVITY_STATE_1

Controller on the Bus Activity State 1

LL_I3C_BUS_ACTIVITY_STATE_2

Controller on the Bus Activity State 2

LL_I3C_BUS_ACTIVITY_STATE_3

Controller on the Bus Activity State 3

CONTROLLER ERROR

LL_I3C_CONTROLLER_ERROR_CE0

Controller detected an illegally formatted CCC

LL_I3C_CONTROLLER_ERROR_CE1

Controller detected that transmitted data on the bus is different than expected

LL_I3C_CONTROLLER_ERROR_CE2

Controller detected that broadcast address 7'h7E has been nacked

LL_I3C_CONTROLLER_ERROR_CE3

Controller detected that new Controller did not drive the bus after Controller-role handoff

CONTROLLER MTYPE

LL_I3C_CONTROLLER_MTYPE_RELEASE

SCL output clock stops running until next instruction executed

LL_I3C_CONTROLLER_MTYPE_HEADER

Header Message

LL_I3C_CONTROLLER_MTYPE_PRIVATE

Private Message Type

LL_I3C_CONTROLLER_MTYPE_DIRECT

Direct Message Type

LL_I3C_CONTROLLER_MTYPE_LEGACY_I2C

Legacy I2C Message Type

LL_I3C_CONTROLLER_MTYPE_CCC

Common Command Code

CR CAPABILITY

LL_I3C_CR_CAPABILITY

Controller acknowledge Target Controller Role capable

LL_I3C_CR_NO_CAPABILITY

Controller no acknowledge Target Controller Role capable

DEVICE ROLE AS

LL_I3C_DEVICE_ROLE_AS_TARGET

I3C Target

LL_I3C_DEVICE_ROLE_AS_CONTROLLER

I3C Controller

DIRECTION

LL_I3C_DIRECTION_WRITE

Write transfer

LL_I3C_DIRECTION_READ

Read transfer

DMA Register Data

LL_I3C_DMA_REG_DATA_TRANSMIT_BYTE

Get address of data register used for transmission in Byte

LL_I3C_DMA_REG_DATA_RECEIVE_BYTE

Get address of data register used for reception in Byte

LL_I3C_DMA_REG_DATA_TRANSMIT_WORD

Get address of data register used for transmission in Word

LL_I3C_DMA_REG_DATA_RECEIVE_WORD

Get address of data register used for reception in Word

LL_I3C_DMA_REG_STATUS

Get address of status register used for transfer status in Word

LL_I3C_DMA_REG_CONTROL

Get address of control register used for transfer control in Word

GENERATE

LL_I3C_GENERATE_STOP

Generate Stop condition after sending a message

LL_I3C_GENERATE_RESTART

Generate Restart condition after sending a message

GETMXDS FORMAT

LL_I3C_GETMXDS_FORMAT_1

GETMXDS CCC Format 1 is used, no MaxRdTurn field in response

LL_I3C_GETMXDS_FORMAT_2_LSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, LSB = RDTURN[7:0]

LL_I3C_GETMXDS_FORMAT_2_MID

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, Middle byte = RDTURN[7:0]

LL_I3C_GETMXDS_FORMAT_2_MSB

GETMXDS CCC Format 2 is used, MaxRdTurn field in response, MSB = RDTURN[7:0]

GETMXDS TSCO

LL_I3C_TURNAROUND_TIME_TSCO_LESS_12NS

clock-to-data turnaround time tSCO <= 12ns

LL_I3C_TURNAROUND_TIME_TSCO_GREATER_12NS

clock-to-data turnaround time tSCO > 12ns

Get Flags Defines

LL_I3C_EVR_CFEF

LL_I3C_EVR_TXFEF

LL_I3C_EVR_CFNFF

LL_I3C_EVR_SFNEF

LL_I3C_EVR_TXFNFF

LL_I3C_EVR_RXFNEF

LL_I3C_EVR_RXLASTF

LL_I3C_EVR_TXLASTF

LL_I3C_EVR_FCF

LL_I3C_EVR_RXTGTENDF

LL_I3C_EVR_ERRF

LL_I3C_EVR_IBIF

LL_I3C_EVR_IBIENDF

LL_I3C_EVR_CRF

LL_I3C_EVR_CRUPDF

LL_I3C_EVR_HJF

LL_I3C_EVR_WKPF

LL_I3C_EVR_GETF

LL_I3C_EVR_STAF

LL_I3C_EVR_DAUPDF

LL_I3C_EVR_MWLUPDF

LL_I3C_EVR_MRLUPDF

LL_I3C_EVR_RSTF

LL_I3C_EVR_ASUPDF

LL_I3C_EVR_INTUPDF

LL_I3C_EVR_DEFF

LL_I3C_EVR_GRPF

LL_I3C_SER_PERR

LL_I3C_SER_STALL

LL_I3C_SER_DOVR

LL_I3C_SER_COVR

LL_I3C_SER_ANACK

LL_I3C_SER_DNACK

LL_I3C_SER_DERR

HANDOFF

LL_I3C_HANDOFF_NOT_DELAYED

Additional time to process controllership handoff is not needed

LL_I3C_HANDOFF_DELAYED

Additional time to process controllership handoff is needed

HANDOFF ACTIVITY STATE

LL_I3C_HANDOFF_ACTIVITY_STATE_0

Indicates that will act according to Activity State 0 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_1

Indicates that will act according to Activity State 1 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_2

Indicates that will act according to Activity State 2 after controllership handoff

LL_I3C_HANDOFF_ACTIVITY_STATE_3

Indicates that will act according to Activity State 3 after controllership handoff

HANDOFF GRP ADDR NOT

LL_I3C_HANDOFF_GRP_ADDR_NOT_SUPPORTED

Group Address Handoff is not supported

LL_I3C_HANDOFF_GRP_ADDR_SUPPORTED

Group Address Handoff is supported

IBI ADDITIONAL DATA

LL_I3C_IBI_DATA_ENABLE

A mandatory data byte follows the IBI acknowledgement

LL_I3C_IBI_DATA_DISABLE

No mandatory data byte follows the IBI acknowledgement

IBI CAPABILITY

LL_I3C_IBI_CAPABILITY

Controller acknowledge Target In Band Interrupt capable

LL_I3C_IBI_NO_CAPABILITY

Controller no acknowledge Target In Band Interrupt capable

IBI MDB READ NOTIFICATION

LL_I3C_MDB_NO_PENDING_READ_NOTIFICATION

No support of pending read notification via the IBI MDB[7:0] value

LL_I3C_MDB_PENDING_READ_NOTIFICATION

Support of pending read notification via the IBI MDB[7:0] value

IBI NO ADDITIONAL

LL_I3C_IBI_NO_ADDITIONAL_DATA

No data byte follows the accepted IBI

LL_I3C_IBI_ADDITIONAL_DATA

A Mandatory Data Byte (MDB) follows the accepted IBI

IT Defines

LL_I3C_IER_CFNFIE

LL_I3C_IER_SFNEIE

LL_I3C_IER_TXFNFIE

LL_I3C_IER_RXFNEIE

LL_I3C_IER_FCIE

LL_I3C_IER_RXTGTENDIE

LL_I3C_IER_ERRIE

LL_I3C_IER_IBIIE

LL_I3C_IER_IBIENDIE

LL_I3C_IER_CRIE

LL_I3C_IER_CRUPDIE

LL_I3C_IER_HJIE

LL_I3C_IER_WKPIE

LL_I3C_IER_GETIE

LL_I3C_IER_STAIE

LL_I3C_IER_DAUPDIE

LL_I3C_IER_MWLUPDIE

LL_I3C_IER_MRLUPDIE

LL_I3C_IER_RSTIE

LL_I3C_IER_ASUPDIE

LL_I3C_IER_INTUPDIE

LL_I3C_IER_DEFIE

LL_I3C_IER_GRPIE

MAX DATA SPEED LIMITATION

LL_I3C_NO_DATA_SPEED_LIMITATION

No max data speed limitation

LL_I3C_MAX_DATA_SPEED_LIMITATION

Max data speed limitation

MESSAGE

LL_I3C_MESSAGE_ERROR

An error has been detected in the message

LL_I3C_MESSAGE_SUCCESS

The message ended with success

MESSAGE DIRECTION

LL_I3C_MESSAGE_DIRECTION_WRITE

Write data or command

LL_I3C_MESSAGE_DIRECTION_READ

Read data

MODE

LL_I3C_MODE_CONTROLLER

I3C Controller mode

LL_I3C_MODE_TARGET

I3C Target (Controller capable) mode

OWN ACTIVITY STATE

LL_I3C_OWN_ACTIVITY_STATE_0

Own Controller Activity state 0

LL_I3C_OWN_ACTIVITY_STATE_1

Own Controller Activity state 1

LL_I3C_OWN_ACTIVITY_STATE_2

Own Controller Activity state 2

LL_I3C_OWN_ACTIVITY_STATE_3

Own Controller Activity state 3

PAYLOAD

LL_I3C_PAYLOAD_EMPTY

Empty payload, no additional data after IBI acknowledge

LL_I3C_PAYLOAD_1_BYTE

One additional data byte after IBI acknowledge

LL_I3C_PAYLOAD_2_BYTES

Two additional data bytes after IBI acknowledge

LL_I3C_PAYLOAD_3_BYTES

Three additional data bytes after IBI acknowledge

LL_I3C_PAYLOAD_4_BYTES

Four additional data bytes after IBI acknowledge

RESET ACTION

LL_I3C_RESET_ACTION_NONE

No Reset Action Required

LL_I3C_RESET_ACTION_PARTIAL

Reset of some internal registers of the peripheral

LL_I3C_RESET_ACTION_FULL

Reset all internal registers of the peripheral

RX THRESHOLD

LL_I3C_RXFIFO_THRESHOLD_1_4

Rx Fifo Threshold is 1 byte in a Fifo depth of 4 bytes

LL_I3C_RXFIFO_THRESHOLD_4_4

Rx Fifo Threshold is 4 bytes in a Fifo depth of 4 bytes

SDA HOLD TIME 0

LL_I3C_SDA_HOLD_TIME_0_5

SDA hold time is 0.5 x ti3cclk

LL_I3C_SDA_HOLD_TIME_1_5

SDA hold time is 1.5 x ti3cclk

TARGET ERROR

LL_I3C_TARGET_ERROR_TE0

Target detected an invalid broadcast address

LL_I3C_TARGET_ERROR_TE1

Target detected an invalid CCC Code

LL_I3C_TARGET_ERROR_TE2

Target detected an invalid write data

LL_I3C_TARGET_ERROR_TE3

Target detected an invalid assigned address during Dynamic Address Assignment procedure

LL_I3C_TARGET_ERROR_TE4

Target detected 7'h7E missing after Restart during Dynamic Address Assignment procedure

LL_I3C_TARGET_ERROR_TE5

Target detected an illegally formatted CCC

LL_I3C_TARGET_ERROR_TE6

Target detected that transmitted data on the bus is different than expected

TARGET MTYPE HOT

LL_I3C_TARGET_MTYPE_HOT_JOIN

Hot Join

LL_I3C_TARGET_MTYPE_CONTROLLER_ROLE_REQ

Controller-role Request

LL_I3C_TARGET_MTYPE_IBI

In Band Interrupt (IBI)

TX THRESHOLD

LL_I3C_TXFIFO_THRESHOLD_1_4

Tx Fifo Threshold is 1 byte in a Fifo depth of 4 bytes

LL_I3C_TXFIFO_THRESHOLD_4_4

Tx Fifo Threshold is 4 bytes in a Fifo depth of 4 bytes

Common Write and read registers Macros

LL_I3C_GET_BCR

Description:

- Get Bus Characterics in payload (64bits) receive during ENTDAA procedure.

Parameters:

- __PAYLOAD__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00(uint64_t) and Max_Data=0xFFFFFFFFFFFFFFFF.

Return value:

- The: value of BCR Return value between Min_Data=0x00 and Max_Data=0xFF.

LL_I3C_GET_IBI_CAPABLE

Description:

- Check IBI request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_GET_IBI_PAYLOAD

Description:

- Check IBI additional data byte capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_GET_CR_CAPABLE

Description:

- Check Controller role request capabilities.

Parameters:

- __BCR__: specifies the Bus Characteristics capabilities retrieve during ENTDAA procedure. This parameter must be a number between Min_Data=0x00 and Max_Data=0xFF.

Return value:

- Value: of

LL_I3C_WriteReg

Description:

- Write a value in I3C register.

Parameters:

- __INSTANCE__: I3C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I3C_ReadReg

Description:

- Read a value in I3C register.

Parameters:

- __INSTANCE__: I3C Instance
- __REG__: Register to be read

Return value:

- Register: value

102 LL ICACHE Generic Driver

102.1 ICACHE Firmware driver registers structures

102.1.1 LL_ICACHE_RegionTypeDef

LL_ICACHE_RegionTypeDef is defined in the stm32h5xx_ll_icache.h

Data Fields

- ***uint32_t BaseAddress***
- ***uint32_t RemapAddress***
- ***uint32_t Size***
- ***uint32_t TrafficRoute***
- ***uint32_t OutputBurstType***

Field Documentation

- ***uint32_t LL_ICACHE_RegionTypeDef::BaseAddress***
  Configures the C-AHB base address to be remapped
- ***uint32_t LL_ICACHE_RegionTypeDef::RemapAddress***
  Configures the remap address to be remapped
- ***uint32_t LL_ICACHE_RegionTypeDef::Size***
  Configures the region size. This parameter can be a value of ***ICACHE_LL_EC_Region_Size***
- ***uint32_t LL_ICACHE_RegionTypeDef::TrafficRoute***
  Selects the traffic route. This parameter can be a value of ***ICACHE_LL_EC_Traffic_Route***
- ***uint32_t LL_ICACHE_RegionTypeDef::OutputBurstType***
  Selects the output burst type. This parameter can be a value of ***ICACHE_LL_EC_Output_Burst_Type***

102.2 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

102.2.1 Detailed description of functions

LL_ICACHE_Enable

Function name

__STATIC_INLINE void LL_ICACHE_Enable (void)

Function description

Enable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Enable

LL_ICACHE_Disable

Function name

__STATIC_INLINE void LL_ICACHE_Disable (void)

Function description

Disable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Disable

LL_ICACHE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabled (void)

Function description

Return if ICACHE is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_IsEnabled

LL_ICACHE_SetMode

Function name

__STATIC_INLINE void LL_ICACHE_SetMode (uint32_t Mode)

Function description

Select the ICACHE operating mode.

Parameters

- **Mode:** This parameter can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_2WAYS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_SetMode

LL_ICACHE_GetMode

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMode (void)

Function description

Get the selected ICACHE operating mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_2WAYS

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_GetMode

LL_ICACHE_Invalidate

Function name

__STATIC_INLINE void LL_ICACHE_Invalidate (void)

Function description

Invalidate the ICACHE.

Return values

- **None:**

Notes

- Until the BSYEND flag is set, the cache is bypassed.

Reference Manual to LL API cross reference:

- CR CACHEINV LL_ICACHE_Invalidate

LL_ICACHE_EnableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_EnableMonitors (uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_EnableMonitors
- CR MISSMEN LL_ICACHE_EnableMonitors

LL_ICACHE_DisableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_DisableMonitors (uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_DisableMonitors
- CR MISSMEN LL_ICACHE_DisableMonitors

LL_ICACHE_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledMonitors (uint32_t Monitors)

Function description

Check if the monitor(s) is(are) enabled or disabled.

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_IsEnabledMonitors
- CR MISSMEN LL_ICACHE_IsEnabledMonitors

LL_ICACHE_ResetMonitors

Function name

__STATIC_INLINE void LL_ICACHE_ResetMonitors (uint32_t Monitors)

Function description

Reset the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMRST LL_ICACHE_ResetMonitors
- CR MISSMRST LL_ICACHE_ResetMonitors

LL_ICACHE_GetHitMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetHitMonitor (void)

Function description

Get the Hit monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Upon reaching the 32-bit maximum value, hit monitor does not wrap.

Reference Manual to LL API cross reference:

- HMONR HITMON LL_ICACHE_GetHitMonitor

LL_ICACHE_GetMissMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMissMonitor (void)

Function description

Get the Miss monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Notes

- Upon reaching the 16-bit maximum value, miss monitor does not wrap.

Reference Manual to LL API cross reference:

- MMONR MISSMON LL_ICACHE_GetMissMonitor

LL_ICACHE_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_BSYEND (void)

Function description

Enable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_EnableIT_BSYEND

LL_ICACHE_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_BSYEND (void)

Function description

Disable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_DisableIT_BSYEND

LL_ICACHE_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_BSYEND (void)

Function description

Check if the BSYEND Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_IsEnabledIT_BSYEND

LL_ICACHE_EnableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_ERR (void)

Function description

Enable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_EnableIT_ERR

LL_ICACHE_DisableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_ERR (void)

Function description

Disable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_DisableIT_ERR

LL_ICACHE_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_ERR (void)

Function description

Check if the ERR Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_IsEnabledIT_ERR

LL_ICACHE_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BUSY (void)

Function description

Indicate the status of an ongoing operation flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_ICACHE_IsActiveFlag_BUSY

LL_ICACHE_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BSYEND (void)

Function description

Indicate the status of an operation end flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYEND LL_ICACHE_IsActiveFlag_BSYEND

LL_ICACHE_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_ERR (void)

Function description

Indicate the status of an error flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_ICACHE_IsActiveFlag_ERR

LL_ICACHE_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_BSYEND (void)

Function description

Clear busy end of operation flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_ICACHE_ClearFlag_BSYEND

LL_ICACHE_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_ERR (void)

Function description

Clear error flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR ERRF LL_ICACHE_ClearFlag_ERR

LL_ICACHE_EnableRegion

Function name

__STATIC_INLINE void LL_ICACHE_EnableRegion (uint32_t Region)

Function description

Enable the remapped memory region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **None:**

Notes

- The region must have been already configured.

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_EnableRegion

LL_ICACHE_DisableRegion

Function name

__STATIC_INLINE void LL_ICACHE_DisableRegion (uint32_t Region)

Function description

Disable the remapped memory region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_DisableRegion

LL_ICACHE_IsEnabledRegion

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledRegion (uint32_t Region)

Function description

Return if remapped memory region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_IsEnabledRegion

LL_ICACHE_SetRegionBaseAddress

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionBaseAddress (uint32_t Region, uint32_t Address)

Function description

Select the memory remapped region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Address:** Alias address in the Code region

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx BASEADDR LL_ICACHE_SetRegionBaseAddress

LL_ICACHE_GetRegionBaseAddress

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionBaseAddress (uint32_t Region)

Function description

Get the memory remapped region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Address:** Alias address in the Code region

Notes

- The base address is the alias in the Code region.

Reference Manual to LL API cross reference:

- CRRx BASEADDR LL_ICACHE_GetRegionBaseAddress

LL_ICACHE_SetRegionRemapAddress

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionRemapAddress (uint32_t Region, uint32_t Address)

Function description

Select the memory remapped region remap address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Address:** External memory address

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx REMAPADDR LL_ICACHE_SetRegionRemapAddress

LL_ICACHE_GetRegionRemapAddress

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionRemapAddress (uint32_t Region)

Function description

Get the memory remapped region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Address:** External memory address

Reference Manual to LL API cross reference:

- CRRx REMAPADDR LL_ICACHE_GetRegionRemapAddress

LL_ICACHE_SetRegionSize

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionSize (uint32_t Region, uint32_t Size)

Function description

Select the memory remapped region size.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Size:** This parameter can be one of the following values:
  - LL_ICACHE_REGIONSIZE_2MB
  - LL_ICACHE_REGIONSIZE_4MB
  - LL_ICACHE_REGIONSIZE_8MB
  - LL_ICACHE_REGIONSIZE_16MB
  - LL_ICACHE_REGIONSIZE_32MB
  - LL_ICACHE_REGIONSIZE_64MB
  - LL_ICACHE_REGIONSIZE_128MB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx RSIZE LL_ICACHE_SetRegionSize

LL_ICACHE_GetRegionSize

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionSize (uint32_t Region)

Function description

Get the selected the memory remapped region size.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_REGIONSIZE_2MB
  - LL_ICACHE_REGIONSIZE_4MB
  - LL_ICACHE_REGIONSIZE_8MB
  - LL_ICACHE_REGIONSIZE_16MB
  - LL_ICACHE_REGIONSIZE_32MB
  - LL_ICACHE_REGIONSIZE_64MB
  - LL_ICACHE_REGIONSIZE_128MB

Reference Manual to LL API cross reference:

- CRRx RSIZE LL_ICACHE_GetRegionSize

LL_ICACHE_SetRegionOutputBurstType

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionOutputBurstType (uint32_t Region, uint32_t Type)

Function description

Select the memory remapped region output burst type.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Type:** This parameter can be one of the following values:
  - LL_ICACHE_OUTPUT_BURST_WRAP
  - LL_ICACHE_OUTPUT_BURST_INCR

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx HBURST LL_ICACHE_SetRegionOutputBurstType

LL_ICACHE_GetRegionOutputBurstType

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionOutputBurstType (uint32_t Region)

Function description

Get the selected the memory remapped region output burst type.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_OUTPUT_BURST_WRAP
  - LL_ICACHE_OUTPUT_BURST_INCR

Reference Manual to LL API cross reference:

- CRRx HBURST LL_ICACHE_GetRegionOutputBurstType

LL_ICACHE_SetRegionMasterPort

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionMasterPort (uint32_t Region, uint32_t Port)

Function description

Select the memory remapped region cache master port.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Port:** This parameter can be one of the following values:
  - LL_ICACHE_MASTER1_PORT
  - LL_ICACHE_MASTER2_PORT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx MSTSEL LL_ICACHE_SetRegionMasterPort

LL_ICACHE_GetRegionMasterPort

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionMasterPort (uint32_t Region)

Function description

Get the selected the memory remapped region cache master port.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_MASTER1_PORT
  - LL_ICACHE_MASTER2_PORT

Reference Manual to LL API cross reference:

- CRRx MSTSEL LL_ICACHE_GetRegionMasterPort

LL_ICACHE_ConfigRegion

Function name

void LL_ICACHE_ConfigRegion (uint32_t Region, const LL_ICACHE_RegionTypeDef *const pICACHE_RegionStruct)

Function description

Configure and enable the memory remapped region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **pICACHE_RegionStruct:** pointer to a LL_ICACHE_RegionTypeDef structure.

Return values

- **None:**

Notes

- The Instruction Cache and corresponding region must be disabled.

102.3 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

102.3.1 ICACHE

ICACHE

Clear Flags Defines

LL_ICACHE_FCR_CBSYENDF

Busy end flag

LL_ICACHE_FCR_CERRF

Cache error flag

Get Flags Defines

LL_ICACHE_SR_BUSYF

Busy flag

LL_ICACHE_SR_BSYENDF

Busy end flag

LL_ICACHE_SR_ERRF

Cache error flag

IT Defines

LL_ICACHE_IER_BSYENDIE

Busy end interrupt

LL_ICACHE_IER_ERRIE

Cache error interrupt

Monitor type

LL_ICACHE_MONITOR_HIT

Hit monitor counter

LL_ICACHE_MONITOR_MISS

Miss monitor counter

LL_ICACHE_MONITOR_ALL

All monitors counters

Remapped Output burst type

LL_ICACHE_OUTPUT_BURST_WRAP

WRAP

LL_ICACHE_OUTPUT_BURST_INCR

INCR

Remapped Region number

LL_ICACHE_REGION_0

Region 0

LL_ICACHE_REGION_1

Region 1

LL_ICACHE_REGION_2

Region 2

LL_ICACHE_REGION_3

Region 3

Remapped Region size

LL_ICACHE_REGIONSIZE_2MB

Region size 2MB

LL_ICACHE_REGIONSIZE_4MB

Region size 4MB

LL_ICACHE_REGIONSIZE_8MB

Region size 8MB

LL_ICACHE_REGIONSIZE_16MB

Region size 16MB

LL_ICACHE_REGIONSIZE_32MB

Region size 32MB

LL_ICACHE_REGIONSIZE_64MB

Region size 64MB

LL_ICACHE_REGIONSIZE_128MB

Region size 128MB

Remapped Traffic route

LL_ICACHE_MASTER1_PORT

Master1 port

LL_ICACHE_MASTER2_PORT

Master2 port

Ways selection

LL_ICACHE_1WAY

1-way cache (direct mapped cache)

LL_ICACHE_2WAYS

2-ways set associative cache (default)

Common write and read registers Macros

LL_ICACHE_WriteReg

Description:

- Write a value in ICACHE register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ICACHE_ReadReg

Description:

- Read a value in ICACHE register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

103 LL IWDG Generic Driver

103.1 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

103.1.1 Detailed description of functions

LL_IWDG_Enable

Function name

__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)

Function description

Start the Independent Watchdog.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Notes

- Except if the hardware watchdog option is selected

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Reloads IWDG counter with value defined in the reload register.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)

Function description

Select the prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256
  - LL_IWDG_PRESCALER_512
  - LL_IWDG_PRESCALER_1024

Return values

- **None:**

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (const IWDG_TypeDef * IWDGx)

Function description

Get the selected prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256
  - LL_IWDG_PRESCALER_512
  - LL_IWDG_PRESCALER_1024

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)

Function description

Specify the IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance
- **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (const IWDG_TypeDef * IWDGx)

Function description

Get the specified IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name

__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)

Function description

Specify high limit of the window value to be compared to the down-counter.

Parameters

- **IWDGx:** IWDG Instance
- **Window:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetWindow (const IWDG_TypeDef * IWDGx)

Function description

Get the high limit of the window value specified.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_GetWindow

LL_IWDG_SetEwiTime

Function name

__STATIC_INLINE void LL_IWDG_SetEwiTime (IWDG_TypeDef * IWDGx, uint32_t Time)

Function description

Specify comparator value that will be used to trig Early Wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance
- **Time:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_SetEwiTime

LL_IWDG_GetEwiTime

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetEwiTime (const IWDG_TypeDef * IWDGx)

Function description

Get the Early Wakeup interrupt comparator value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_GetEwiTime

LL_IWDG_EnableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_EnableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Enable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_EnableIT_EWI

LL_IWDG_DisableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_DisableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Disable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_DisableIT_EWI

LL_IWDG_IsEnabledIT_EWI

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsEnabledIT_EWI (const IWDG_TypeDef * IWDGx)

Function description

Indicates whether Early wakeup interrupt is enable.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_IsEnabledIT_EWI

LL_IWDG_IsActiveFlag_PVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Prescaler Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Reload Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Window Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsActiveFlag_EWU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag EWI Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EVU LL_IWDG_IsActiveFlag_EWU

LL_IWDG_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (const IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload, Window & Early Interrupt Value Update are reset or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bits (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady
- SR WVU LL_IWDG_IsReady
- SR EWU LL_IWDG_IsReady

LL_IWDG_IsActiveFlag_ONF

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_ONF (const IWDG_TypeDef * IWDGx)

Function description

Check if IWDG has been started or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ONF LL_IWDG_IsActiveFlag_ONF

LL_IWDG_IsActiveFlag_EWIF

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWIF (const IWDG_TypeDef * IWDGx)

Function description

Check if Early Wakeup interrupt flag is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EWIF LL_IWDG_IsActiveFlag_EWIF

LL_IWDG_ClearFlag_EWIF

Function name

__STATIC_INLINE void LL_IWDG_ClearFlag_EWIF (IWDG_TypeDef * IWDGx)

Function description

Clear the Early Wakeup interrupt flag.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIC LL_IWDG_ClearFlag_EWIF

103.2 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

103.2.1 IWDG

IWDG

Get Flags Defines

LL_IWDG_SR_PVU

Watchdog prescaler value update

LL_IWDG_SR_RVU

Watchdog counter reload value update

LL_IWDG_SR_WVU

Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4

Divider by 4

LL_IWDG_PRESCALER_8

Divider by 8

LL_IWDG_PRESCALER_16

Divider by 16

LL_IWDG_PRESCALER_32

Divider by 32

LL_IWDG_PRESCALER_64

Divider by 64

LL_IWDG_PRESCALER_128

Divider by 128

LL_IWDG_PRESCALER_256

Divider by 256

LL_IWDG_PRESCALER_512

Divider by 512

LL_IWDG_PRESCALER_1024

Divider by 1024

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

104 LL LPTIM Generic Driver

104.1 LPTIM Firmware driver registers structures

104.1.1 LL_LPTIM_InitTypeDef

LL_LPTIM_InitTypeDef is defined in the stm32h5xx_ll_lptim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***
  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***
  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***
  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetWaveform()**.

104.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

104.2.1 Detailed description of functions

LL_LPTIM_DeInit

Function name

ErrorStatus LL_LPTIM_DeInit (const LPTIM_TypeDef * LPTIMx)

Function description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx registers are de-initialized
  - ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name

void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Set each fields of the LPTIM_InitStruct structure to its default value.

Parameters

- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **None:**

LL_LPTIM_Init

Function name

ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, const LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Configure the LPTIMx peripheral according to the specified parameters.

Parameters

- **LPTIMx:** LP Timer Instance
- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx instance has been initialized
  - ERROR: LPTIMx instance hasn't been initialized

Notes

- LL_LPTIM_Init can only be called when the LPTIM instance is disabled.
- LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

LL_LPTIM_Enable

Function name

__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)

Function description

Enable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Enable

LL_LPTIM_Disable

Function name

__STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)

Function description

Disable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Disable

LL_LPTIM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM instance is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name

__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)

Function description

Starts the LPTIM counter in the desired mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **OperatingMode:** This parameter can be one of the following values:
  - LL_LPTIM_OPERATING_MODE_CONTINUOUS
  - LL_LPTIM_OPERATING_MODE_ONESHOT

Return values

- **None:**

Notes

- LPTIM instance must be enabled before starting the counter.
- It is possible to change on the fly from One Shot mode to Continuous mode.

Reference Manual to LL API cross reference:

- CR CNTSTRT LL_LPTIM_StartCounter
- CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_EnableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_EnableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Enable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After calling this function any read access to LPTIM_CNT register will asynchronously reset the LPTIM_CNT register content.

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_EnableResetAfterRead

LL_LPTIM_DisableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_DisableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Disable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_DisableResetAfterRead

LL_LPTIM_IsEnabledResetAfterRead

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledResetAfterRead (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the reset after read feature is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_IsEnabledResetAfterRead

LL_LPTIM_ResetCounter

Function name

__STATIC_INLINE void LL_LPTIM_ResetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Reset of the LPTIM_CNT counter register (synchronous).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- Due to the synchronous nature of this reset, it only takes place after a synchronization delay of 3 LPTIM core clock cycles (LPTIM core clock may be different from APB clock).
- COUNTRST is automatically cleared by hardware

Reference Manual to LL API cross reference:

- CR COUNTRST LL_LPTIM_ResetCounter
-

LL_LPTIM_SetUpdateMode

Function name

__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)

Function description

Set the LPTIM registers update mode (enable/disable register preload)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **UpdateMode:** This parameter can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the LPTIM registers update mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name

__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)

Function description

Set the auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- The LPTIMx_ARR register content must only be modified when the LPTIM is enabled
- After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag is set, will lead to unpredictable results.
- autoreload value be strictly greater than the compare value.

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetRepetition

Function name

__STATIC_INLINE void LL_LPTIM_SetRepetition (LPTIM_TypeDef * LPTIMx, uint32_t Repetition)

Function description

Set the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- The LPTIMx_RCR register content must only be modified when the LPTIM is enabled

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_SetRepetition

LL_LPTIM_GetRepetition

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetRepetition (const LPTIM_TypeDef * LPTIMx)

Function description

Get the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_GetRepetition

LL_LPTIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_EnableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Enable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_EnableChannel
- CCMR1 CC2E LL_LPTIM_CC_EnableChannel

LL_LPTIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_DisableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Disable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_DisableChannel
- CCMR1 CC2E LL_LPTIM_CC_DisableChannel

LL_LPTIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_IsEnabledChannel (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Indicate whether channel is enabled.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_IsEnabledChannel
- CCMR1 CC2E LL_LPTIM_CC_IsEnabledChannel

LL_LPTIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH1 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR1 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP1OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_SetCompareCH1

LL_LPTIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_GetCompareCH1

LL_LPTIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH2 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR2 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP2OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_SetCompareCH2

LL_LPTIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_GetCompareCH2

LL_LPTIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual counter value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Counter:** value

Notes

- When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

Reference Manual to LL API cross reference:

- CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name

__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)

Function description

Set the counter mode (selection of the LPTIM counter clock source).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Return values

- **None:**

Notes

- The counter mode can be set only when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the counter mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_SetWaveform

Function name

__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)

Function description

Set waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_SetPolarity
-

LL_LPTIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_GetPolarity
-

LL_LPTIM_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)

Function description

Set actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT1_SRC_GPIO (*) Value not defined for all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN1SEL LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 2 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT2_SRC_GPIO

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN2SEL LL_LPTIM_SetInput2Src

LL_LPTIM_SetRemap

Function name

__STATIC_INLINE void LL_LPTIM_SetRemap (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_LPTIM1_IC1_RMP_GPIO
  - LL_LPTIM_LPTIM1_IC2_RMP_GPIO
  - LL_LPTIM_LPTIM2_IC1_RMP_GPIO
  - LL_LPTIM_LPTIM2_IC2_RMP_GPIO
  - LL_LPTIM_LPTIM2_IC2_RMP_HSI_1024
  - LL_LPTIM_LPTIM2_IC2_RMP_CSI_128
  - LL_LPTIM_LPTIM2_IC2_RMP_HSI_8
  - LL_LPTIM_LPTIM3_IC1_RMP_GPIO (*)
  - LL_LPTIM_LPTIM3_IC2_RMP_GPIO (*)
  - LL_LPTIM_LPTIM5_IC1_RMP_GPIO (*)
  - LL_LPTIM_LPTIM5_IC2_RMP_GPIO (*)
  - LL_LPTIM_LPTIM6_IC1_RMP_GPIO (*)
  - LL_LPTIM_LPTIM6_IC2_RMP_GPIO (*)

  (*) Value not defined in all devices.
-

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IC1SEL LL_LPTIM_SetRemap
- CFGR2 IC2SEL LL_LPTIM_SetRemap

LL_LPTIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of IC channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_ICPOLARITY_RISING
  - LL_LPTIM_ICPOLARITY_FALLING
  - LL_LPTIM_ICPOLARITY_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_SetPolarity
-

LL_LPTIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICPOLARITY_RISING
  - LL_LPTIM_ICPOLARITY_FALLING
  - LL_LPTIM_ICPOLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_GetPolarity
-

LL_LPTIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetFilter (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Filter)

Function description

Set the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_SetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_SetFilter
-

LL_LPTIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetFilter (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_GetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_GetFilter
-

LL_LPTIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Prescaler)

Function description

Set the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_ICPSC_DIV1
  - LL_LPTIM_ICPSC_DIV2
  - LL_LPTIM_ICPSC_DIV4
  - LL_LPTIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_SetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_SetPrescaler
-

LL_LPTIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPrescaler (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ICPSC_DIV1
  - LL_LPTIM_ICPSC_DIV2
  - LL_LPTIM_ICPSC_DIV4
  - LL_LPTIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_GetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_GetPrescaler
-

LL_LPTIM_CC_SetChannelMode

Function name

__STATIC_INLINE void LL_LPTIM_CC_SetChannelMode (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t CCMode)

Function description

Set the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **CCMode:** This parameter can be one of the following values:
  - LL_LPTIM_CCMODE_OUTPUT_PWM
  - LL_LPTIM_CCMODE_INPUTCAPTURE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_SetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_SetChannelMode

LL_LPTIM_CC_GetChannelMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_GetChannelMode (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CCMODE_OUTPUT_PWM
  - LL_LPTIM_CCMODE_INPUTCAPTURE

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_GetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_GetChannelMode

LL_LPTIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_IC_GetCaptureCH1

LL_LPTIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_IC_GetCaptureCH2

LL_LPTIM_EnableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Enable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.
- The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Disable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- A trigger event arriving when the timer is already started will be ignored.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the timeout function is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name

__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)

Function description

Start the LPTIM counter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name

__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)

Function description

Configure the external trigger used as a trigger event for the LPTIM.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Source:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2

  (*) Value not defined in all devices.
-
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2

  (*) Value not defined in all devices.
-

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:

- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name

__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)

Function description

Set the source of the clock used by the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual LPTIM instance clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name

__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)

Function description

Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockFilter:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.
- An internal clock source must be present when a digital filter is required for external clock.

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_ConfigClock
- CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock digital filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)

Function description

Configure the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Enable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.
- LPTIM instance must be configured in continuous mode prior enabling the encoder mode.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Disable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM operates in encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 1 (CC1CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1CF LL_LPTIM_ClearFlag_CC1

LL_LPTIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1IF LL_LPTIM_IsActiveFlag_CC1

LL_LPTIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 2 (CC2CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2CF LL_LPTIM_ClearFlag_CC2

LL_LPTIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2IF LL_LPTIM_IsActiveFlag_CC2

LL_LPTIM_ClearFlag_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 1 over-capture flag for channel 1 (CC1OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1OCF LL_LPTIM_ClearFlag_CC1O

LL_LPTIM_IsActiveFlag_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 1 over-capture has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1OF LL_LPTIM_IsActiveFlag_CC1O

LL_LPTIM_ClearFlag_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 2 over-capture flag for channel 2 (CC2OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2OCF LL_LPTIM_ClearFlag_CC2O

LL_LPTIM_IsActiveFlag_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 2 over-capture has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2OF LL_LPTIM_IsActiveFlag_CC2O

LL_LPTIM_ClearFlag_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload match flag (ARRMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARRMCF LL_LPTIM_ClearFlag_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a autoreload match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Clear the external trigger valid edge flag(EXTTRIGCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP1OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP1OKCF LL_LPTIM_ClearFlag_CMP1OK

LL_LPTIM_IsActiveFlag_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR1 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP1OK LL_LPTIM_IsActiveFlag_CMP1OK

LL_LPTIM_ClearFlag_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP2OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP2OKCF LL_LPTIM_ClearFlag_CMP2OK

LL_LPTIM_IsActiveFlag_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR2 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP2OK LL_LPTIM_IsActiveFlag_CMP2OK

LL_LPTIM_ClearFlag_DIEROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DIEROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the interrupt register update interrupt flag (DIEROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DIEROKCF LL_LPTIM_ClearFlag_DIEROK

LL_LPTIM_IsActiveFlag_DIEROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DIEROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_DIER register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DIEROK LL_LPTIM_IsActiveFlag_DIEROK

LL_LPTIM_ClearFlag_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload register update interrupt flag (ARROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to up interrupt flag (UPCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_ClearFlag_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the repetition register update interrupt flag (REPOKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR REPOKCF LL_LPTIM_ClearFlag_REPOK

LL_LPTIM_IsActiveFlag_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_RCR register has been successfully completed; If so, a new one can be initiated.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REPOK LL_LPTIM_IsActiveFlag_REPOK

LL_LPTIM_ClearFlag_UE

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UE (LPTIM_TypeDef * LPTIMx)

Function description

Clear the update event flag (UECF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UECF LL_LPTIM_ClearFlag_UE

LL_LPTIM_IsActiveFlag_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the LPTIMx update event has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UE LL_LPTIM_IsActiveFlag_UE

LL_LPTIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_EnableIT_CC1

LL_LPTIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_DisableIT_CC1

LL_LPTIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_IsEnabledIT_CC1

LL_LPTIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_EnableIT_CC2

LL_LPTIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_DisableIT_CC2

LL_LPTIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_IsEnabledIT_CC2

LL_LPTIM_EnableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_EnableIT_CC1O

LL_LPTIM_DisableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_DisableIT_CC1O

LL_LPTIM_IsEnabledIT_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 over-capture interrupt (CC1OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_IsEnabledIT_CC1O

LL_LPTIM_EnableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_EnableIT_CC2O

LL_LPTIM_DisableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_DisableIT_CC2O

LL_LPTIM_IsEnabledIT_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 over-capture interrupt (CC2OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_IsEnabledIT_CC2O

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_EnableIT_CMP1OK

LL_LPTIM_DisableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPO1KIE LL_LPTIM_DisableIT_CMP1OK

LL_LPTIM_IsEnabledIT_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP1OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_IsEnabledIT_CMP1OK

LL_LPTIM_EnableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_EnableIT_CMP2OK

LL_LPTIM_DisableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_DisableIT_CMP2OK

LL_LPTIM_IsEnabledIT_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP2OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_IsEnabledIT_CMP2OK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

LL_LPTIM_EnableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_EnableIT_REPOK

LL_LPTIM_DisableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_DisableIT_REPOK

LL_LPTIM_IsEnabledIT_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the repetition register update successfully completed interrupt (REPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_IsEnabledIT_REPOK

LL_LPTIM_EnableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_EnableIT_UE

LL_LPTIM_DisableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_DisableIT_UE

LL_LPTIM_IsEnabledIT_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update event interrupt (UEIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance @ retval State of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_IsEnabledIT_UE

LL_LPTIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_EnableDMAReq_UPDATE

LL_LPTIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_DisableDMAReq_UPDATE

LL_LPTIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_UPDATE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update DMA request is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_IsEnabledDMAReq_UPDATE

LL_LPTIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_EnableDMAReq_CC1

LL_LPTIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_DisableDMAReq_CC1

LL_LPTIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_IsEnabledDMAReq_CC1

LL_LPTIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_EnableDMAReq_CC2

LL_LPTIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_DisableDMAReq_CC2

LL_LPTIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_IsEnabledDMAReq_CC2

104.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

104.3.1 LPTIM

LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4, LPTIM5 and LPTIM6

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO

For LPTIM1, LPTIM2, LPTIM3, LPTIM4 , LPTIM5 and LPTIM6

LPTIM1 Input Ch1 Remap

LL_LPTIM_LPTIM1_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM1 Input Ch2 Remap

LL_LPTIM_LPTIM1_IC2_RMP_GPIO

IC2 connected to GPIO

LL_LPTIM_LPTIM1_IC2_RMP_LSI

IC2 connected to LSI

LL_LPTIM_LPTIM1_IC2_RMP_LSE

IC2 connected to LSE

LPTIM2 Input Ch1 Remap

LL_LPTIM_LPTIM2_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM2 Input Ch2 Remap

LL_LPTIM_LPTIM2_IC2_RMP_GPIO

IC2 connected to GPIO

LL_LPTIM_LPTIM2_IC2_RMP_HSI_1024

IC2 connected to HSI/1024

LL_LPTIM_LPTIM2_IC2_RMP_CSI_128

IC2 connected to CSI/128

LL_LPTIM_LPTIM2_IC2_RMP_HSI_8

IC2 connected to HSI/8

LPTIM3 Input Ch1 Remap

LL_LPTIM_LPTIM3_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM3 Input Ch2 Remap

LL_LPTIM_LPTIM3_IC2_RMP_GPIO

IC2 connected to GPIO

LPTIM5 Input Ch1 Remap

LL_LPTIM_LPTIM5_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM5 Input Ch2 Remap

LL_LPTIM_LPTIM5_IC2_RMP_GPIO

IC2 connected to GPIO

LPTIM6 Input Ch1 Remap

LL_LPTIM_LPTIM6_IC1_RMP_GPIO

IC1 connected to GPIO

LPTIM6 Input Ch2 Remap

LL_LPTIM_LPTIM6_IC2_RMP_GPIO

IC2 connected to GPIO

Clock Filter

LL_LPTIM_CLK_FILTER_NONE

Any external clock signal level change is considered as a valid transition

LL_LPTIM_CLK_FILTER_2

External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_4

External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_8

External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING

The rising edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_RISING_FALLING

Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL

LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

LL_LPTIM_CLK_SOURCE_EXTERNAL

LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL

The counter is incremented following each internal clock pulse

LL_LPTIM_COUNTER_MODE_EXTERNAL

The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING

The rising edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_RISING_FALLING

Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMP1OK

Compare register 1 update OK

LL_LPTIM_ISR_CMP2OK

Compare register 2 update OK

LL_LPTIM_ISR_CC1IF

Capture/Compare 1 interrupt flag

LL_LPTIM_ISR_CC2IF

Capture/Compare 2 interrupt flag

LL_LPTIM_ISR_CC1OF

Capture/Compare 1 over-capture flag

LL_LPTIM_ISR_CC2OF

Capture/Compare 2 over-capture flag

LL_LPTIM_ISR_DIEROK

Interrupt enable register update OK

LL_LPTIM_ISR_ARRM

Autoreload match

LL_LPTIM_ISR_EXTTRIG

External trigger edge event

LL_LPTIM_ISR_ARROK

Autoreload register update OK

LL_LPTIM_ISR_UP

Counter direction change down to up

LL_LPTIM_ISR_DOWN

Counter direction change up to down

LL_LPTIM_ISR_UE

Update event

LL_LPTIM_ISR_REPOK

Repetition register update OK

IT Defines

LL_LPTIM_DIER_CMP1OKIE

Compare register 1 update OK

LL_LPTIM_DIER_CMP2OKIE

Compare register 2 update OK

LL_LPTIM_DIER_CC1IFIE

Capture/Compare 1 interrupt flag

LL_LPTIM_DIER_CC2IFIE

Capture/Compare 2 interrupt flag

LL_LPTIM_DIER_CC1OFIE

Capture/Compare 1 over-capture flag

LL_LPTIM_DIER_CC2OFIE

Capture/Compare 2 over-capture flag

LL_LPTIM_DIER_ARRMIE

Autoreload match

LL_LPTIM_DIER_EXTTRIGIE

External trigger edge event

LL_LPTIM_DIER_ARROKIE

Autoreload register update OK

LL_LPTIM_DIER_UPIE

Counter direction change down to up

LL_LPTIM_DIER_DOWNIE

Counter direction change up to down

LL_LPTIM_DIER_UEIE

Update event

LL_LPTIM_DIER_REPOKIE

Repetition register update OK

Input Capture Filter

LL_LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LL_LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Input Capture Polarity

LL_LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LL_LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LL_LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

Input Capture Prescaler

LL_LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LL_LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LL_LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LL_LPTIM_ICPSC_DIV8

Capture performed once every 8 events

Input Capture selection

LL_LPTIM_CCMODE_OUTPUT_PWM

Select PWM mode

LL_LPTIM_CCMODE_INPUTCAPTURE

Select Input Capture mode

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS

LP Timer starts in continuous mode

LL_LPTIM_OPERATING_MODE_ONESHOT

LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CCRx registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CCx registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINUOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_RTCTAMP2

External input trigger is connected to RTC Tamper 2

LL_LPTIM_TRIG_SOURCE_GPDMA_CH0_TCF

External input trigger is connected to GPDMA CH0 transfer complete

LL_LPTIM_TRIG_SOURCE_GPDMA_CH1_TCF

External input trigger is connected to GPDMA CH1 transfer complete

LL_LPTIM_TRIG_SOURCE_GPDMA_CH4_TCF

External input trigger is connected to GPDMA CH4 transfer complete

LL_LPTIM_TRIG_SOURCE_EVENTOUT

External input trigger is connected to EVENTOUT

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE

Preload is disabled: registers are updated after each APB bus write access

LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

LL_LPTIM_IC_GET_OFFSET

Description:

- LPTimer Input Capture Get Offset(in counter step unit)

Parameters:

- __PSC__: This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128
- __FLT__: This parameter can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Return value:

- offset: value

Notes:

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset The Offset value is depending on the glitch filter value for the channel and the value of the prescaler for the kernel clock. Please check Errata Sheet V1_8 for more details under "variable latency on input capture channel" section.

105 LL LPUART Generic Driver

105.1 LPUART Firmware driver registers structures

105.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32h5xx_ll_lpuart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

105.2 LPUART Firmware driver API description

The following section lists the various functions of the LPUART library.

105.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (const USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (const USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (const USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (const USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (const USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (const USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (const USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (const USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (const USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (const USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (const USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (const USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetWKUPType

Function name

__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Type:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (const USART_TypeDef * LPUARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value ! = 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (const USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (const USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (const USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from stop mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (const USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_LPUART_DMA_REG_DATA_TRANSMIT
  - LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (const USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, const LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

105.3 LPUART Firmware driver defines

The following section lists the various define and macros of the module.

105.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error clear flag

LL_LPUART_ICR_FECF

Framing error clear flag

LL_LPUART_ICR_NCF

Noise error detected clear flag

LL_LPUART_ICR_ORECF

Overrun error clear flag

LL_LPUART_ICR_IDLECF

Idle line detected clear flag

LL_LPUART_ICR_TCCF

Transmission complete clear flag

LL_LPUART_ICR_CTSCF

CTS clear flag

LL_LPUART_ICR_CMCF

Character match clear flag

LL_LPUART_ICR_WUCF

Wakeup from Stop mode clear flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_WUF

Wakeup from Stop mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not divided

LL_LPUART_PRESCALER_DIV2

Input clock divided by 2

LL_LPUART_PRESCALER_DIV4

Input clock divided by 4

LL_LPUART_PRESCALER_DIV6

Input clock divided by 6

LL_LPUART_PRESCALER_DIV8

Input clock divided by 8

LL_LPUART_PRESCALER_DIV10

Input clock divided by 10

LL_LPUART_PRESCALER_DIV12

Input clock divided by 12

LL_LPUART_PRESCALER_DIV16

Input clock divided by 16

LL_LPUART_PRESCALER_DIV32

Input clock divided by 32

LL_LPUART_PRESCALER_DIV64

Input clock divided by 64

LL_LPUART_PRESCALER_DIV128

Input clock divided by 128

LL_LPUART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_LPUART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_LPUART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

106 LL OPAMP Generic Driver

106.1 OPAMP Firmware driver registers structures

106.1.1 LL_OPAMP_InitTypeDef

LL_OPAMP_InitTypeDef is defined in the stm32h5xx_ll_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t FunctionalMode***
- ***uint32_t InputNonInverting***
- ***uint32_t InputInverting***

Field Documentation

- ***uint32_t LL_OPAMP_InitTypeDef::PowerMode***
  Set OPAMP power mode. This parameter can be a value of ***OPAMP_LL_EC_POWER_MODE*** This feature can be modified afterwards using unitary function **LL_OPAMP_SetPowerMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::FunctionalMode***
  Set OPAMP functional mode by setting internal connections: OPAMP operation in standalone, follower, ... This parameter can be a value of ***OPAMP_LL_EC_FUNCTIONAL_MODE***
  Note:
  – If OPAMP is configured in mode PGA, the gain can be configured using function **LL_OPAMP_SetPGAGain()**. This feature can be modified afterwards using unitary function **LL_OPAMP_SetFunctionalMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputNonInverting***
  Set OPAMP input non-inverting connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_NONINVERTING*** This feature can be modified afterwards using unitar function **LL_OPAMP_SetInputNonInverting()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputInverting***
  Set OPAMP inverting input connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_INVERTING***
  Note:
  – OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin), this parameter is discarded. This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputInverting()**.

106.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

106.2.1 Detailed description of functions

LL_OPAMP_SetMode

Function name

__STATIC_INLINE void LL_OPAMP_SetMode (OPAMP_TypeDef * OPAMPx, uint32_t Mode)

Function description

Set OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance
- **Mode:** This parameter can be one of the following values:
  – LL_OPAMP_MODE_FUNCTIONAL
  – LL_OPAMP_MODE_CALIBRATION

Return values

- **None:**

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_SetMode

LL_OPAMP_GetMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_GetMode

LL_OPAMP_SetFunctionalMode

Function name

__STATIC_INLINE void LL_OPAMP_SetFunctionalMode (OPAMP_TypeDef * OPAMPx, uint32_t FunctionalMode)

Function description

Set OPAMP functional mode by setting internal connections.

Parameters

- **OPAMPx:** OPAMP instance
- **FunctionalMode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Return values

- **None:**

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetFunctionalMode

LL_OPAMP_GetFunctionalMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetFunctionalMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP functional mode from setting of internal connections.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetFunctionalMode

LL_OPAMP_SetPGAGain

Function name

__STATIC_INLINE void LL_OPAMP_SetPGAGain (OPAMP_TypeDef * OPAMPx, uint32_t PGAGain)

Function description

Set OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance
- **PGAGain:** This parameter can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_SetPGAGain

LL_OPAMP_GetPGAGain

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPGAGain (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_GetPGAGain

LL_OPAMP_SetPowerMode

Function name

__STATIC_INLINE void LL_OPAMP_SetPowerMode (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode)

Function description

Set OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Return values

- **None:**

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR OPAHSM LL_OPAMP_SetPowerMode

LL_OPAMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPowerMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR OPAHSM LL_OPAMP_GetPowerMode

LL_OPAMP_SetInputNonInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputNonInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)

Function description

Set OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputNonInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_IO1
  - LL_OPAMP_INPUT_NONINVERT_DAC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_SetInputNonInverting

LL_OPAMP_GetInputNonInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInverting (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_IO1
  - LL_OPAMP_INPUT_NONINVERT_DAC

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_GetInputNonInverting

LL_OPAMP_SetInputInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)

Function description

Set OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Return values

- **None:**

Notes

- OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetInputInverting

LL_OPAMP_GetInputInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputInverting (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetInputInverting

LL_OPAMP_SetTrimmingMode

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingMode (OPAMP_TypeDef * OPAMPx, uint32_t TrimmingMode)

Function description

Set OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TrimmingMode:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_SetTrimmingMode

LL_OPAMP_GetTrimmingMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_GetTrimmingMode

LL_OPAMP_SetCalibrationSelection

Function name

__STATIC_INLINE void LL_OPAMP_SetCalibrationSelection (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)

Function description

Set OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_SetCalibrationSelection

LL_OPAMP_GetCalibrationSelection

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetCalibrationSelection (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_GetCalibrationSelection

LL_OPAMP_IsCalibrationOutputSet

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsCalibrationOutputSet (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP calibration result of toggling output.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- This functions returns: 0 if OPAMP calibration output is reset 1 if OPAMP calibration output is set

Reference Manual to LL API cross reference:

- CSR OUTCAL LL_OPAMP_IsCalibrationOutputSet

LL_OPAMP_SetTrimmingValue

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair, uint32_t TrimmingValue)

Function description

Set OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS
- **TrimmingValue:** 0x00...0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_SetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_SetTrimmingValue
- HSOTR TRIMHSOFFSETN LL_OPAMP_SetTrimmingValue
- HSOTR TRIMHSOFFSETP LL_OPAMP_SetTrimmingValue

LL_OPAMP_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingValue (const OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair)

Function description

Get OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Return values

- **0x0...0x1F:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_GetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_GetTrimmingValue
- HSOTR TRIMHSOFFSETN LL_OPAMP_GetTrimmingValue
- HSOTR TRIMHSOFFSETP LL_OPAMP_GetTrimmingValue

LL_OPAMP_Enable

Function name

__STATIC_INLINE void LL_OPAMP_Enable (OPAMP_TypeDef * OPAMPx)

Function description

Enable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- After enable from off state, OPAMP requires a delay to fulfill wake up time specification. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Enable

LL_OPAMP_Disable

Function name

__STATIC_INLINE void LL_OPAMP_Disable (OPAMP_TypeDef * OPAMPx)

Function description

Disable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Disable

LL_OPAMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsEnabled (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP instance enable state (0: OPAMP is disabled, 1: OPAMP is enabled)

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_IsEnabled

LL_OPAMP_DeInit

Function name

ErrorStatus LL_OPAMP_DeInit (OPAMP_TypeDef * OPAMPx)

Function description

De-initialize registers of the selected OPAMP instance to their default reset values.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are de-initialized
  - ERROR: OPAMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_OPAMP_Init

Function name

ErrorStatus LL_OPAMP_Init (OPAMP_TypeDef * OPAMPx, const LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Initialize some features of OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance
- **OPAMP_InitStruct:** Pointer to a LL_OPAMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are initialized
  - ERROR: OPAMP registers are not initialized

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

LL_OPAMP_StructInit

Function name

void LL_OPAMP_StructInit (LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Set each LL_OPAMP_InitTypeDef field to default value.

Parameters

- **OPAMP_InitStruct:** pointer to a LL_OPAMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

106.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

106.3.1 OPAMP

OPAMP

OPAMP functional mode

LL_OPAMP_MODE_STANDALONE

OPAMP functional mode, OPAMP operation in standalone

LL_OPAMP_MODE_FOLLOWER

OPAMP functional mode, OPAMP operation in follower

LL_OPAMP_MODE_PGA

OPAMP functional mode, OPAMP operation in PGA

LL_OPAMP_MODE_PGA_IO0

In PGA mode, the inverting input is connected to VINM0 for filtering

LL_OPAMP_MODE_PGA_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0.

LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0.

Definitions of OPAMP hardware constraints delays

LL_OPAMP_DELAY_STARTUP_US

Delay for OPAMP startup time

OPAMP input inverting

LL_OPAMP_INPUT_INVERT_IO0

OPAMP inverting input connected to I/O VINM0 (PC5 for OPAMP1) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_IO1

OPAMP inverting input connected to I/0 VINM1 (PB1 for OPAMP1) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_CONNECT_NO

OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA with positive gain without bias). Note: On this STM32 series, this literal include cases of value 0x11 for mode follower and value 0x10 for mode PGA.

OPAMP input non-inverting

LL_OPAMP_INPUT_NONINVERT_IO0

OPAMP non inverting input connected to I/O VINP0 (PB0 for OPAMP1) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_IO1

OPAMP non inverting input connected to I/O VINP2 (PA0 for OPAMP1) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_DAC

OPAMP non inverting input connected internally to DAC channel (DAC1_CH1 for OPAMP1) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP mode calibration or functional.

LL_OPAMP_MODE_FUNCTIONAL

OPAMP functional mode

LL_OPAMP_MODE_CALIBRATION

OPAMP calibration mode

OPAMP PGA gain (relevant when OPAMP is in functional mode PGA)

LL_OPAMP_PGA_GAIN_2_OR_MINUS_1

OPAMP PGA gain 2 or -1

LL_OPAMP_PGA_GAIN_4_OR_MINUS_3

OPAMP PGA gain 4 or -3

LL_OPAMP_PGA_GAIN_8_OR_MINUS_7

OPAMP PGA gain 8 or -7

LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

OPAMP PGA gain 16 or -15

OPAMP PowerMode

LL_OPAMP_POWERMODE_NORMAL

OPAMP output in normal mode

LL_OPAMP_POWERMODE_HIGHSPEED

OPAMP output in highspeed mode

OPAMP trimming mode

LL_OPAMP_TRIMMING_FACTORY

OPAMP trimming factors set to factory values

LL_OPAMP_TRIMMING_USER

OPAMP trimming factors set to user values

OPAMP trimming of transistors differential pair NMOS or PMOS

LL_OPAMP_TRIMMING_NMOS_VREF_90PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.5*Vdda).

LL_OPAMP_TRIMMING_PMOS_VREF_10PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.1*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.33*Vdda).

LL_OPAMP_TRIMMING_NMOS

OPAMP trimming of transistors differential pair NMOS (internal reference voltage setto 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS

OPAMP trimming of transistors differential pair PMOS (internal reference voltage setto 0.1*Vdda). Default parameters to be used for calibration using two trimming steps one with each transistors differential pair NMOS and PMOS).

Common write and read registers macro

LL_OPAMP_WriteReg

Description:

- Write a value in OPAMP LL_OPAMP_GetPowerModeregister.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_OPAMP_ReadReg

Description:

- Read a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be read

Return value:

- Register: value

107 LL PKA Generic Driver

107.1 PKA Firmware driver registers structures

107.1.1 LL_PKA_InitTypeDef

LL_PKA_InitTypeDef is defined in the stm32h5xx_ll_pka.h

Data Fields

- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_PKA_InitTypeDef::Mode***
  Specifies the PKA operation mode. This parameter can be a value of ***PKA_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_PKA_SetMode()**.

107.2 PKA Firmware driver API description

The following section lists the various functions of the PKA library.

107.2.1 Detailed description of functions

LL_PKA_Config

Function name

__STATIC_INLINE void LL_PKA_Config (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Configure PKA peripheral.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_Config

LL_PKA_Enable

Function name

__STATIC_INLINE void LL_PKA_Enable (PKA_TypeDef * PKAx)

Function description

Enable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Enable

LL_PKA_Disable

Function name

__STATIC_INLINE void LL_PKA_Disable (PKA_TypeDef * PKAx)

Function description

Disable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Disable

LL_PKA_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabled (const PKA_TypeDef * PKAx)

Function description

Check if the PKA peripheral is enabled or disabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_PKA_IsEnabled

LL_PKA_SetMode

Function name

__STATIC_INLINE void LL_PKA_SetMode (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Set PKA operating mode.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_SetMode

LL_PKA_GetMode

Function name

__STATIC_INLINE uint32_t LL_PKA_GetMode (const PKA_TypeDef * PKAx)

Function description

Get PKA operating mode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_GetMode

LL_PKA_Start

Function name

__STATIC_INLINE void LL_PKA_Start (PKA_TypeDef * PKAx)

Function description

Start the operation selected using LL_PKA_SetMode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_PKA_Start

LL_PKA_EnableIT_ADDRERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_ADDRERR (PKA_TypeDef * PKAx)

Function description

Enable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_EnableIT_ADDRERR

LL_PKA_EnableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Enable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_EnableIT_RAMERR

LL_PKA_EnableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Enable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_EnableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_EnableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Enable end of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_EnableIT_PROCEND

LL_PKA_DisableIT_ADDERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_ADDERR (PKA_TypeDef * PKAx)

Function description

Disable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_DisableIT_ADDERR

LL_PKA_DisableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Disable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_DisableIT_RAMERR

LL_PKA_DisableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_DisableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Disable End of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_DisableIT_PROCEND

LL_PKA_DisableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Disable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_IsEnabledIT_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Check if address error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_IsEnabledIT_ADDRERR

LL_PKA_IsEnabledIT_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_RAMERR (const PKA_TypeDef * PKAx)

Function description

Check if RAM error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_IsEnabledIT_RAMERR

LL_PKA_IsEnabledIT_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_OPERR (const PKA_TypeDef * PKAx)

Function description

Check if OPERATION error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_IsEnabledIT_OPERR

LL_PKA_IsEnabledIT_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_PROCEND (const PKA_TypeDef * PKAx)

Function description

Check if end of operation interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_IsEnabledIT_PROCEND

LL_PKA_IsActiveFlag_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Get PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ADDRERRF LL_PKA_IsActiveFlag_ADDRERR

LL_PKA_IsActiveFlag_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_RAMERR (const PKA_TypeDef * PKAx)

Function description

Get PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RAMERRF LL_PKA_IsActiveFlag_RAMERR

LL_PKA_IsActiveFlag_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_OPERR (const PKA_TypeDef * PKAx)

Function description

Get PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OPERRF LL_PKA_IsActiveFlag_OPERR

LL_PKA_IsActiveFlag_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_PROCEND (const PKA_TypeDef * PKAx)

Function description

Get PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PROCENDF LL_PKA_IsActiveFlag_PROCEND

LL_PKA_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_BUSY (const PKA_TypeDef * PKAx)

Function description

Get PKA busy flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSY LL_PKA_IsActiveFlag_BUSY

LL_PKA_ClearFlag_ADDERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_ADDERR (PKA_TypeDef * PKAx)

Function description

Clear PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR ADDRERRFC LL_PKA_ClearFlag_ADDERR

LL_PKA_ClearFlag_RAMERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_RAMERR (PKA_TypeDef * PKAx)

Function description

Clear PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR RAMERRFC LL_PKA_ClearFlag_RAMERR

LL_PKA_ClearFlag_OPERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_OPERR (PKA_TypeDef * PKAx)

Function description

Clear PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR OPERRFC LL_PKA_ClearFlag_OPERR

LL_PKA_ClearFlag_PROCEND

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_PROCEND (PKA_TypeDef * PKAx)

Function description

Clear PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR PROCENDFC LL_PKA_ClearFlag_PROCEND

LL_PKA_DeInit

Function name

ErrorStatus LL_PKA_DeInit (const PKA_TypeDef * PKAx)

Function description

De-initialize PKA registers (Registers restored to their default values).

Parameters

- **PKAx:** PKA Instance.

Return values

- **ErrorStatus:**
  - SUCCESS: PKA registers are de-initialized
  - ERROR: PKA registers are not de-initialized

LL_PKA_Init

Function name

ErrorStatus LL_PKA_Init (PKA_TypeDef * PKAx, LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Initialize PKA registers according to the specified parameters in PKA_InitStruct.

Parameters

- **PKAx:** PKA Instance.
- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure that contains the configuration information for the specified PKA peripheral.

Return values

- **ErrorStatus:**
  - SUCCESS: PKA registers are initialized according to PKA_InitStruct content
  - ERROR: Not applicable

LL_PKA_StructInit

Function name

void LL_PKA_StructInit (LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Set each LL_PKA_InitTypeDef field to default value.

Parameters

- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

107.3 PKA Firmware driver defines

The following section lists the various define and macros of the module.

107.3.1 PKA

PKA

Get Flags Defines

LL_PKA_SR_ADDRERRF

LL_PKA_SR_RAMERRF

LL_PKA_SR_PROCENDF

LL_PKA_SR_BUSY

LL_PKA_SR_INITOK

LL_PKA_SR_OPERRF

IT Defines

LL_PKA_CR_ADDRERRIE

LL_PKA_CR_RAMERRIE

LL_PKA_CR_PROCENDIE

LL_PKA_CLRFR_PROCENDFC

LL_PKA_CLRFR_RAMERRFC

LL_PKA_CLRFR_ADDRERRFC

LL_PKA_CR_OPERRIE

LL_PKA_CLRFR_OPERRFC

Operation Mode

LL_PKA_MODE_MODULAR_EXP
: modular exponentiation

LL_PKA_MODE_MONTGOMERY_PARAM
: Compute Montgomery parameter only

LL_PKA_MODE_MODULAR_EXP_FAST
: modular exponentiation fast mode

LL_PKA_MODE_MODULAR_EXP_PROTECT
: modular exponentiation protect mode

LL_PKA_MODE_ECC_MUL
: compute ECC kP operation

LL_PKA_MODE_ECC_COMPLETE_ADD
: ECC complete addition

LL_PKA_MODE_ECDSA_SIGNATURE
: ECDSA signature

LL_PKA_MODE_ECDSA_VERIFICATION

ECDSA verification

LL_PKA_MODE_POINT_CHECK

Point check

LL_PKA_MODE_RSA_CRT_EXP

RSA CRT exponentiation

LL_PKA_MODE_MODULAR_INV

Modular inversion

LL_PKA_MODE_ARITHMETIC_ADD

Arithmetic addition

LL_PKA_MODE_ARITHMETIC_SUB

Arithmetic subtraction

LL_PKA_MODE_ARITHMETIC_MUL

Arithmetic multiplication

LL_PKA_MODE_COMPARISON

Comparison

LL_PKA_MODE_MODULAR_REDUC

Modular reduction

LL_PKA_MODE_MODULAR_ADD

Modular addition

LL_PKA_MODE_MODULAR_SUB

Modular subtraction

LL_PKA_MODE_MONTGOMERY_MUL

Montgomery multiplication

LL_PKA_MODE_DOUBLE_BASE_LADDER

Double base ladder

LL_PKA_MODE_ECC_PROJECTIVE_AFF

ECC projective to affine

Common Write and read registers Macros

LL_PKA_WriteReg

Description:

- Write a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PKA_ReadReg

Description:

- Read a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be read

Return value:

- Register: value

108 LL PWR Generic Driver

108.1 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

108.1.1 Detailed description of functions

LL_PWR_SetPowerMode

Function name

__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t Mode)

Function description

Set system power mode.

Parameters

- **Mode:** : This parameter can be one of the following values:
  - LL_PWR_STOP_MODE
  - LL_PWR_STANDBY_MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR LPMS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)

Function description

Get system power mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_STOP_MODE
  - LL_PWR_STANDBY_MODE

Reference Manual to LL API cross reference:

- PMCR LPMS LL_PWR_GetPowerMode

LL_PWR_SetStopModeRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetStopModeRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the internal Regulator output voltage in STOP mode.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE3
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE4
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SVOS LL_PWR_SetStopModeRegulVoltageScaling

LL_PWR_GetStopModeRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetStopModeRegulVoltageScaling (void)

Function description

Get the internal Regulator output voltage in STOP mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE3
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE4
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Reference Manual to LL API cross reference:

- PMCR SVOS LL_PWR_GetStopModeRegulVoltageScaling

LL_PWR_EnableFlashPowerDown

Function name

__STATIC_INLINE void LL_PWR_EnableFlashPowerDown (void)

Function description

Enable the Flash Power Down in Stop Mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR FLPS LL_PWR_EnableFlashPowerDown

LL_PWR_DisableFlashPowerDown

Function name

__STATIC_INLINE void LL_PWR_DisableFlashPowerDown (void)

Function description

Disable the Flash Power Down in Stop Mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR FLPS LL_PWR_DisableFlashPowerDown

LL_PWR_IsEnabledFlashPowerDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledFlashPowerDown (void)

Function description

Check if the Flash Power Down in Stop Mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR FLPS LL_PWR_IsEnabledFlashPowerDown

LL_PWR_EnableAnalogBooster

Function name

__STATIC_INLINE void LL_PWR_EnableAnalogBooster (void)

Function description

Enable the Analog Voltage Booster (VDDA)

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR BOOSTE LL_PWR_EnableAnalogBooster

LL_PWR_DisableAnalogBooster

Function name

__STATIC_INLINE void LL_PWR_DisableAnalogBooster (void)

Function description

Disable the Analog Voltage Booster (VDDA)

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR BOOSTE LL_PWR_DisableAnalogBooster

LL_PWR_IsEnabledAnalogBooster

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAnalogBooster (void)

Function description

Check if the Analog Voltage Booster (VDDA) is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR BOOSTE LL_PWR_IsEnabledAnalogBooster

LL_PWR_EnableAnalogVoltageReady

Function name

__STATIC_INLINE void LL_PWR_EnableAnalogVoltageReady (void)

Function description

Enable the Analog Voltage Ready to isolate the BOOST IP until VDDA will be ready.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR AVD_READY LL_PWR_EnableAnalogVoltageReady

LL_PWR_DisableAnalogVoltageReady

Function name

__STATIC_INLINE void LL_PWR_DisableAnalogVoltageReady (void)

Function description

Disable the Analog Voltage Ready (VDDA)

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR AVD_READY LL_PWR_DisableAnalogVoltageReady

LL_PWR_IsEnabledAnalogVoltageReady

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAnalogVoltageReady (void)

Function description

Check if the Analog Voltage Booster (VDDA) is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR AVD_READY LL_PWR_IsEnabledAnalogVoltageReady

LL_PWR_EnableAHBRAM1ShutOff

Function name

__STATIC_INLINE void LL_PWR_EnableAHBRAM1ShutOff (void)

Function description

Enable the AHB RAM1 shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM1SO LL_PWR_EnableAHBRAM1ShutOff

LL_PWR_DisableAHBRAM1ShutOff

Function name

__STATIC_INLINE void LL_PWR_DisableAHBRAM1ShutOff (void)

Function description

Disable the AHB RAM1 shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM1SO LL_PWR_DisableAHBRAM1ShutOff

LL_PWR_IsEnabledAHBRAM1ShutOff

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAHBRAM1ShutOff (void)

Function description

Check if the AHB RAM1 shut-off in Stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SRAM1SO LL_PWR_IsEnabledAHBRAM1ShutOff

LL_PWR_EnableAHBRAM2_48K_ShutOff

Function name

__STATIC_INLINE void LL_PWR_EnableAHBRAM2_48K_ShutOff (void)

Function description

Enable the AHB RAM2 48K Bytes shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM2_48SO LL_PWR_EnableAHBRAM2_48K_ShutOff

LL_PWR_DisableAHBRAM2_48K_ShutOff

Function name

__STATIC_INLINE void LL_PWR_DisableAHBRAM2_48K_ShutOff (void)

Function description

Disable the AHB RAM2 48K Bytes shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM2_48SO LL_PWR_DisableAHBRAM2_48K_ShutOff

LL_PWR_IsEnabledAHBRAM2_48K_ShutOff

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAHBRAM2_48K_ShutOff (void)

Function description

Check if the AHB RAM2 shut-off in Stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR SRAM2_48SO LL_PWR_IsEnabledAHBRAM2_48K_ShutOff

LL_PWR_EnableAHBRAM2_16K_ShutOff

Function name

__STATIC_INLINE void LL_PWR_EnableAHBRAM2_16K_ShutOff (void)

Function description

Enable the AHB RAM2 16K Bytes shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM2_16SO LL_PWR_EnableAHBRAM2_16K_ShutOff

LL_PWR_DisableAHBRAM2_16K_ShutOff

Function name

__STATIC_INLINE void LL_PWR_DisableAHBRAM2_16K_ShutOff (void)

Function description

Disable the AHB RAM2 16K Bytes shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM2_16SO LL_PWR_DisableAHBRAM2_16K_ShutOff

LL_PWR_IsEnabledAHBRAM2_16K_ShutOff

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAHBRAM2_16K_ShutOff (void)

Function description

Check if the AHB RAM2 shut-off in Stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR SRAM2_16SO LL_PWR_IsEnabledAHBRAM2_16K_ShutOff

LL_PWR_EnableAHBRAM3ShutOff

Function name

__STATIC_INLINE void LL_PWR_EnableAHBRAM3ShutOff (void)

Function description

Enable the AHB RAM3 shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM3SO LL_PWR_EnableAHBRAM3ShutOff

LL_PWR_DisableAHBRAM3ShutOff

Function name

__STATIC_INLINE void LL_PWR_DisableAHBRAM3ShutOff (void)

Function description

Disable the AHB RAM3 shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR SRAM3SO LL_PWR_DisableAHBRAM3ShutOff

LL_PWR_IsEnabledAHBRAM3ShutOff

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAHBRAM3ShutOff (void)

Function description

Check if the AHB RAM3 shut-off in Stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR SRAM3SO LL_PWR_IsEnabledAHBRAM3ShutOff

LL_PWR_EnableETHERNETRAMShutOff

Function name

__STATIC_INLINE void LL_PWR_EnableETHERNETRAMShutOff (void)

Function description

Enable the ETHERNET RAM shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR ETHERNETSO LL_PWR_EnableETHERNETRAMShutOff

LL_PWR_DisableETHERNETRAMShutOff

Function name

__STATIC_INLINE void LL_PWR_DisableETHERNETRAMShutOff (void)

Function description

Disable the ETHERNET RAM shut-off in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR ETHERNETSO LL_PWR_DisableETHERNETRAMShutOff

LL_PWR_IsEnabledETHERNETRAMShutOff

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledETHERNETRAMShutOff (void)

Function description

Check if the ETHERNET RAM shut-off in Stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMCR ETHERNETSO LL_PWR_IsEnabledETHERNETRAMShutOff

LL_PWR_SetRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the regulator supply output voltage.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE0
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSCR VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)

Function description

Get the regulator supply output voltage.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE0
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3

Reference Manual to LL API cross reference:

- VOSCR VOS LL_PWR_GetRegulVoltageScaling

LL_PWR_GetCurrentVOS

Function name

__STATIC_INLINE uint32_t LL_PWR_GetCurrentVOS (void)

Function description

Get currently voltage scaling applied to VCORE.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE0
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3

Reference Manual to LL API cross reference:

- VOSSR ACTVOS[1:0] LL_PWR_GetCurrentVOS

LL_PWR_EnableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpRegulator (void)

Function description

Enable Backup Regulator.

Return values

- **None:**

Notes

- When set, the Backup Regulator (used to maintain backup SRAM content in Standby and VBAT modes) is enabled. If BRE is reset, the backup Regulator is switched off. The backup SRAM can still be used but its content will be lost in the Standby and VBAT modes. Once set, the application must wait that the Backup Regulator Ready flag (BRR) is set to indicate that the data written into the RAM will be maintained in the Standby and VBAT modes.

Reference Manual to LL API cross reference:

- BDCR BREN LL_PWR_EnableBkUpRegulator

LL_PWR_DisableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpRegulator (void)

Function description

Disable Backup Regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BREN LL_PWR_DisableBkUpRegulator

LL_PWR_IsEnabledBkUpRegulator

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpRegulator (void)

Function description

Check if the backup Regulator is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR BREN LL_PWR_IsEnabledBkUpRegulator

LL_PWR_EnableMonitoring

Function name

__STATIC_INLINE void LL_PWR_EnableMonitoring (void)

Function description

Enable VBAT and Temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR MONEN LL_PWR_EnableMonitoring

LL_PWR_DisableMonitoring

Function name

__STATIC_INLINE void LL_PWR_DisableMonitoring (void)

Function description

Disable VBAT and Temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR MONEN LL_PWR_DisableMonitoring

LL_PWR_IsEnabledMonitoring

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledMonitoring (void)

Function description

Check if the VBAT and Temperature monitoring is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR MONEN LL_PWR_IsEnabledMonitoring

LL_PWR_EnableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_EnableBatteryCharging (void)

Function description

Enable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR VBE LL_PWR_EnableBatteryCharging

LL_PWR_DisableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_DisableBatteryCharging (void)

Function description

Disable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR VBE LL_PWR_DisableBatteryCharging

LL_PWR_IsEnabledBatteryCharging

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBatteryCharging (void)

Function description

Check if battery charging is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR VBE LL_PWR_IsEnabledBatteryCharging

LL_PWR_SetBattChargResistor

Function name

__STATIC_INLINE void LL_PWR_SetBattChargResistor (uint32_t Resistor)

Function description

Set the Battery charge resistor impedance.

Parameters

- **Resistor:** This parameter can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARG_RESISTOR_1_5K

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR VBRS LL_PWR_SetBattChargResistor

LL_PWR_GetBattChargResistor

Function name

__STATIC_INLINE uint32_t LL_PWR_GetBattChargResistor (void)

Function description

Get the Battery charge resistor impedance.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARG_RESISTOR_1_5K

Reference Manual to LL API cross reference:

- BDCR VBRS LL_PWR_GetBattChargResistor

LL_PWR_EnableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBPCR DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBPCR DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)

Function description

Check if the backup domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DBPCR DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_EnableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDStandbyMode (void)

Function description

Enable the USB type-C and power delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called just before entering Standby mode.

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_EnableUCPDStandbyMode

LL_PWR_DisableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDStandbyMode (void)

Function description

Disable the USB type-C and power delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called after exiting Standby mode and before any UCPD configuration update.

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_DisableUCPDStandbyMode

LL_PWR_IsEnabledUCPDStandbyMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDStandbyMode (void)

Function description

Check if the USB Type-C and Power Delivery Standby mode memorization is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_IsEnabledUCPDStandbyMode

LL_PWR_EnableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDDeadBattery (void)

Function description

Enable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_EnableUCPDDeadBattery

LL_PWR_DisableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDDeadBattery (void)

Function description

Disable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_DisableUCPDDeadBattery

LL_PWR_IsEnabledUCPDDeadBattery

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDDeadBattery (void)

Function description

Check the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **State:** of feature (1 : enabled; 0 : disabled).

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_IsEnabledUCPDDeadBattery

LL_PWR_ConfigSupply

Function name

__STATIC_INLINE void LL_PWR_ConfigSupply (uint32_t SupplySource)

Function description

Configure the PWR supply.

Parameters

- **SupplySource:** This parameter can be one of the following values:
  - LL_PWR_EXTERNAL_SOURCE_SUPPLY

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCCR BYPASS LL_PWR_ConfigSupply

LL_PWR_GetSupply

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSupply (void)

Function description

Get the PWR supply.

Return values

- **The:** supply configuration.

Reference Manual to LL API cross reference:

- SCCR BYPASS LL_PWR_GetSupply

LL_PWR_EnablePVD

Function name

__STATIC_INLINE void LL_PWR_EnablePVD (void)

Function description

Enable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR PVDEN LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name

__STATIC_INLINE void LL_PWR_DisablePVD (void)

Function description

Disable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR PVDEN LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)

Function description

Check if Power Voltage Detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMCR PVDEN LL_PWR_IsEnabledPVD

LL_PWR_SetPVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)

Function description

Configure the voltage threshold detected by the Power Voltage Detector.

Parameters

- **PVDLevel:** This parameter can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)

Function description

Get the voltage threshold detection.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:

- VMCR PLS LL_PWR_GetPVDLevel

LL_PWR_EnableAVD

Function name

__STATIC_INLINE void LL_PWR_EnableAVD (void)

Function description

Enable Analog Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR AVDEN LL_PWR_EnableAVD

LL_PWR_DisableAVD

Function name

__STATIC_INLINE void LL_PWR_DisableAVD (void)

Function description

Disable Analog Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR AVDEN LL_PWR_DisableAVD

LL_PWR_IsEnabledAVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAVD (void)

Function description

Check if Analog Power Voltage Detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMCR AVDEN LL_PWR_IsEnabledAVD

LL_PWR_SetAVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetAVDLevel (uint32_t AVDLevel)

Function description

Configure the voltage threshold to be detected by the Analog Power Voltage Detector.

Parameters

- **AVDLevel:** This parameter can be one of the following values:
  - LL_PWR_AVDLEVEL_0
  - LL_PWR_AVDLEVEL_1
  - LL_PWR_AVDLEVEL_2
  - LL_PWR_AVDLEVEL_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- VMCR ALS LL_PWR_SetAVDLevel

LL_PWR_GetAVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetAVDLevel (void)

Function description

Get the Analog Voltage threshold to be detected by the Analog Power Voltage Detector.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_AVDLEVEL_0
  - LL_PWR_AVDLEVEL_1
  - LL_PWR_AVDLEVEL_2
  - LL_PWR_AVDLEVEL_3

Reference Manual to LL API cross reference:

- CR1 ALS LL_PWR_GetAVDLevel

LL_PWR_EnableUSBVoltageDetector

Function name

__STATIC_INLINE void LL_PWR_EnableUSBVoltageDetector (void)

Function description

Enable the USB voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- USBSCR USB33DEN LL_PWR_EnableUSBVoltageDetector

LL_PWR_DisableUSBVoltageDetector

Function name

__STATIC_INLINE void LL_PWR_DisableUSBVoltageDetector (void)

Function description

Disable the USB voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- USBSCR USB33DEN LL_PWR_DisableUSBVoltageDetector

LL_PWR_IsEnabledUSBVoltageDetector

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUSBVoltageDetector (void)

Function description

Check if the USB voltage detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- USBSCR USB33DEN LL_PWR_IsEnabledUSBVoltageDetector

LL_PWR_EnableVDDUSB

Function name

__STATIC_INLINE void LL_PWR_EnableVDDUSB (void)

Function description

Enable the independent USB supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- USBSCR USB33SV LL_PWR_EnableVDDUSB

LL_PWR_DisableVDDUSB

Function name

__STATIC_INLINE void LL_PWR_DisableVDDUSB (void)

Function description

Disable the independent USB supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- USBSCR USB33SV LL_PWR_DisableVDDUSB

LL_PWR_IsEnabledVDDUSB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVDDUSB (void)

Function description

Check if the independent USB supply is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- USBSCR USB33SV LL_PWR_IsEnabledVDDUSB

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the wake up pin_x.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WUPENx LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the wake up pin_x.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WUPENx LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)

Function description

Check if the wake up pin_x is enabled.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUCR WUPPx LL_PWR_IsEnabledWakeUpPin

LL_PWR_SetWakeUpPinPolarityLow

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity low for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WKUPP1 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP2 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP3 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP4 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP5 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP6 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP7 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WKUPP8 LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPolarityHigh

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityHigh (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity high for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WKUPP1 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP2 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP3 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP4 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP5 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP6 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP7 LL_PWR_SetWakeUpPinPolarityHigh
- WUCR WKUPP8 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name

__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Get the Wake-Up pin polarity for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUCR WUPP1 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP2 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP3 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP4 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP5 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP6 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP7 LL_PWR_SetWakeUpPinPolarityLow
- WUCR WUPP8 LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPullNone

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullNone (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull None.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WUPPUPD1 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD2 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD3 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD4 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD5 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD6 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD7 LL_PWR_SetWakeUpPinPullNone
- WUCR WUPPUPD8 LL_PWR_SetWakeUpPinPullNone

LL_PWR_SetWakeUpPinPullUp

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullUp (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull Up.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WUPPUPD1 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD2 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD3 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD4 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD5 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD6 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD7 LL_PWR_SetWakeUpPinPullUp
- WUCR WUPPUPD8 LL_PWR_SetWakeUpPinPullUp

LL_PWR_SetWakeUpPinPullDown

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullDown (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull Down.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR WUPPUPD1 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD2 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD3 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD4 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD5 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD6 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD7 LL_PWR_SetWakeUpPinPullDown
- WUCR WUPPUPD8 LL_PWR_SetWakeUpPinPullDown

LL_PWR_GetWakeUpPinPull

Function name

__STATIC_INLINE uint32_t LL_PWR_GetWakeUpPinPull (uint32_t WakeUpPin)

Function description

Get the Wake-Up pin pull.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_WAKEUP_PIN_NOPULL
  - LL_PWR_WAKEUP_PIN_PULLUP
  - LL_PWR_WAKEUP_PIN_PULLDOWN

Reference Manual to LL API cross reference:

- WUCR WUPPUPD1 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD2 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD3 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD4 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD5 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD7 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD7 LL_PWR_GetWakeUpPinPull
- WUCR WUPPUPD8 LL_PWR_GetWakeUpPinPull

LL_PWR_EnableIORetention

Function name

__STATIC_INLINE void LL_PWR_EnableIORetention (void)

Function description

Enable IO Retention.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IORETR IORETEN LL_PWR_EnableIORetention

LL_PWR_DisableIORetention

Function name

__STATIC_INLINE void LL_PWR_DisableIORetention (void)

Function description

Disable IO Retention.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IORETR IORETEN LL_PWR_DisableIORetention

LL_PWR_IsEnabledIORetention

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledIORetention (void)

Function description

Check if IO Retention is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IORETR IORETEN LL_PWR_IsEnabledIORetention

LL_PWR_EnableJTAGIORetention

Function name

__STATIC_INLINE void LL_PWR_EnableJTAGIORetention (void)

Function description

Enable JTAGIO Retention.

Return values

- **None:**

Reference Manual to LL API cross reference:

- JTAGIORETR JTAGIORETEN LL_PWR_EnableJTAGIORetention

LL_PWR_DisableJTAGIORetention

Function name

__STATIC_INLINE void LL_PWR_DisableJTAGIORetention (void)

Function description

Disable JTAGIO Retention.

Return values

- **None:**

Reference Manual to LL API cross reference:

- JTAGIORETR JTAGIORETEN LL_PWR_DisableJTAGIORetention

LL_PWR_IsEnabledJTAGIORetention

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledJTAGIORetention (void)

Function description

Check if JTAGIO Retention is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IORETR JTAGIORETEN LL_PWR_IsEnabledJTAGIORetention

LL_PWR_IsActiveFlag_VOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOS (void)

Function description

Indicate whether the regulator voltage output is above voltage scaling range or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSSR VOSRDY LL_PWR_IsActiveFlag_VOS

LL_PWR_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)

Function description

Indicate whether the system was in standby mode or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMSR SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_STOP (void)

Function description

Indicate whether the system was in stop mode or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PMSR STOPF LL_PWR_IsActiveFlag_STOP

LL_PWR_IsActiveFlag_PVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)

Function description

Indicate whether the VDD voltage is below the threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMSR PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_AVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_AVDO (void)

Function description

Indicate whether the VDD voltage is below the threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMSR AVDO LL_PWR_IsActiveFlag_AVDO

LL_PWR_IsActiveFlag_ACTVOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_ACTVOS (void)

Function description

Indicate whether the regulator voltage output is equal to current used voltage scaling range or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSSR ACTVOSRDY LL_PWR_IsActiveFlag_ACTVOS

LL_PWR_IsActiveFlag_VDDUSB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDUSB (void)

Function description

Indicate whether the VDDUSB is below the threshold of monitor or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMSR USB33RDY LL_PWR_IsActiveFlag_VDDUSB

LL_PWR_IsActiveFlag_VDDIO2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDIO2 (void)

Function description

Indicate whether VDDMMC voltage is below 1V2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VMSR VDDIO2RDY LL_PWR_IsActiveFlag_VDDIO2

LL_PWR_IsActiveFlag_BRR

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_BRR (void)

Function description

Get Backup Regulator ready Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR BRRDY LL_PWR_IsActiveFlag_BRR

LL_PWR_IsActiveFlag_VBATL

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VBATL (void)

Function description

Indicate whether the VBAT level is below high threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR VBATL LL_PWR_IsActiveFlag_VBATL

LL_PWR_IsActiveFlag_VBATH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VBATH (void)

Function description

Indicate whether the VBAT level is below high threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR VBATH LL_PWR_IsActiveFlag_VBATH

LL_PWR_IsActiveFlag_TEMPL

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPL (void)

Function description

Indicate whether the CPU temperature level is above low threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR TEMPL LL_PWR_IsActiveFlag_TEMPL

LL_PWR_IsActiveFlag_TEMPH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPH (void)

Function description

Indicate whether the CPU temperature level is below high threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR TEMPH LL_PWR_IsActiveFlag_TEMPH

LL_PWR_IsActiveFlag_WU1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 1.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF1 LL_PWR_IsActiveFlag_WU1

LL_PWR_IsActiveFlag_WU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 3.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 4.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU5

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 5.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU6

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU6 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 6.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF6 LL_PWR_IsActiveFlag_WU6

LL_PWR_IsActiveFlag_WU7

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU7 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 7.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF7 LL_PWR_IsActiveFlag_WU7

LL_PWR_IsActiveFlag_WU8

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU8 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 8.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF8 LL_PWR_IsActiveFlag_WU8

LL_PWR_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_STOP (void)

Function description

Clear stop flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR CSSF LL_PWR_ClearFlag_STOP

LL_PWR_ClearFlag_SB

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)

Function description

Clear standby flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR CSSF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU1

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU1 (void)

Function description

Clear wake up flag 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF1 LL_PWR_ClearFlag_WU1

LL_PWR_ClearFlag_WU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU2 (void)

Function description

Clear wake up flag 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU3

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU3 (void)

Function description

Clear wake up flag 3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU4

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU4 (void)

Function description

Clear wake up flag 4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU5

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU5 (void)

Function description

Clear wake up flag 5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU6

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU6 (void)

Function description

Clear wake up flag 6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF6 LL_PWR_ClearFlag_WU6

LL_PWR_ClearFlag_WU7

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU7 (void)

Function description

Clear wake up flag 7.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF7 LL_PWR_ClearFlag_WU7

LL_PWR_ClearFlag_WU8

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU8 (void)

Function description

Clear wake up flag 8.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF8 LL_PWR_ClearFlag_WU8

LL_PWR_ClearFlag_WU

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)

Function description

Clear all wake up flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF LL_PWR_ClearFlag_WU

LL_PWR_EnableNSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_EnableNSecurePrivilege (void)

Function description

Enable privileged mode for nsecure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_EnableNSecurePrivilege

LL_PWR_DisableNSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_DisableNSecurePrivilege (void)

Function description

Disable privileged mode for nsecure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_DisableNSecurePrivilege

LL_PWR_IsEnabledNSecurePrivilege

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledNSecurePrivilege (void)

Function description

Check if privileged mode for nsecure items is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_IsEnabledNSecurePrivilege

LL_PWR_EnableSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_EnableSecurePrivilege (void)

Function description

Enable privileged mode for secure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_EnableSecurePrivilege

LL_PWR_DisableSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_DisableSecurePrivilege (void)

Function description

Disable privileged mode for secure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_DisableSecurePrivilege

LL_PWR_IsEnabledSecurePrivilege

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledSecurePrivilege (void)

Function description

Check if privileged mode for secure items is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_IsEnabledSecurePrivilege

LL_PWR_ConfigSecure

Function name

__STATIC_INLINE void LL_PWR_ConfigSecure (uint32_t SecureConfig)

Function description

Configure secure attribute mode.

Parameters

- **SecureConfig:** This parameter can be the full combination of the following values:
  - LL_PWR_WAKEUP_PIN1_NSEC or LL_PWR_WAKEUP_PIN1_SEC
  - LL_PWR_WAKEUP_PIN2_NSEC or LL_PWR_WAKEUP_PIN2_SEC
  - LL_PWR_WAKEUP_PIN3_NSEC or LL_PWR_WAKEUP_PIN3_SEC
  - LL_PWR_WAKEUP_PIN4_NSEC or LL_PWR_WAKEUP_PIN4_SEC
  - LL_PWR_WAKEUP_PIN5_NSEC or LL_PWR_WAKEUP_PIN5_SEC
  - LL_PWR_WAKEUP_PIN6_NSEC or LL_PWR_WAKEUP_PIN6_SEC
  - LL_PWR_WAKEUP_PIN7_NSEC or LL_PWR_WAKEUP_PIN7_SEC
  - LL_PWR_WAKEUP_PIN8_NSEC or LL_PWR_WAKEUP_PIN8_SEC
  - LL_PWR_RET_NSEC or LL_PWR_RET_SEC
  - LL_PWR_LPM_NSEC or LL_PWR_LPM_SEC
  - LL_PWR_VDM_NSEC or LL_PWR_VDM_SEC
  - LL_PWR_VB_NSEC or LL_PWR_VB_SEC
  - LL_PWR_APC_NSEC or LL_PWR_APC_SEC

Return values

- **None.:**

Notes

- This API can be executed only by CPU in secure mode.

Reference Manual to LL API cross reference:

- SECCFGR WUP1SEC LL_PWR_ConfigSecure
- SECCFGR WUP2SEC LL_PWR_ConfigSecure
- SECCFGR WUP3SEC LL_PWR_ConfigSecure
- SECCFGR WUP4SEC LL_PWR_ConfigSecure
- SECCFGR WUP5SEC LL_PWR_ConfigSecure
- SECCFGR WUP6SEC LL_PWR_ConfigSecure
- SECCFGR WUP7SEC LL_PWR_ConfigSecure
- SECCFGR WUP8SEC LL_PWR_ConfigSecure
- SECCFGR RETSEC LL_PWR_ConfigSecure
- SECCFGR LPMSEC LL_PWR_ConfigSecure
- SECCFGR VDMSEC LL_PWR_ConfigSecure
- SECCFGR VBSEC LL_PWR_ConfigSecure
- SECCFGR APCSEC LL_PWR_ConfigSecure

LL_PWR_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_PWR_GetConfigSecure (void)

Function description

Get secure attribute configuration.

Return values

- **Returned:** value is the combination of the following values:
  - LL_PWR_WAKEUP_PIN1_NSEC or LL_PWR_WAKEUP_PIN1_SEC
  - LL_PWR_WAKEUP_PIN2_NSEC or LL_PWR_WAKEUP_PIN2_SEC
  - LL_PWR_WAKEUP_PIN3_NSEC or LL_PWR_WAKEUP_PIN3_SEC
  - LL_PWR_WAKEUP_PIN4_NSEC or LL_PWR_WAKEUP_PIN4_SEC
  - LL_PWR_WAKEUP_PIN5_NSEC or LL_PWR_WAKEUP_PIN5_SEC
  - LL_PWR_WAKEUP_PIN6_NSEC or LL_PWR_WAKEUP_PIN6_SEC
  - LL_PWR_WAKEUP_PIN7_NSEC or LL_PWR_WAKEUP_PIN7_SEC
  - LL_PWR_WAKEUP_PIN8_NSEC or LL_PWR_WAKEUP_PIN8_SEC
  - LL_PWR_RET_NSEC or LL_PWR_RET_SEC
  - LL_PWR_LPM_NSEC or LL_PWR_LPM_SEC
  - LL_PWR_VDM_NSEC or LL_PWR_VDM_SEC
  - LL_PWR_VB_NSEC or LL_PWR_VB_SEC
  - LL_PWR_APC_NSEC or LL_PWR_APC_SEC

Notes

- This API can be executed only by CPU in secure mode.

Reference Manual to LL API cross reference:

- SECCFGR WUP1SEC LL_PWR_GetConfigSecure
- SECCFGR WUP2SEC LL_PWR_GetConfigSecure
- SECCFGR WUP3SEC LL_PWR_GetConfigSecure
- SECCFGR WUP4SEC LL_PWR_GetConfigSecure
- SECCFGR WUP5SEC LL_PWR_GetConfigSecure
- SECCFGR WUP6SEC LL_PWR_GetConfigSecure
- SECCFGR WUP7SEC LL_PWR_GetConfigSecure
- SECCFGR WUP8SEC LL_PWR_GetConfigSecure
- SECCFGR RETSEC LL_PWR_ConfigSecure
- SECCFGR LPMSEC LL_PWR_GetConfigSecure
- SECCFGR VDMSEC LL_PWR_GetConfigSecure
- SECCFGR VBSEC LL_PWR_GetConfigSecure
- SECCFGR APCSEC LL_PWR_GetConfigSecure

LL_PWR_DeInit

Function name

ErrorStatus LL_PWR_DeInit (void)

Function description

De-initialize the PWR registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : PWR registers are de-initialized.
  - ERROR : not applicable.

108.2 PWR Firmware driver defines

The following section lists the various define and macros of the module.

108.2.1 PWR

PWR

Power Analog Voltage Level Detector

LL_PWR_AVDLEVEL_0

Analog Voltage threshold detected by AVD 1.7 V

LL_PWR_AVDLEVEL_1

Analog Voltage threshold detected by AVD 2.1 V

LL_PWR_AVDLEVEL_2

Analog Voltage threshold detected by AVD 2.5 V

LL_PWR_AVDLEVEL_3

Analog Voltage threshold detected by AVD 2.8 V

PWR VBAT Charging Resistor Selection

LL_PWR_BATT_CHARG_RESISTOR_5K

Charge the battery through a 5 kO resistor

LL_PWR_BATT_CHARG_RESISTOR_1_5K

Charge the battery through a 1.5 kO resistor

Clear Flags Defines

LL_PWR_PMCR_CSSF

Clear STOP and STANDBY flags

LL_PWR_WUSCR_CWUF1

Clear Wakeup flag 1

LL_PWR_WUSCR_CWUF2

Clear Wakeup flag 2

LL_PWR_WUSCR_CWUF3

Clear Wakeup flag 3

LL_PWR_WUSCR_CWUF4

Clear Wakeup flag 4

LL_PWR_WUSCR_CWUF5

Clear Wakeup flag 5

LL_PWR_WUSCR_CWUF6

Clear Wakeup flag 6

LL_PWR_WUSCR_CWUF7

Clear Wakeup flag 7

LL_PWR_WUSCR_CWUF8

Clear Wakeup flag 8

LL_PWR_WUSCR_CWUF_ALL

Clear all Wakeup flags

Get Flags Defines

LL_PWR_FLAG_VOSRDY

Voltage scaling ready flag

LL_PWR_FLAG_ACTVOSRDY

Currently applied VOS ready flag

LL_PWR_FLAG_STOPF

STOP flag

LL_PWR_FLAG_SBF

STANDBY flag

LL_PWR_FLAG_AVDO

Analog voltage detector output on VDDA flag

LL_PWR_FLAG_VDDIO2RDY

VDDIO2 ready flag

LL_PWR_FLAG_PVDO

Programmable voltage detect output flag

LL_PWR_FLAG_USB33RDY

VDDUSB ready flag

LL_PWR_FLAG_TEMPH

Temperature level flag (versus high threshold)

LL_PWR_FLAG_TEMPL

Temperature level flag (versus low threshold)

LL_PWR_FLAG_VBATH

VBAT level flag (versus high threshold)

LL_PWR_FLAG_VBATL

VBAT level flag (versus low threshold)

LL_PWR_WAKEUP_FLAG1

Wakeup flag 1

LL_PWR_WAKEUP_FLAG2

Wakeup flag 2

LL_PWR_WAKEUP_FLAG3

Wakeup flag 3

LL_PWR_WAKEUP_FLAG4

Wakeup flag 4

LL_PWR_WAKEUP_FLAG5

Wakeup flag 5

LL_PWR_WAKEUP_FLAG6

Wakeup flag 6

LL_PWR_WAKEUP_FLAG7

Wakeup flag 7

LL_PWR_WAKEUP_FLAG8

Wakeup flag 8

PWR Items Secure Attribute

LL_PWR_WAKEUP_PIN1_NSEC

LL_PWR_WAKEUP_PIN1_SEC

LL_PWR_WAKEUP_PIN2_NSEC

LL_PWR_WAKEUP_PIN2_SEC

LL_PWR_WAKEUP_PIN3_NSEC

LL_PWR_WAKEUP_PIN3_SEC

LL_PWR_WAKEUP_PIN4_NSEC

LL_PWR_WAKEUP_PIN4_SEC

LL_PWR_WAKEUP_PIN5_NSEC

LL_PWR_WAKEUP_PIN5_SEC

LL_PWR_WAKEUP_PIN6_NSEC

LL_PWR_WAKEUP_PIN6_SEC

LL_PWR_WAKEUP_PIN7_NSEC

LL_PWR_WAKEUP_PIN7_SEC

LL_PWR_WAKEUP_PIN8_NSEC

LL_PWR_WAKEUP_PIN8_SEC

LL_PWR_RET_NSEC

LL_PWR_RET_SEC

LL_PWR_LPM_NSEC

LL_PWR_LPM_SEC

LL_PWR_VDM_NSEC

LL_PWR_VDM_SEC

LL_PWR_VB_NSEC

LL_PWR_VB_SEC

LL_PWR_APC_NSEC

LL_PWR_APC_SEC

Low Power Mode Selection

LL_PWR_STOP_MODE

STOP 0 mode

LL_PWR_STANDBY_MODE

STANDBY mode

PWR Power Voltage Detector Level Selection

LL_PWR_PVDLEVEL_0

Voltage threshold detected by PVD 1.95 V

LL_PWR_PVDLEVEL_1

Voltage threshold detected by PVD 2.10 V

LL_PWR_PVDLEVEL_2

Voltage threshold detected by PVD 2.25 V

LL_PWR_PVDLEVEL_3

Voltage threshold detected by PVD 2.40 V

LL_PWR_PVDLEVEL_4

Voltage threshold detected by PVD 2.55 V

LL_PWR_PVDLEVEL_5

Voltage threshold detected by PVD 2.70 V

LL_PWR_PVDLEVEL_6

Voltage threshold detected by PVD 2.85 V

LL_PWR_PVDLEVEL_7

External input analog voltage on PVD_IN pin, compared to internal VREFINT level

Stop mode Regulator Voltage Scaling

LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Select voltage scale 5 when system enters STOP mode

LL_PWR_REGU_VOLTAGE_SVOS_SCALE4

Select voltage scale 4 when system enters STOP mode

LL_PWR_REGU_VOLTAGE_SVOS_SCALE3

Select voltage scale 3 when system enters STOP mode

Power supply source configuration

LL_PWR_EXTERNAL_SOURCE_SUPPLY

The SMPS and the LDO are Bypassed. The Core domains are supplied from an external source

PWR Voltage scaling range selection

LL_PWR_REGU_VOLTAGE_SCALE0

Voltage scaling range 0

LL_PWR_REGU_VOLTAGE_SCALE1

Voltage scaling range 1

LL_PWR_REGU_VOLTAGE_SCALE2

Voltage scaling range 2

LL_PWR_REGU_VOLTAGE_SCALE3

Voltage scaling range 3

PWR Wake Up Pin

LL_PWR_WAKEUP_PIN1

Wakeup pin 1 enable

LL_PWR_WAKEUP_PIN2

Wakeup pin 2 enable

LL_PWR_WAKEUP_PIN3

Wakeup pin 3 enable

LL_PWR_WAKEUP_PIN4

Wakeup pin 4 enable

LL_PWR_WAKEUP_PIN5

Wakeup pin 5 enable

LL_PWR_WAKEUP_PIN6

Wakeup pin 6 enable

LL_PWR_WAKEUP_PIN7

Wakeup pin 7 enable

LL_PWR_WAKEUP_PIN8

Wakeup pin 8 enable

Wakeup Pins pull configuration

LL_PWR_WAKEUP_PIN_NOPULL

Configure Wake-Up pin in no pull

LL_PWR_WAKEUP_PIN_PULLUP

Configure Wake-Up pin in pull Up

LL_PWR_WAKEUP_PIN_PULLDOWN

Configure Wake-Up pin in pull Down

Common Write and Read Registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written.
- __VALUE__: Value to be written in the register.

Return value:

- None.

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read.

Return value:

- Register: value.

Wake-Up Pins register offsets Defines

LL_PWR_WAKEUP_PINS_PULL_SHIFT_OFFSET

LL_PWR_WAKEUP_PINS_MAX_SHIFT_MASK

109 LL RCC Generic Driver

109.1 RCC Firmware driver registers structures

109.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32h5xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***
- ***uint32_t PCLK3_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK3_Frequency***
  PCLK3 clock frequency

109.1.2 LL_PLL_ClocksTypeDef

LL_PLL_ClocksTypeDef is defined in the stm32h5xx_ll_rcc.h

Data Fields

- ***uint32_t PLL_P_Frequency***
- ***uint32_t PLL_Q_Frequency***
- ***uint32_t PLL_R_Frequency***

Field Documentation

- ***uint32_t LL_PLL_ClocksTypeDef::PLL_P_Frequency***
- ***uint32_t LL_PLL_ClocksTypeDef::PLL_Q_Frequency***
- ***uint32_t LL_PLL_ClocksTypeDef::PLL_R_Frequency***

109.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

109.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)

Function description

Enable the HSE Clock Security System.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSECSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)

Function description

Enable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)

Function description

Disable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_HSE_Enable (void)

Function description

Enable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_HSE_Disable (void)

Function description

Disable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSE_SetExternalClockType

Function name

__STATIC_INLINE void LL_RCC_HSE_SetExternalClockType (uint32_t HSEClockMode)

Function description

Set external HSE clock type in Bypass mode.

Parameters

- **HSEClockMode:** This parameter can be one of the following values:
  - LL_RCC_HSE_ANALOG_TYPE
  - LL_RCC_HSE_DIGITAL_TYPE

Return values

- **None:**

Notes

- This bit can be written only if the HSE oscillator is disabled

Reference Manual to LL API cross reference:

- CR HSEEXT LL_RCC_HSE_SetExternalClockType

LL_RCC_HSE_GetExternalClockType

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetExternalClockType (void)

Function description

Get external HSE clock type in Bypass mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HSE_ANALOG_TYPE
  - LL_RCC_HSE_DIGITAL_TYPE

Reference Manual to LL API cross reference:

- CR HSEEXT LL_RCC_HSE_GetExternalClockType

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)

Function description

Check if HSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)

Function description

Enable HSI even in stop mode for some peripherals kernel.

Return values

- **None:**

Notes

- HSI oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)

Function description

Disable HSI in stop mode for some peripherals kernel.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsEnabledInStopMode (void)

Function description

Check if HSI is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_IsEnabledInStopMode

LL_RCC_HSI_IsDividerReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsDividerReady (void)

Function description

Check if HSI new divider applied and ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIDIVF LL_RCC_HSI_IsDividerReady

LL_RCC_HSI_SetDivider

Function name

__STATIC_INLINE void LL_RCC_HSI_SetDivider (uint32_t Divider)

Function description

Set HSI divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_SetDivider

LL_RCC_HSI_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetDivider (void)

Function description

Get HSI divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_GetDivider

LL_RCC_HSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)

Function description

Get HSI Calibration value.

Return values

- **A:** value between 0 and 4095 (0xFFF)

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- HSICFGR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)

Function description

Set HSI Calibration trimming.

Parameters

- **Value:** can be a value between Min_Data = 0 and Max_Data = 127 (0x7F)

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSICAL
- Default value is 64, which, when added to the HSICAL value, should trim the HSI to 64 MHz +/- 1 %

Reference Manual to LL API cross reference:

- HSICFGR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)

Function description

Get HSI Calibration trimming.

Return values

- **A:** value between Min_Data = 0 and Max_Data = 127 (0x7F)

Reference Manual to LL API cross reference:

- ICSC3R HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_CSI_Enable

Function name

__STATIC_INLINE void LL_RCC_CSI_Enable (void)

Function description

Enable CSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSION LL_RCC_CSI_Enable

LL_RCC_CSI_Disable

Function name

__STATIC_INLINE void LL_RCC_CSI_Disable (void)

Function description

Disable CSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSION LL_RCC_CSI_Disable

LL_RCC_CSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_IsReady (void)

Function description

Check if CSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CSIRDY LL_RCC_CSI_IsReady

LL_RCC_CSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_CSI_EnableInStopMode (void)

Function description

Enable CSI oscillator in Stop mode for some peripherals kernel clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSIKERON LL_RCC_CSI_EnableInStopMode

LL_RCC_CSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_CSI_DisableInStopMode (void)

Function description

Disable CSI oscillator in Stop mode for some peripherals kernel clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSIKERON LL_RCC_CSI_DisableInStopMode

LL_RCC_CSI_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_IsEnabledInStopMode (void)

Function description

Check if CSI is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CSIKERON LL_RCC_CSI_IsEnabledInStopMode

LL_RCC_CSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_GetCalibration (void)

Function description

Get CSI Calibration value.

Return values

- **A:** value between 0 and 255 (0xFF)

Notes

- When CSITRIM is written, CSICAL is updated with the sum of CSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- CSICFGR CSICAL LL_RCC_CSI_GetCalibration

LL_RCC_CSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_CSI_SetCalibTrimming (uint32_t Value)

Function description

Set CSI Calibration trimming.

Parameters

- **Value:** can be a value between Min_Data = 0 and Max_Data = 63 (0x3F)

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the CSICAL
- Default value is 16, which, when added to the CSICAL value, should trim the CSI to 4 MHz +/- 1 %

Reference Manual to LL API cross reference:

- CSICFGR CSITRIM LL_RCC_CSI_SetCalibTrimming

LL_RCC_CSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_GetCalibTrimming (void)

Function description

Get CSI Calibration trimming.

Return values

- **A:** value between 0 and 63 (0x3F)

Reference Manual to LL API cross reference:

- CSICFGR CSITRIM LL_RCC_CSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Enable (void)

Function description

Enable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Disable (void)

Function description

Disable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)

Function description

Check if HSI48 oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)

Function description

Get HSI48 Calibration value.

Return values

- **A:** value between 0 and 1023 (0x3FF)

Reference Manual to LL API cross reference:

- CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_LSE_Enable

Function name

__STATIC_INLINE void LL_RCC_LSE_Enable (void)

Function description

Enable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name

__STATIC_INLINE void LL_RCC_LSE_Disable (void)

Function description

Disable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)

Function description

Check if LSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)

Function description

Enable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)

Function description

Disable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetExternalClockType

Function name

__STATIC_INLINE void LL_RCC_LSE_SetExternalClockType (uint32_t LSEClockMode)

Function description

Set external LSE clock type in Bypass mode.

Parameters

- **LSEClockMode:** This parameter can be one of the following values:
  - LL_RCC_LSE_ANALOG_TYPE
  - LL_RCC_LSE_DIGITAL_TYPE (*)

Return values

- **None:** (*) not to be used if RTC is active

Notes

- This bit can be written only if the LSE oscillator is disabled

Reference Manual to LL API cross reference:

- BDCR LSEEXT LL_RCC_LSE_SetExternalClockType

LL_RCC_LSE_GetExternalClockType

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetExternalClockType (void)

Function description

Get external LSE clock type in Bypass mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSE_ANALOG_TYPE
  - LL_RCC_LSE_DIGITAL_TYPE

Reference Manual to LL API cross reference:

- BDCR LSEEXT LL_RCC_LSE_GetExternalClockType

LL_RCC_LSE_SetDriveCapability

Function name

__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)

Function description

Set LSE oscillator drive capability.

Parameters

- **LSEDrive:** This parameter can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Return values

- **None:**

Notes

- The oscillator is in Xtal mode when it is not in bypass mode.

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)

Function description

Get LSE oscillator drive capability.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)

Function description

Enable Clock security system on LSE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)

Function description

Disable Clock security system on LSE.

Return values

- **None:**

Notes

- Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsCSSDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void)

Function description

Check if CSS on LSE failure Detection.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSECSSD LL_RCC_LSE_IsCSSDetected

LL_RCC_LSI_Enable

Function name

__STATIC_INLINE void LL_RCC_LSI_Enable (void)

Function description

Enable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name

__STATIC_INLINE void LL_RCC_LSI_Disable (void)

Function description

Disable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)

Function description

Check if LSI is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_LSCO_Enable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Enable (void)

Function description

Enable Low Speed Microcontroller Clock Output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Enable

LL_RCC_LSCO_Disable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Disable (void)

Function description

Disable Low Speed Microcontroller Clock Output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Disable

LL_RCC_LSCO_SetSource

Function name

__STATIC_INLINE void LL_RCC_LSCO_SetSource (uint32_t Source)

Function description

Configure Low Speed Microcontroller Clock Output selection.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_LSCO_CLKSOURCE_LSI
  - LL_RCC_LSCO_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_SetSource

LL_RCC_LSCO_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_LSCO_GetSource (void)

Function description

Get Low Speed Microcontroller Clock Output selection.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSCO_CLKSOURCE_LSI
  - LL_RCC_LSCO_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_GetSource

LL_RCC_SetSysClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)

Function description

Configure the system clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_HSI
  - LL_RCC_SYS_CLKSOURCE_CSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_PLL1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)

Function description

Get the system clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_CSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

Reference Manual to LL API cross reference:

- CFGR1 SWS LL_RCC_GetSysClkSource

LL_RCC_SetAHBPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)

Function description

Set AHB prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetSystickClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSystickClockSource (uint32_t SystickSource)

Function description

Set Systick clock source.

Parameters

- **SystickSource:** This parameter can be one of the following values:
  - LL_RCC_SYSTICK_CLKSOURCE_LSI
  - LL_RCC_SYSTICK_CLKSOURCE_LSE
  - LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 SYSTICKSEL LL_RCC_SetSystickClockSource

LL_RCC_SetAPB1Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)

Function description

Set APB1 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)

Function description

Set APB2 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_SetAPB3Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB3Prescaler (uint32_t Prescaler)

Function description

Set APB3 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR3 PPRE3 LL_RCC_SetAPB3Prescaler

LL_RCC_GetAHBPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)

Function description

Get AHB prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetSystickClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSystickClockSource (void)

Function description

Get Sysctick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSTICK_CLKSOURCE_LSI
  - LL_RCC_SYSTICK_CLKSOURCE_LSE
  - LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8

Reference Manual to LL API cross reference:

- CCIPR4 SYSTICKSEL LL_RCC_SetSystickClockSource

LL_RCC_GetAPB1Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)

Function description

Get APB1 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)

Function description

Get APB2 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_GetAPB3Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB3Prescaler (void)

Function description

Get APB3 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Reference Manual to LL API cross reference:

- CFGR3 PPRE3 LL_RCC_GetAPB2Prescaler

LL_RCC_SetClkAfterWakeFromStop

Function name

__STATIC_INLINE void LL_RCC_SetClkAfterWakeFromStop (uint32_t Clock)

Function description

Set System Clock After Wake-Up From Stop mode.

Parameters

- **Clock:** This parameter can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_SetClkAfterWakeFromStop

LL_RCC_GetClkAfterWakeFromStop

Function name

__STATIC_INLINE uint32_t LL_RCC_GetClkAfterWakeFromStop (void)

Function description

Get System Clock After Wake-Up From Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_GetClkAfterWakeFromStop

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCO1 (pin PA8) or MCO2 (pin PC9)

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_LSE
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_PLL1Q
  - LL_RCC_MCO1SOURCE_HSI48
  - LL_RCC_MCO2SOURCE_SYSCLK
  - LL_RCC_MCO2SOURCE_PLL2P
  - LL_RCC_MCO2SOURCE_HSE
  - LL_RCC_MCO2SOURCE_PLL1P
  - LL_RCC_MCO2SOURCE_CSI
  - LL_RCC_MCO2SOURCE_LSI
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO1_DIV_1
  - LL_RCC_MCO1_DIV_2
  - LL_RCC_MCO1_DIV_3
  - LL_RCC_MCO1_DIV_4
  - LL_RCC_MCO1_DIV_5
  - LL_RCC_MCO1_DIV_6
  - LL_RCC_MCO1_DIV_7
  - LL_RCC_MCO1_DIV_8
  - LL_RCC_MCO1_DIV_9
  - LL_RCC_MCO1_DIV_10
  - LL_RCC_MCO1_DIV_11
  - LL_RCC_MCO1_DIV_12
  - LL_RCC_MCO1_DIV_13
  - LL_RCC_MCO1_DIV_14
  - LL_RCC_MCO1_DIV_15
  - LL_RCC_MCO2_DIV_1
  - LL_RCC_MCO2_DIV_2
  - LL_RCC_MCO2_DIV_3
  - LL_RCC_MCO2_DIV_4
  - LL_RCC_MCO2_DIV_5
  - LL_RCC_MCO2_DIV_6
  - LL_RCC_MCO2_DIV_7
  - LL_RCC_MCO2_DIV_8
  - LL_RCC_MCO2_DIV_9
  - LL_RCC_MCO2_DIV_10
  - LL_RCC_MCO2_DIV_11
  - LL_RCC_MCO2_DIV_12
  - LL_RCC_MCO2_DIV_13
  - LL_RCC_MCO2_DIV_14
  - LL_RCC_MCO2_DIV_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 MCO1 LL_RCC_ConfigMCO
- CFGR1 MCO1PRE LL_RCC_ConfigMCO
- CFGR1 MCO2 LL_RCC_ConfigMCO
- CFGR1 MCO2PRE LL_RCC_ConfigMCO

LL_RCC_SetClockSource

Function name

__STATIC_INLINE void LL_RCC_SetClockSource (uint32_t ClkSource)

Function description

Configure periph clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_PLL2Q
  - LL_RCC_USART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_CSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_PLL2Q
  - LL_RCC_USART2_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_CSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_PLL2Q
  - LL_RCC_USART3_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_CSI
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART6_CLKSOURCE_HSI (*)
  - LL_RCC_USART6_CLKSOURCE_CSI (*)
  - LL_RCC_USART6_CLKSOURCE_LSE (*)
  - LL_RCC_USART10_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART10_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART10_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART10_CLKSOURCE_HSI (*)
  - LL_RCC_USART10_CLKSOURCE_CSI (*)
  - LL_RCC_USART10_CLKSOURCE_LSE (*)
  - LL_RCC_USART11_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART11_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART11_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART11_CLKSOURCE_HSI (*)
  - LL_RCC_USART11_CLKSOURCE_CSI (*)
  - LL_RCC_USART11_CLKSOURCE_LSE (*)
  - LL_RCC_UART4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART4_CLKSOURCE_PLL2Q (*)
  - LL_RCC_UART4_CLKSOURCE_PLL3Q (*)
  - LL_RCC_UART4_CLKSOURCE_HSI (*)
  - LL_RCC_UART4_CLKSOURCE_CSI (*)
  - LL_RCC_UART4_CLKSOURCE_LSE (*)
  - LL_RCC_UART5_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART5_CLKSOURCE_PLL2Q (*)
  - LL_RCC_UART5_CLKSOURCE_PLL3Q (*)
  - LL_RCC_UART5_CLKSOURCE_HSI (*)
  - LL_RCC_UART5_CLKSOURCE_CSI (*)
  - LL_RCC_UART5_CLKSOURCE_LSE (*)

- LL_RCC_UART7_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART7_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART7_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART7_CLKSOURCE_HSI (*)
- LL_RCC_UART7_CLKSOURCE_CSI (*)
- LL_RCC_UART7_CLKSOURCE_LSE (*)
- LL_RCC_UART8_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART8_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART8_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART8_CLKSOURCE_HSI (*)
- LL_RCC_UART8_CLKSOURCE_CSI (*)
- LL_RCC_UART8_CLKSOURCE_LSE (*)
- LL_RCC_UART9_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART9_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART9_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART9_CLKSOURCE_HSI (*)
- LL_RCC_UART9_CLKSOURCE_CSI (*)
- LL_RCC_UART9_CLKSOURCE_LSE (*)
- LL_RCC_UART12_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART12_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART12_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART12_CLKSOURCE_HSI (*)
- LL_RCC_UART12_CLKSOURCE_CSI (*)
- LL_RCC_UART12_CLKSOURCE_LSE (*)
- LL_RCC_LPUART1_CLKSOURCE_PCLK3
- LL_RCC_LPUART1_CLKSOURCE_PLL2Q
- LL_RCC_LPUART1_CLKSOURCE_PLL3Q (*)
- LL_RCC_LPUART1_CLKSOURCE_HSI
- LL_RCC_LPUART1_CLKSOURCE_CSI
- LL_RCC_LPUART1_CLKSOURCE_LSE
- LL_RCC_I2C1_CLKSOURCE_PCLK1
- LL_RCC_I2C1_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C1_CLKSOURCE_PLL2R (*)
- LL_RCC_I2C1_CLKSOURCE_HSI
- LL_RCC_I2C1_CLKSOURCE_CSI
- LL_RCC_I2C2_CLKSOURCE_PCLK1
- LL_RCC_I2C2_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C2_CLKSOURCE_PLL2R (*)
- LL_RCC_I2C2_CLKSOURCE_HSI
- LL_RCC_I2C2_CLKSOURCE_CSI
- LL_RCC_I2C3_CLKSOURCE_PCLK3 (*)
- LL_RCC_I2C3_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C3_CLKSOURCE_HSI (*)
- LL_RCC_I2C3_CLKSOURCE_CSI (*)
- LL_RCC_I2C4_CLKSOURCE_PCLK3 (*)
- LL_RCC_I2C4_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C4_CLKSOURCE_HSI (*)
- LL_RCC_I2C4_CLKSOURCE_CSI (*)
- LL_RCC_I3C1_CLKSOURCE_PCLK1
- LL_RCC_I3C1_CLKSOURCE_PLL3R (*)

- LL_RCC_I3C1_CLKSOURCE_PLL2R (*)
- LL_RCC_I3C1_CLKSOURCE_HSI
- LL_RCC_I3C1_CLKSOURCE_NONE
- LL_RCC_I3C2_CLKSOURCE_PCLK3 (*)
- LL_RCC_I3C2_CLKSOURCE_PLL2R (*)
- LL_RCC_I3C2_CLKSOURCE_HSI (*)
- LL_RCC_I3C2_CLKSOURCE_NONE (*)
- LL_RCC_SPI1_CLKSOURCE_PLL1Q
- LL_RCC_SPI1_CLKSOURCE_PLL2P
- LL_RCC_SPI1_CLKSOURCE_PLL3P
- LL_RCC_SPI1_CLKSOURCE_PIN
- LL_RCC_SPI1_CLKSOURCE_CLKP
- LL_RCC_SPI2_CLKSOURCE_PLL1Q
- LL_RCC_SPI2_CLKSOURCE_PLL2P (*)
- LL_RCC_SPI2_CLKSOURCE_PLL3P (*)
- LL_RCC_SPI2_CLKSOURCE_PIN
- LL_RCC_SPI2_CLKSOURCE_CLKP
- LL_RCC_SPI3_CLKSOURCE_PLL1Q
- LL_RCC_SPI3_CLKSOURCE_PLL2P (*)
- LL_RCC_SPI3_CLKSOURCE_PLL3P (*)
- LL_RCC_SPI3_CLKSOURCE_PIN
- LL_RCC_SPI3_CLKSOURCE_CLKP
- LL_RCC_SPI4_CLKSOURCE_PCLK2 (*)
- LL_RCC_SPI4_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI4_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI4_CLKSOURCE_HSI (*)
- LL_RCC_SPI4_CLKSOURCE_CSI (*)
- LL_RCC_SPI4_CLKSOURCE_HSE (*)
- LL_RCC_SPI5_CLKSOURCE_PCLK3 (*)
- LL_RCC_SPI5_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI5_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI5_CLKSOURCE_HSI (*)
- LL_RCC_SPI5_CLKSOURCE_CSI (*)
- LL_RCC_SPI5_CLKSOURCE_HSE (*)
- LL_RCC_SPI6_CLKSOURCE_PCLK2 (*)
- LL_RCC_SPI6_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI6_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI6_CLKSOURCE_HSI (*)
- LL_RCC_SPI6_CLKSOURCE_CSI (*)
- LL_RCC_SPI6_CLKSOURCE_HSE (*)
- LL_RCC_LPTIM1_CLKSOURCE_PCLK3
- LL_RCC_LPTIM1_CLKSOURCE_PLL2P
- LL_RCC_LPTIM1_CLKSOURCE_PLL3R
- LL_RCC_LPTIM1_CLKSOURCE_LSE
- LL_RCC_LPTIM1_CLKSOURCE_LSI
- LL_RCC_LPTIM1_CLKSOURCE_CLKP
- LL_RCC_LPTIM2_CLKSOURCE_PCLK1
- LL_RCC_LPTIM2_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM2_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM2_CLKSOURCE_LSE

- LL_RCC_LPTIM2_CLKSOURCE_LSI
- LL_RCC_LPTIM2_CLKSOURCE_CLKP
- LL_RCC_LPTIM3_CLKSOURCE_PCLK3 (*)
- LL_RCC_LPTIM3_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM3_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM3_CLKSOURCE_LSE (*)
- LL_RCC_LPTIM3_CLKSOURCE_LSI (*)
- LL_RCC_LPTIM3_CLKSOURCE_CLKP (*)
- LL_RCC_LPTIM4_CLKSOURCE_PCLK3 (*)
- LL_RCC_LPTIM4_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM4_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM4_CLKSOURCE_LSE (*)
- LL_RCC_LPTIM4_CLKSOURCE_LSI (*)
- LL_RCC_LPTIM4_CLKSOURCE_CLKP (*)
- LL_RCC_LPTIM5_CLKSOURCE_PCLK3 (*)
- LL_RCC_LPTIM5_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM5_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM5_CLKSOURCE_LSE (*)
- LL_RCC_LPTIM5_CLKSOURCE_LSI (*)
- LL_RCC_LPTIM5_CLKSOURCE_CLKP (*)
- LL_RCC_LPTIM6_CLKSOURCE_PCLK3 (*)
- LL_RCC_LPTIM6_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM6_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM6_CLKSOURCE_LSE (*)
- LL_RCC_LPTIM6_CLKSOURCE_LSI (*)
- LL_RCC_LPTIM6_CLKSOURCE_CLKP (*)
- LL_RCC_SAI1_CLKSOURCE_PLL1Q (*)
- LL_RCC_SAI1_CLKSOURCE_PLL2P (*)
- LL_RCC_SAI1_CLKSOURCE_PLL3P (*)
- LL_RCC_SAI1_CLKSOURCE_PIN (*)
- LL_RCC_SAI1_CLKSOURCE_CLKP (*)
- LL_RCC_SAI2_CLKSOURCE_PLL1Q (*)
- LL_RCC_SAI2_CLKSOURCE_PLL2P (*)
- LL_RCC_SAI2_CLKSOURCE_PLL3P (*)
- LL_RCC_SAI2_CLKSOURCE_PIN (*)
- LL_RCC_SAI2_CLKSOURCE_CLKP (*)
- LL_RCC_SDMMC1_CLKSOURCE_PLL1Q (*)
- LL_RCC_SDMMC1_CLKSOURCE_PLL2R (*)
- LL_RCC_SDMMC2_CLKSOURCE_PLL1Q (*)
- LL_RCC_SDMMC2_CLKSOURCE_PLL2R (*)

(*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 * LL_RCC_SetClockSource
- CCIPR2 * LL_RCC_SetClockSource
- CCIPR3 * LL_RCC_SetClockSource
- CCIPR4 * LL_RCC_SetClockSource
- CCIPR5 * LL_RCC_SetClockSource

LL_RCC_SetUSARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTxSource)

Function description

Configure USARTx kernel clock source.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_PLL2Q
  - LL_RCC_USART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_CSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_PLL2Q
  - LL_RCC_USART2_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_CSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_PLL2Q
  - LL_RCC_USART3_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_CSI
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART6_CLKSOURCE_HSI (*)
  - LL_RCC_USART6_CLKSOURCE_CSI (*)
  - LL_RCC_USART6_CLKSOURCE_LSE (*)
  - LL_RCC_USART10_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART10_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART10_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART10_CLKSOURCE_HSI (*)
  - LL_RCC_USART10_CLKSOURCE_CSI (*)
  - LL_RCC_USART10_CLKSOURCE_LSE (*)
  - LL_RCC_USART11_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART11_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART11_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART11_CLKSOURCE_HSI (*)
  - LL_RCC_USART11_CLKSOURCE_CSI (*)
  - LL_RCC_USART11_CLKSOURCE_LSE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 USART1SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART2SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART3SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART6SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART10SEL LL_RCC_SetUSARTClockSource
- CCIPR2 USART11SEL LL_RCC_SetUSARTClockSource

LL_RCC_SetUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUARTClockSource (uint32_t UARTxSource)

Function description

Configure UARTx kernel clock source.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1
  - LL_RCC_UART4_CLKSOURCE_PLL2Q
  - LL_RCC_UART4_CLKSOURCE_PLL3Q
  - LL_RCC_UART4_CLKSOURCE_HSI
  - LL_RCC_UART4_CLKSOURCE_CSI
  - LL_RCC_UART4_CLKSOURCE_LSE
  - LL_RCC_UART5_CLKSOURCE_PCLK1
  - LL_RCC_UART5_CLKSOURCE_PLL2Q
  - LL_RCC_UART5_CLKSOURCE_PLL3Q
  - LL_RCC_UART5_CLKSOURCE_HSI
  - LL_RCC_UART5_CLKSOURCE_CSI
  - LL_RCC_UART5_CLKSOURCE_LSE
  - LL_RCC_UART7_CLKSOURCE_PCLK1
  - LL_RCC_UART7_CLKSOURCE_PLL2Q
  - LL_RCC_UART7_CLKSOURCE_PLL3Q
  - LL_RCC_UART7_CLKSOURCE_HSI
  - LL_RCC_UART7_CLKSOURCE_CSI
  - LL_RCC_UART7_CLKSOURCE_LSE
  - LL_RCC_UART8_CLKSOURCE_PCLK1
  - LL_RCC_UART8_CLKSOURCE_PLL2Q
  - LL_RCC_UART8_CLKSOURCE_PLL3Q
  - LL_RCC_UART8_CLKSOURCE_HSI
  - LL_RCC_UART8_CLKSOURCE_CSI
  - LL_RCC_UART8_CLKSOURCE_LSE
  - LL_RCC_UART9_CLKSOURCE_PCLK1
  - LL_RCC_UART9_CLKSOURCE_PLL2Q
  - LL_RCC_UART9_CLKSOURCE_PLL3Q
  - LL_RCC_UART9_CLKSOURCE_HSI
  - LL_RCC_UART9_CLKSOURCE_CSI
  - LL_RCC_UART9_CLKSOURCE_LSE
  - LL_RCC_UART12_CLKSOURCE_PCLK1
  - LL_RCC_UART12_CLKSOURCE_PLL2Q
  - LL_RCC_UART12_CLKSOURCE_PLL3Q
  - LL_RCC_UART12_CLKSOURCE_HSI
  - LL_RCC_UART12_CLKSOURCE_CSI
  - LL_RCC_UART12_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 UART4SEL LL_RCC_SetUARTClockSource
- CCIPR1 UART5SEL LL_RCC_SetUARTClockSource
- CCIPR1 UART7SEL LL_RCC_SetUARTClockSource
- CCIPR1 UART8SEL LL_RCC_SetUARTClockSource
- CCIPR1 UART9SEL LL_RCC_SetUARTClockSource
- CCIPR2 UART12SEL LL_RCC_SetUARTClockSource

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t LPUARTxSource)

Function description

Configure LPUARTx kernel clock source.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK3
  - LL_RCC_LPUART1_CLKSOURCE_PLL2Q
  - LL_RCC_LPUART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_CSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)

Function description

Configure I2Cx kernel clock source.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C1_CLKSOURCE_PLL2R (**)
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C1_CLKSOURCE_CSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C2_CLKSOURCE_PLL2R (**)
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_CSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK3 (*)
  - LL_RCC_I2C3_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C3_CLKSOURCE_HSI (*)
  - LL_RCC_I2C3_CLKSOURCE_CSI (*)
  - LL_RCC_I2C4_CLKSOURCE_PCLK3 (*)
  - LL_RCC_I2C4_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C4_CLKSOURCE_HSI (*)
  - LL_RCC_I2C4_CLKSOURCE_CSI (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only. (**) : For stm32h503xx family line only.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 I2C1SEL LL_RCC_SetI2CClockSource
- CCIPR4 I2C2SEL LL_RCC_SetI2CClockSource
- CCIPR4 I2C3SEL LL_RCC_SetI2CClockSource
- CCIPR4 I2C4SEL LL_RCC_SetI2CClockSource

LL_RCC_SetI3CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI3CClockSource (uint32_t I3CxSource)

Function description

Configure I3Cx kernel clock source.

Parameters

- **I3CxSource:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_PLL3R (*)
  - LL_RCC_I3C1_CLKSOURCE_PLL2R (**)
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C1_CLKSOURCE_NONE
  - LL_RCC_I3C2_CLKSOURCE_PCLK3 (**)
  - LL_RCC_I3C2_CLKSOURCE_PLL2R (**)
  - LL_RCC_I3C2_CLKSOURCE_HSI (**)
  - LL_RCC_I3C2_CLKSOURCE_NONE (**)

Return values

- **None:** (*) : For stm32h56xxx and stm32h57xxx family lines. (**) : For stm32h503xx family line.

Reference Manual to LL API cross reference:

- CCIPR4 I3C1SEL LL_RCC_SetI3CClockSource
- CCIPR4 I3C2SEL LL_RCC_SetI3CClockSource

LL_RCC_SetSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPIClockSource (uint32_t SPIxSource)

Function description

Configure SPIx kernel clock source.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PLL1Q
  - LL_RCC_SPI1_CLKSOURCE_PLL2P
  - LL_RCC_SPI1_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI1_CLKSOURCE_PIN
  - LL_RCC_SPI1_CLKSOURCE_CLKP
  - LL_RCC_SPI2_CLKSOURCE_PLL1Q
  - LL_RCC_SPI2_CLKSOURCE_PLL2P
  - LL_RCC_SPI2_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI2_CLKSOURCE_PIN
  - LL_RCC_SPI2_CLKSOURCE_CLKP
  - LL_RCC_SPI3_CLKSOURCE_PLL1Q
  - LL_RCC_SPI3_CLKSOURCE_PLL2P
  - LL_RCC_SPI3_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI3_CLKSOURCE_PIN
  - LL_RCC_SPI3_CLKSOURCE_CLKP
  - LL_RCC_SPI4_CLKSOURCE_PCLK2 (*)
  - LL_RCC_SPI4_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI4_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI4_CLKSOURCE_HSI (*)
  - LL_RCC_SPI4_CLKSOURCE_CSI (*)
  - LL_RCC_SPI4_CLKSOURCE_HSE (*)
  - LL_RCC_SPI5_CLKSOURCE_PCLK3 (*)
  - LL_RCC_SPI5_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI5_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI5_CLKSOURCE_HSI (*)
  - LL_RCC_SPI5_CLKSOURCE_CSI (*)
  - LL_RCC_SPI5_CLKSOURCE_HSE (*)
  - LL_RCC_SPI6_CLKSOURCE_PCLK2 (*)
  - LL_RCC_SPI6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI6_CLKSOURCE_HSI (*)
  - LL_RCC_SPI6_CLKSOURCE_CSI (*)
  - LL_RCC_SPI6_CLKSOURCE_HSE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 SPI1SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI2SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI3SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI4SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI5SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI6SEL LL_RCC_SetSPIClockSource

LL_RCC_SetLPTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)

Function description

Configure LPTIMx kernel clock source.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK3
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM3_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM3_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM4_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM5_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM6_CLKSOURCE_CLKP (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 LPTIM1SEL LL_RCC_SetLPTIMClockSource
- CCIPR2 LPTIM2SEL LL_RCC_SetLPTIMClockSource
- CCIPR2 LPTIM3SEL LL_RCC_SetLPTIMClockSource
- CCIPR2 LPTIM4SEL LL_RCC_SetLPTIMClockSource
- CCIPR2 LPTIM5SEL LL_RCC_SetLPTIMClockSource
- CCIPR2 LPTIM6SEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t FDCANxSource)

Function description

Configure FDCAN kernel clock source.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL1Q
  - LL_RCC_FDCAN_CLKSOURCE_PLL2Q

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 FDCANSEL LL_RCC_SetFDCANClockSource

LL_RCC_SetSAIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t SAIxSource)

Function description

Configure SAIx kernel clock source.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_PIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI2_CLKSOURCE_PLL1Q
  - LL_RCC_SAI2_CLKSOURCE_PLL2P
  - LL_RCC_SAI2_CLKSOURCE_PLL3P
  - LL_RCC_SAI2_CLKSOURCE_PIN
  - LL_RCC_SAI2_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 SAI1SEL LL_RCC_SetSAIClockSource
- CCIPR2 SAI2SEL LL_RCC_SetSAIClockSource

LL_RCC_SetSDMMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSDMMCClockSource (uint32_t SDMMCxSource)

Function description

Configure SDMMCx kernel clock source.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE_PLL1Q
  - LL_RCC_SDMMC1_CLKSOURCE_PLL2R
  - LL_RCC_SDMMC2_CLKSOURCE_PLL1Q
  - LL_RCC_SDMMC2_CLKSOURCE_PLL2R

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 SDMMC1SEL LL_RCC_SetSDMMCClockSource
- CCIPR4 SDMMC2SEL LL_RCC_SetSDMMCClockSource

LL_RCC_SetRNGClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)

Function description

Configure RNG kernel clock source.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL1Q
  - LL_RCC_RNG_CLKSOURCE_LSE
  - LL_RCC_RNG_CLKSOURCE_LSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 RNGSEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)

Function description

Configure USB clock source.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_NONE
  - LL_RCC_USB_CLKSOURCE_PLL1Q
  - LL_RCC_USB_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USB_CLKSOURCE_HSI48

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 USBSEL LL_RCC_SetUSBClockSource

LL_RCC_SetADCDACClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCDACClockSource (uint32_t ADCDACxSource)

Function description

Configure ADCx kernel clock source.

Parameters

- **ADCDACxSource:** This parameter can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE_HCLK
  - LL_RCC_ADCDAC_CLKSOURCE_SYSCLK
  - LL_RCC_ADCDAC_CLKSOURCE_PLL2R
  - LL_RCC_ADCDAC_CLKSOURCE_HSE
  - LL_RCC_ADCDAC_CLKSOURCE_HSI
  - LL_RCC_ADCDAC_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 ADCDACSEL LL_RCC_SetADCDACClockSource

LL_RCC_SetDACLPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetDACLPClockSource (uint32_t DACLPxSource)

Function description

Configure DAC low-power kernel clock source.

Parameters

- **DACLPxSource:** This parameter can be one of the following values:
  - LL_RCC_DAC_LP_CLKSOURCE_LSE
  - LL_RCC_DAC_LP_CLKSOURCE_LSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 DACSEL LL_RCC_SetDACLPClockSource

LL_RCC_SetCECClockSource

Function name

__STATIC_INLINE void LL_RCC_SetCECClockSource (uint32_t CECxSource)

Function description

Configure CECx kernel clock source.

Parameters

- **CECxSource:** This parameter can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE_LSE
  - LL_RCC_CEC_CLKSOURCE_LSI
  - LL_RCC_CEC_CLKSOURCE_CSI_DIV122

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 CECSEL LL_RCC_SetCECClockSource

LL_RCC_SetOCTOSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetOCTOSPIClockSource (uint32_t OCTOSPIxSource)

Function description

Configure OCTOSPIx kernel clock source.

Parameters

- **OCTOSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_OSPI_CLKSOURCE_HCLK
  - LL_RCC_OSPI_CLKSOURCE_PLL1Q
  - LL_RCC_OSPI_CLKSOURCE_PLL2R
  - LL_RCC_OSPI_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR4 OCTOSPISEL LL_RCC_SetOCTOSPIClockSource

LL_RCC_SetCLKPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetCLKPClockSource (uint32_t ClkSource)

Function description

Configure CLKP Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_CSI
  - LL_RCC_CLKP_CLKSOURCE_HSE
  - LL_RCC_CLKP_CLKSOURCE_NONE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR5 CKPERSEL LL_RCC_SetCLKPClockSource

LL_RCC_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetClockSource (uint32_t Periph)

Function description

Get periph clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE (*)
  - LL_RCC_USART10_CLKSOURCE (*)
  - LL_RCC_USART11_CLKSOURCE (*)
  - LL_RCC_UART4_CLKSOURCE (*)
  - LL_RCC_UART5_CLKSOURCE (*)
  - LL_RCC_UART7_CLKSOURCE (*)
  - LL_RCC_UART8_CLKSOURCE (*)
  - LL_RCC_UART9_CLKSOURCE (*)
  - LL_RCC_UART12_CLKSOURCE (*)
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE (*)
  - LL_RCC_SPI5_CLKSOURCE (*)
  - LL_RCC_SPI6_CLKSOURCE (*)
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE (*)
  - LL_RCC_I2C4_CLKSOURCE (*)
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE (*)
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE (*)
  - LL_RCC_LPTIM4_CLKSOURCE (*)
  - LL_RCC_LPTIM5_CLKSOURCE (*)
  - LL_RCC_LPTIM6_CLKSOURCE (*)
  - LL_RCC_SAI1_CLKSOURCE (*)
  - LL_RCC_SAI2_CLKSOURCE (*)
  - LL_RCC_SDMMC1_CLKSOURCE (*)
  - LL_RCC_SDMMC2_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_PLL2Q
  - LL_RCC_USART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_CSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_PLL2Q
  - LL_RCC_USART2_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_CSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_PLL2Q
  - LL_RCC_USART3_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_CSI
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART6_CLKSOURCE_HSI (*)
  - LL_RCC_USART6_CLKSOURCE_CSI (*)
  - LL_RCC_USART6_CLKSOURCE_LSE (*)
  - LL_RCC_USART10_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART10_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART10_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART10_CLKSOURCE_HSI (*)
  - LL_RCC_USART10_CLKSOURCE_CSI (*)
  - LL_RCC_USART10_CLKSOURCE_LSE (*)
  - LL_RCC_USART11_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART11_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART11_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART11_CLKSOURCE_HSI (*)
  - LL_RCC_USART11_CLKSOURCE_CSI (*)
  - LL_RCC_USART11_CLKSOURCE_LSE (*)
  - LL_RCC_UART4_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART4_CLKSOURCE_PLL2Q (*)
  - LL_RCC_UART4_CLKSOURCE_PLL3Q (*)
  - LL_RCC_UART4_CLKSOURCE_HSI (*)
  - LL_RCC_UART4_CLKSOURCE_CSI (*)
  - LL_RCC_UART4_CLKSOURCE_LSE (*)
  - LL_RCC_UART5_CLKSOURCE_PCLK1 (*)
  - LL_RCC_UART5_CLKSOURCE_PLL2Q (*)
  - LL_RCC_UART5_CLKSOURCE_PLL3Q (*)
  - LL_RCC_UART5_CLKSOURCE_HSI (*)
  - LL_RCC_UART5_CLKSOURCE_CSI (*)
  - LL_RCC_UART5_CLKSOURCE_LSE (*)

- LL_RCC_UART7_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART7_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART7_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART7_CLKSOURCE_HSI (*)
- LL_RCC_UART7_CLKSOURCE_CSI (*)
- LL_RCC_UART7_CLKSOURCE_LSE (*)
- LL_RCC_UART8_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART8_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART8_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART8_CLKSOURCE_HSI (*)
- LL_RCC_UART8_CLKSOURCE_CSI (*)
- LL_RCC_UART8_CLKSOURCE_LSE (*)
- LL_RCC_UART9_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART9_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART9_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART9_CLKSOURCE_HSI (*)
- LL_RCC_UART9_CLKSOURCE_CSI (*)
- LL_RCC_UART9_CLKSOURCE_LSE (*)
- LL_RCC_UART12_CLKSOURCE_PCLK1 (*)
- LL_RCC_UART12_CLKSOURCE_PLL2Q (*)
- LL_RCC_UART12_CLKSOURCE_PLL3Q (*)
- LL_RCC_UART12_CLKSOURCE_HSI (*)
- LL_RCC_UART12_CLKSOURCE_CSI (*)
- LL_RCC_UART12_CLKSOURCE_LSE (*)
- LL_RCC_LPUART1_CLKSOURCE_PCLK3
- LL_RCC_LPUART1_CLKSOURCE_PLL2Q
- LL_RCC_LPUART1_CLKSOURCE_PLL3Q (*)
- LL_RCC_LPUART1_CLKSOURCE_HSI
- LL_RCC_LPUART1_CLKSOURCE_CSI
- LL_RCC_LPUART1_CLKSOURCE_LSE
- LL_RCC_I2C1_CLKSOURCE_PCLK1
- LL_RCC_I2C1_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C1_CLKSOURCE_PLL2R (*)
- LL_RCC_I2C1_CLKSOURCE_HSI
- LL_RCC_I2C1_CLKSOURCE_CSI
- LL_RCC_I2C2_CLKSOURCE_PCLK1
- LL_RCC_I2C2_CLKSOURCE_PLL3R
- LL_RCC_I2C2_CLKSOURCE_PLL2R
- LL_RCC_I2C2_CLKSOURCE_HSI
- LL_RCC_I2C2_CLKSOURCE_CSI
- LL_RCC_I2C3_CLKSOURCE_PCLK3 (*)
- LL_RCC_I2C3_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C3_CLKSOURCE_HSI (*)
- LL_RCC_I2C3_CLKSOURCE_CSI (*)
- LL_RCC_I2C4_CLKSOURCE_PCLK3 (*)
- LL_RCC_I2C4_CLKSOURCE_PLL3R (*)
- LL_RCC_I2C4_CLKSOURCE_HSI (*)
- LL_RCC_I2C4_CLKSOURCE_CSI (*)
- LL_RCC_I3C1_CLKSOURCE_PCLK1
- LL_RCC_I3C1_CLKSOURCE_PLL3R (*)

- LL_RCC_I3C1_CLKSOURCE_PLL2R (*)
- LL_RCC_I3C1_CLKSOURCE_HSI
- LL_RCC_I3C1_CLKSOURCE_NONE
- LL_RCC_I3C2_CLKSOURCE_PCLK3 (*)
- LL_RCC_I3C2_CLKSOURCE_PLL2R (*)
- LL_RCC_I3C2_CLKSOURCE_HSI (*)
- LL_RCC_I3C2_CLKSOURCE_NONE (*)
- LL_RCC_SPI1_CLKSOURCE_PLL1Q
- LL_RCC_SPI1_CLKSOURCE_PLL2P
- LL_RCC_SPI1_CLKSOURCE_PLL3P
- LL_RCC_SPI1_CLKSOURCE_PIN
- LL_RCC_SPI1_CLKSOURCE_CLKP
- LL_RCC_SPI2_CLKSOURCE_PLL1Q
- LL_RCC_SPI2_CLKSOURCE_PLL2P (*)
- LL_RCC_SPI2_CLKSOURCE_PLL3P (*)
- LL_RCC_SPI2_CLKSOURCE_PIN
- LL_RCC_SPI2_CLKSOURCE_CLKP
- LL_RCC_SPI3_CLKSOURCE_PLL1Q
- LL_RCC_SPI3_CLKSOURCE_PLL2P (*)
- LL_RCC_SPI3_CLKSOURCE_PLL3P (*)
- LL_RCC_SPI3_CLKSOURCE_PIN
- LL_RCC_SPI3_CLKSOURCE_CLKP
- LL_RCC_SPI4_CLKSOURCE_PCLK2 (*)
- LL_RCC_SPI4_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI4_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI4_CLKSOURCE_HSI (*)
- LL_RCC_SPI4_CLKSOURCE_CSI (*)
- LL_RCC_SPI4_CLKSOURCE_HSE (*)
- LL_RCC_SPI5_CLKSOURCE_PCLK3 (*)
- LL_RCC_SPI5_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI5_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI5_CLKSOURCE_HSI (*)
- LL_RCC_SPI5_CLKSOURCE_CSI (*)
- LL_RCC_SPI5_CLKSOURCE_HSE (*)
- LL_RCC_SPI6_CLKSOURCE_PCLK2 (*)
- LL_RCC_SPI6_CLKSOURCE_PLL2Q (*)
- LL_RCC_SPI6_CLKSOURCE_PLL3Q (*)
- LL_RCC_SPI6_CLKSOURCE_HSI (*)
- LL_RCC_SPI6_CLKSOURCE_CSI (*)
- LL_RCC_SPI6_CLKSOURCE_HSE (*)
- LL_RCC_LPTIM1_CLKSOURCE_PCLK3
- LL_RCC_LPTIM1_CLKSOURCE_PLL2P
- LL_RCC_LPTIM1_CLKSOURCE_PLL3R
- LL_RCC_LPTIM1_CLKSOURCE_LSE
- LL_RCC_LPTIM1_CLKSOURCE_LSI
- LL_RCC_LPTIM1_CLKSOURCE_CLKP
- LL_RCC_LPTIM2_CLKSOURCE_PCLK1
- LL_RCC_LPTIM2_CLKSOURCE_PLL2P (*)
- LL_RCC_LPTIM2_CLKSOURCE_PLL3R (*)
- LL_RCC_LPTIM2_CLKSOURCE_LSE

  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM3_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM3_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM4_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM5_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM6_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI1_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI1_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI1_CLKSOURCE_PIN (*)
  - LL_RCC_SAI1_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI2_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI2_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI2_CLKSOURCE_PIN (*)
  - LL_RCC_SAI2_CLKSOURCE_CLKP (*)
  - LL_RCC_SDMMC1_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SDMMC1_CLKSOURCE_PLL2R (*)
  - LL_RCC_SDMMC2_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SDMMC2_CLKSOURCE_PLL2R (*)

  (*) value not defined in all devices.
- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 * LL_RCC_GetClockSource
- CCIPR2 * LL_RCC_GetClockSource
- CCIPR3 * LL_RCC_GetClockSource
- CCIPR4 * LL_RCC_GetClockSource
- CCIPR5 * LL_RCC_GetClockSource

LL_RCC_GetUSARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)

Function description

Get USARTx kernel clock source.

Parameters

- **USARTx:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE (*)
  - LL_RCC_USART10_CLKSOURCE (*)
  - LL_RCC_USART11_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK2
  - LL_RCC_USART1_CLKSOURCE_PLL2Q
  - LL_RCC_USART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART1_CLKSOURCE_HSI
  - LL_RCC_USART1_CLKSOURCE_CSI
  - LL_RCC_USART1_CLKSOURCE_LSE
  - LL_RCC_USART2_CLKSOURCE_PCLK1
  - LL_RCC_USART2_CLKSOURCE_PLL2Q
  - LL_RCC_USART2_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART2_CLKSOURCE_HSI
  - LL_RCC_USART2_CLKSOURCE_CSI
  - LL_RCC_USART2_CLKSOURCE_LSE
  - LL_RCC_USART3_CLKSOURCE_PCLK1
  - LL_RCC_USART3_CLKSOURCE_PLL2Q
  - LL_RCC_USART3_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART3_CLKSOURCE_HSI
  - LL_RCC_USART3_CLKSOURCE_CSI
  - LL_RCC_USART3_CLKSOURCE_LSE
  - LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART6_CLKSOURCE_HSI (*)
  - LL_RCC_USART6_CLKSOURCE_CSI (*)
  - LL_RCC_USART6_CLKSOURCE_LSE (*)
  - LL_RCC_USART10_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART10_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART10_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART10_CLKSOURCE_HSI (*)
  - LL_RCC_USART10_CLKSOURCE_CSI (*)
  - LL_RCC_USART10_CLKSOURCE_LSE (*)
  - LL_RCC_USART11_CLKSOURCE_PCLK1 (*)
  - LL_RCC_USART11_CLKSOURCE_PLL2Q (*)
  - LL_RCC_USART11_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USART11_CLKSOURCE_HSI (*)
  - LL_RCC_USART11_CLKSOURCE_CSI (*)
  - LL_RCC_USART11_CLKSOURCE_LSE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Reference Manual to LL API cross reference:

- CCIPR1 USART1SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART2SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART3SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART6SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART10SEL LL_RCC_GetUSARTClockSource
- CCIPR2 USART11SEL LL_RCC_GetUSARTClockSource

LL_RCC_GetUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUARTClockSource (uint32_t UARTx)

Function description

Get UARTx kernel clock source.

Parameters

- **UARTx:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE
  - LL_RCC_UART7_CLKSOURCE
  - LL_RCC_UART8_CLKSOURCE
  - LL_RCC_UART9_CLKSOURCE
  - LL_RCC_UART12_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE_PCLK1
  - LL_RCC_UART4_CLKSOURCE_PLL2Q
  - LL_RCC_UART4_CLKSOURCE_PLL3Q
  - LL_RCC_UART4_CLKSOURCE_HSI
  - LL_RCC_UART4_CLKSOURCE_CSI
  - LL_RCC_UART4_CLKSOURCE_LSE
  - LL_RCC_UART5_CLKSOURCE_PCLK1
  - LL_RCC_UART5_CLKSOURCE_PLL2Q
  - LL_RCC_UART5_CLKSOURCE_PLL3Q
  - LL_RCC_UART5_CLKSOURCE_HSI
  - LL_RCC_UART5_CLKSOURCE_CSI
  - LL_RCC_UART5_CLKSOURCE_LSE
  - LL_RCC_UART7_CLKSOURCE_PCLK1
  - LL_RCC_UART7_CLKSOURCE_PLL2Q
  - LL_RCC_UART7_CLKSOURCE_PLL3Q
  - LL_RCC_UART7_CLKSOURCE_HSI
  - LL_RCC_UART7_CLKSOURCE_CSI
  - LL_RCC_UART7_CLKSOURCE_LSE
  - LL_RCC_UART8_CLKSOURCE_PCLK1
  - LL_RCC_UART8_CLKSOURCE_PLL2Q
  - LL_RCC_UART8_CLKSOURCE_PLL3Q
  - LL_RCC_UART8_CLKSOURCE_HSI
  - LL_RCC_UART8_CLKSOURCE_CSI
  - LL_RCC_UART8_CLKSOURCE_LSE
  - LL_RCC_UART9_CLKSOURCE_PCLK1
  - LL_RCC_UART9_CLKSOURCE_PLL2Q
  - LL_RCC_UART9_CLKSOURCE_PLL3Q
  - LL_RCC_UART9_CLKSOURCE_HSI
  - LL_RCC_UART9_CLKSOURCE_CSI
  - LL_RCC_UART9_CLKSOURCE_LSE
  - LL_RCC_UART12_CLKSOURCE_PCLK1
  - LL_RCC_UART12_CLKSOURCE_PLL2Q
  - LL_RCC_UART12_CLKSOURCE_PLL3Q
  - LL_RCC_UART12_CLKSOURCE_HSI
  - LL_RCC_UART12_CLKSOURCE_CSI
  - LL_RCC_UART12_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR1 UART4SEL LL_RCC_GetUARTClockSource
- CCIPR1 UART5SEL LL_RCC_GetUARTClockSource
- CCIPR1 UART7SEL LL_RCC_GetUARTClockSource
- CCIPR1 UART8SEL LL_RCC_GetUARTClockSource
- CCIPR1 UART9SEL LL_RCC_GetUARTClockSource
- CCIPR2 UART12SEL LL_RCC_GetUARTClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)

Function description

Get LPUARTx kernel clock source.

Parameters

- **LPUARTx:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK3
  - LL_RCC_LPUART1_CLKSOURCE_PLL2Q
  - LL_RCC_LPUART1_CLKSOURCE_PLL3Q (*)
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_CSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Reference Manual to LL API cross reference:

- CCIPR3 LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)

Function description

Get I2Cx kernel clock source.

Parameters

- **I2Cx:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE (*)
  - LL_RCC_I2C4_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C1_CLKSOURCE_PLL2R (**)
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C1_CLKSOURCE_CSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C2_CLKSOURCE_PLL2R (**)
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_CSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK3 (*)
  - LL_RCC_I2C3_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C3_CLKSOURCE_HSI (*)
  - LL_RCC_I2C3_CLKSOURCE_CSI (*)
  - LL_RCC_I2C4_CLKSOURCE_PCLK3 (*)
  - LL_RCC_I2C4_CLKSOURCE_PLL3R (*)
  - LL_RCC_I2C4_CLKSOURCE_HSI (*)
  - LL_RCC_I2C4_CLKSOURCE_CSI (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only. (**) : For stm32h503xx family line only.

Reference Manual to LL API cross reference:

- CCIPR4 I2C1SEL LL_RCC_GetI2CClockSource
- CCIPR4 I2C2SEL LL_RCC_GetI2CClockSource
- CCIPR4 I2C3SEL LL_RCC_GetI2CClockSource
- CCIPR4 I2C4SEL LL_RCC_GetI2CClockSource

LL_RCC_GetI3CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI3CClockSource (uint32_t I3Cx)

Function description

Get I3Cx kernel clock source.

Parameters

- **I3Cx:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE (**)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE_PCLK1
  - LL_RCC_I3C1_CLKSOURCE_PLL3R (*)
  - LL_RCC_I3C1_CLKSOURCE_PLL2R (**)
  - LL_RCC_I3C1_CLKSOURCE_HSI
  - LL_RCC_I3C1_CLKSOURCE_NONE
  - LL_RCC_I3C2_CLKSOURCE_PCLK3 (**)
  - LL_RCC_I3C2_CLKSOURCE_PLL2R (**)
  - LL_RCC_I3C2_CLKSOURCE_HSI (**)
  - LL_RCC_I3C2_CLKSOURCE_NONE (**)

  (*) : For stm32h56xxx and stm32h57xxx family lines. (**) : For stm32h503xx family line.

Reference Manual to LL API cross reference:

- CCIPR4 I3C1SEL LL_RCC_GetI3CClockSource
- CCIPR4 I3C2SEL LL_RCC_GetI3CClockSource

LL_RCC_GetSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPIClockSource (uint32_t SPIx)

Function description

Get SPIx kernel clock source.

Parameters

- **SPIx:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE (*)
  - LL_RCC_SPI5_CLKSOURCE (*)
  - LL_RCC_SPI6_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PLL1Q
  - LL_RCC_SPI1_CLKSOURCE_PLL2P
  - LL_RCC_SPI1_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI1_CLKSOURCE_PIN
  - LL_RCC_SPI1_CLKSOURCE_CLKP
  - LL_RCC_SPI2_CLKSOURCE_PLL1Q
  - LL_RCC_SPI2_CLKSOURCE_PLL2P
  - LL_RCC_SPI2_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI2_CLKSOURCE_PIN
  - LL_RCC_SPI2_CLKSOURCE_CLKP
  - LL_RCC_SPI3_CLKSOURCE_PLL1Q
  - LL_RCC_SPI3_CLKSOURCE_PLL2P
  - LL_RCC_SPI3_CLKSOURCE_PLL3P (*)
  - LL_RCC_SPI3_CLKSOURCE_PIN
  - LL_RCC_SPI3_CLKSOURCE_CLKP
  - LL_RCC_SPI4_CLKSOURCE_PCLK2 (*)
  - LL_RCC_SPI4_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI4_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI4_CLKSOURCE_HSI (*)
  - LL_RCC_SPI4_CLKSOURCE_CSI (*)
  - LL_RCC_SPI4_CLKSOURCE_HSE (*)
  - LL_RCC_SPI5_CLKSOURCE_PCLK3 (*)
  - LL_RCC_SPI5_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI5_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI5_CLKSOURCE_HSI (*)
  - LL_RCC_SPI5_CLKSOURCE_CSI (*)
  - LL_RCC_SPI5_CLKSOURCE_HSE (*)
  - LL_RCC_SPI6_CLKSOURCE_PCLK2 (*)
  - LL_RCC_SPI6_CLKSOURCE_PLL2Q (*)
  - LL_RCC_SPI6_CLKSOURCE_PLL3Q (*)
  - LL_RCC_SPI6_CLKSOURCE_HSI (*)
  - LL_RCC_SPI6_CLKSOURCE_CSI (*)
  - LL_RCC_SPI6_CLKSOURCE_HSE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Reference Manual to LL API cross reference:

- CCIPR3 SPI1SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI2SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI3SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI4SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI5SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI6SEL LL_RCC_GetSPIClockSource

LL_RCC_GetLPTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)

Function description

Get LPTIMx kernel clock source.

Parameters

- **LPTIMx:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE (*)
  - LL_RCC_LPTIM4_CLKSOURCE (*)
  - LL_RCC_LPTIM5_CLKSOURCE (*)
  - LL_RCC_LPTIM6_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK3
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM3_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM3_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM3_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM3_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM4_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM4_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM4_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM5_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM5_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM5_CLKSOURCE_CLKP (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PCLK3 (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL2P (*)
  - LL_RCC_LPTIM6_CLKSOURCE_PLL3R (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSE (*)
  - LL_RCC_LPTIM6_CLKSOURCE_LSI (*)
  - LL_RCC_LPTIM6_CLKSOURCE_CLKP (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Reference Manual to LL API cross reference:

- CCIPR2 LPTIM1SEL LL_RCC_GetLPTIMClockSource
- CCIPR2 LPTIM2SEL LL_RCC_GetLPTIMClockSource
- CCIPR2 LPTIM3SEL LL_RCC_GetLPTIMClockSource
- CCIPR2 LPTIM4SEL LL_RCC_GetLPTIMClockSource
- CCIPR2 LPTIM5SEL LL_RCC_GetLPTIMClockSource
- CCIPR2 LPTIM6SEL LL_RCC_GetLPTIMClockSource

LL_RCC_TIMIC_Enable

Function name

__STATIC_INLINE void LL_RCC_TIMIC_Enable (void)

Function description

Enable TIM2,15 and LPTIM2 Input capture clock source.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 TIMICSEL LL_RCC_TIMIC_Enable

LL_RCC_TIMIC_Disable

Function name

__STATIC_INLINE void LL_RCC_TIMIC_Disable (void)

Function description

Disable TIM2,15 and LPTIM2 Input capture clock source.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 TIMICSEL LL_RCC_TIMIC_Disable

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t FDCANx)

Function description

Get FDCAN kernel clock source.

Parameters

- **FDCANx:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL1Q
  - LL_RCC_FDCAN_CLKSOURCE_PLL2Q

Reference Manual to LL API cross reference:

- CCIPR5 FDCANSEL LL_RCC_GetFDCANClockSource

LL_RCC_GetSAIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t SAIx)

Function description

Get SAIx kernel clock source.

Parameters

- **SAIx:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_PIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI2_CLKSOURCE_PLL1Q
  - LL_RCC_SAI2_CLKSOURCE_PLL2P
  - LL_RCC_SAI2_CLKSOURCE_PLL3P
  - LL_RCC_SAI2_CLKSOURCE_PIN
  - LL_RCC_SAI2_CLKSOURCE_CLKP

Reference Manual to LL API cross reference:

- CCIPR2 SAI1SEL LL_RCC_GetSAIClockSource
- CCIPR2 SAI2SEL LL_RCC_GetSAIClockSource

LL_RCC_GetSDMMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSDMMCClockSource (uint32_t SDMMCx)

Function description

Get SDMMCx kernel clock source.

Parameters

- **SDMMCx:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE
  - LL_RCC_SDMMC2_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE_PLL1Q
  - LL_RCC_SDMMC1_CLKSOURCE_PLL2R
  - LL_RCC_SDMMC2_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SDMMC2_CLKSOURCE_PLL2R (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- CCIPR4 SDMMC1SEL LL_RCC_GetSDMMCClockSource CCIPR4 SDMMC2SEL LL_RCC_GetSDMMCClockSource

LL_RCC_GetRNGClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)

Function description

Get RNGx kernel clock source.

Parameters

- **RNGx:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL1Q
  - LL_RCC_RNG_CLKSOURCE_LSE
  - LL_RCC_RNG_CLKSOURCE_LSI

Reference Manual to LL API cross reference:

- CCIPR5 RNGSEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)

Function description

Get USB clock source.

Parameters

- **USBx:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_NONE
  - LL_RCC_USB_CLKSOURCE_PLL1Q
  - LL_RCC_USB_CLKSOURCE_PLL3Q (*)
  - LL_RCC_USB_CLKSOURCE_HSI48

  (*) : For stm32h56xxx and stm32h57xxx family lines.

Reference Manual to LL API cross reference:

- CCIPR4 USBSEL LL_RCC_GetUSBClockSource

LL_RCC_GetADCDACClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCDACClockSource (uint32_t ADCDACx)

Function description

Get ADCDACx kernel clock source.

Parameters

- **ADCDACx:** This parameter can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE_HCLK
  - LL_RCC_ADCDAC_CLKSOURCE_SYSCLK
  - LL_RCC_ADCDAC_CLKSOURCE_PLL2R
  - LL_RCC_ADCDAC_CLKSOURCE_HSE
  - LL_RCC_ADCDAC_CLKSOURCE_HSI
  - LL_RCC_ADCDAC_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- CCIPR5 ADCDACSEL LL_RCC_GetADCDACClockSource

LL_RCC_GetDACLPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDACLPClockSource (uint32_t DACLPx)

Function description

Get DAC low-power kernel Clock Source.

Parameters

- **DACLPx:** This parameter can be one of the following values:
  - LL_RCC_DAC_LP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_DAC_LP_CLKSOURCE_LSE
  - LL_RCC_DAC_LP_CLKSOURCE_LSI

Reference Manual to LL API cross reference:

- CCIPR5 DACSEL LL_RCC_GetDACLPClockSource

LL_RCC_GetCECClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCECClockSource (uint32_t CECx)

Function description

Get CECx kernel clock source.

Parameters

- **CECx:** This parameter can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE_LSE
  - LL_RCC_CEC_CLKSOURCE_LSI
  - LL_RCC_CEC_CLKSOURCE_CSI_DIV122

Reference Manual to LL API cross reference:

- CCIPR5 CECSEL LL_RCC_GetCECClockSource

LL_RCC_GetOCTOSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetOCTOSPIClockSource (uint32_t OCTOSPIx)

Function description

Get OCTOSPI kernel clock source.

Parameters

- **OCTOSPIx:** This parameter can be one of the following values:
  - LL_RCC_OCTOSPI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_OSPI_CLKSOURCE_HCLK
  - LL_RCC_OSPI_CLKSOURCE_PLL1Q
  - LL_RCC_OSPI_CLKSOURCE_PLL2R
  - LL_RCC_OSPI_CLKSOURCE_CLKP

Reference Manual to LL API cross reference:

- CCIPR4 OCTOSPISEL LL_RCC_GetOCTOSPIClockSource

LL_RCC_GetCLKPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCLKPClockSource (uint32_t CLKPx)

Function description

Get CLKP kernel clock source.

Parameters

- **CLKPx:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_CSI
  - LL_RCC_CLKP_CLKSOURCE_HSE
  - LL_RCC_CLKP_CLKSOURCE_NONE

Reference Manual to LL API cross reference:

- CCIPR5 CKPERSEL LL_RCC_GetCLKPClockSource

LL_RCC_SetKerWakeUpClkSource

Function name

__STATIC_INLINE void LL_RCC_SetKerWakeUpClkSource (uint32_t Source)

Function description

Configure the Kernel wakeup clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_KERWAKEUP_CLKSOURCE_HSI
  - LL_RCC_KERWAKEUP_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 STOPKERWUCK LL_RCC_SetKerWakeUpClkSource

LL_RCC_GetKerWakeUpClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetKerWakeUpClkSource (void)

Function description

Get the Kernel wakeup clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_KERWAKEUP_CLKSOURCE_HSI
  - LL_RCC_KERWAKEUP_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- CFGR1 STOPKERWUCK LL_RCC_GetKerWakeUpClkSource

LL_RCC_SetRTCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)

Function description

Set RTC Clock Source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV

Return values

- **None:**

Notes

- Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)

Function description

Get RTC Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name

__STATIC_INLINE void LL_RCC_EnableRTC (void)

Function description

Enable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name

__STATIC_INLINE void LL_RCC_DisableRTC (void)

Function description

Disable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)

Function description

Check if RTC has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)

Function description

Force the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)

Function description

Release the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_SetRTC_HSEPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Prescaler)

Function description

Set HSE Prescalers for RTC Clock.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_RTC_HSE_NOCLOCK
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48

- LL_RCC_RTC_HSE_DIV_49
- LL_RCC_RTC_HSE_DIV_50
- LL_RCC_RTC_HSE_DIV_51
- LL_RCC_RTC_HSE_DIV_52
- LL_RCC_RTC_HSE_DIV_53
- LL_RCC_RTC_HSE_DIV_54
- LL_RCC_RTC_HSE_DIV_55
- LL_RCC_RTC_HSE_DIV_56
- LL_RCC_RTC_HSE_DIV_57
- LL_RCC_RTC_HSE_DIV_58
- LL_RCC_RTC_HSE_DIV_59
- LL_RCC_RTC_HSE_DIV_60
- LL_RCC_RTC_HSE_DIV_61
- LL_RCC_RTC_HSE_DIV_62
- LL_RCC_RTC_HSE_DIV_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 RTCPRE LL_RCC_SetRTC_HSEPrescaler

LL_RCC_GetRTC_HSEPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTC_HSEPrescaler (void)

Function description

Get HSE Prescalers for RTC Clock.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_HSE_NOCLOCK
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48

- LL_RCC_RTC_HSE_DIV_49
- LL_RCC_RTC_HSE_DIV_50
- LL_RCC_RTC_HSE_DIV_51
- LL_RCC_RTC_HSE_DIV_52
- LL_RCC_RTC_HSE_DIV_53
- LL_RCC_RTC_HSE_DIV_54
- LL_RCC_RTC_HSE_DIV_55
- LL_RCC_RTC_HSE_DIV_56
- LL_RCC_RTC_HSE_DIV_57
- LL_RCC_RTC_HSE_DIV_58
- LL_RCC_RTC_HSE_DIV_59
- LL_RCC_RTC_HSE_DIV_60
- LL_RCC_RTC_HSE_DIV_61
- LL_RCC_RTC_HSE_DIV_62
- LL_RCC_RTC_HSE_DIV_63

Reference Manual to LL API cross reference:

- CFGR1 RTCPRE LL_RCC_GetRTC_HSEPrescaler

LL_RCC_SetTIMPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetTIMPrescaler (uint32_t Prescaler)

Function description

Set Timers Clock Prescalers.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_TIM_PRESCALER_TWICE
  - LL_RCC_TIM_PRESCALER_FOUR_TIMES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 TIMPRE LL_RCC_SetTIMPrescaler

LL_RCC_GetTIMPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetTIMPrescaler (void)

Function description

Get Timers Clock Prescalers.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_TIM_PRESCALER_TWICE
  - LL_RCC_TIM_PRESCALER_FOUR_TIMES

Reference Manual to LL API cross reference:

- CFGR1 TIMPRE LL_RCC_GetTIMPrescaler

LL_RCC_PLL1_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Enable (void)

Function description

Enable PLL1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL1ON LL_RCC_PLL1_Enable

LL_RCC_PLL1_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Disable (void)

Function description

Disable PLL1.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLLON LL_RCC_PLL1_Disable

LL_RCC_PLL1_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsReady (void)

Function description

Check if PLL1 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL1RDY LL_RCC_PLL1_IsReady

LL_RCC_PLL1P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Enable (void)

Function description

Enable PLL1 P output mapped to SYSCLK.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1PEN LL_RCC_PLL1P_Enable

LL_RCC_PLL1P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Disable (void)

Function description

Disable PLL1 P output mapped to SYSCLK.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1PEN LL_RCC_PLL1P_Disable

LL_RCC_PLL1Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1Q_Enable (void)

Function description

Enable PLL1 Q output.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1QEN LL_RCC_PLL1Q_Enable

LL_RCC_PLL1Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1Q_Disable (void)

Function description

Disable PLL1 Q output.

Return values

- **None:**

Notes

- In order to save power, when the PLL1 Q output of the PLL1 is not used, PLL1Q should be 0

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1QEN LL_RCC_PLL1Q_Disable

LL_RCC_PLL1R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1R_Enable (void)

Function description

Enable PLL1 R output.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1REN LL_RCC_PLL1R_Enable

LL_RCC_PLL1R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1R_Disable (void)

Function description

Disable PLL1 R output.

Return values

- **None:**

Notes

- In order to save power, when the PLL1 R output of the PLL1 is not used, PLL1R should be 0

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1REN LL_RCC_PLL1R_Disable

LL_RCC_PLL1P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1P_IsEnabled (void)

Function description

Check if PLL1 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVP1EN LL_RCC_PLL1P_IsEnabled

LL_RCC_PLL1Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1Q_IsEnabled (void)

Function description

Check if PLL1 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ1EN LL_RCC_PLL1Q_IsEnabled

LL_RCC_PLL1R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1R_IsEnabled (void)

Function description

Check if PLL1 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVR1EN LL_RCC_PLL1R_IsEnabled

LL_RCC_PLL1_ConfigDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL1_ConfigDomain_SYS (uint32_t Source, uint32_t PLL1M, uint32_t PLL1N, uint32_t PLL1P)

Function description

Configure PLL1 used for SYSCLK.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL1SOURCE_NONE
  - LL_RCC_PLL1SOURCE_HSI
  - LL_RCC_PLL1SOURCE_CSI
  - LL_RCC_PLL1SOURCE_HSE
- **PLL1M:** parameter can be a value between 1 and 63
- **PLL1P:** parameter can be a value between 1 and 128 (odd values not allowed)
- **PLL1N:** parameter can be a value between 4 and 512

Return values

- **None:**

Notes

- PLL1 Source, PLL1M, PLL1N and PLL1P can be written only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1M LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1N LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1R LL_RCC_PLL1_ConfigDomain_SYS

LL_RCC_PLL1_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetSource (uint32_t PLL1Source)

Function description

Configure PLL clock source.

Parameters

- **PLL1Source:** This parameter can be one of the following values:
  - LL_RCC_PLL1SOURCE_NONE
  - LL_RCC_PLL1SOURCE_HSI
  - LL_RCC_PLL1SOURCE_CSI
  - LL_RCC_PLL1SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_SetSource

LL_RCC_PLL1_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetSource (void)

Function description

Get the oscillator used as PLL1 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLL1SOURCE_NONE
  - LL_RCC_PLL1SOURCE_CSI
  - LL_RCC_PLL1SOURCE_HSI
  - LL_RCC_PLL1SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_GetSource

LL_RCC_PLL1_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetN (uint32_t PLL1N)

Function description

Set Main PLL1 multiplication factor for VCO.

Parameters

- **PLL1N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1N LL_RCC_PLL1_SetN

LL_RCC_PLL1_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetN (void)

Function description

Get Main PLL1 multiplication factor for VCO.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1N LL_RCC_PLL1_GetN

LL_RCC_PLL1_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetP (uint32_t PLL1P)

Function description

Set Main PLL1 division factor for PLL1P.

Parameters

- **PLL1P:** parameter can be a value between 2 and 128 (odd value not allowed)

Notes

- Used for System clock

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1P LL_RCC_PLL1_SetP

LL_RCC_PLL1_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetP (void)

Function description

Get PLL1 division factor for PLL1P.

Return values

- **Between:** 2 and 128

Notes

- Used for System clock

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1P LL_RCC_PLL1_GetP

LL_RCC_PLL1_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetQ (uint32_t PLL1Q)

Function description

Set PLL1 division factor for PLL1Q.

Parameters

- **PLL1Q:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLLCFGR PLL1Q LL_RCC_PLL1_SetQ

LL_RCC_PLL1_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetQ (void)

Function description

Get PLL1 division factor for PLL1Q.

Return values

- **Between:** 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1Q LL_RCC_PLL1_GetQ

LL_RCC_PLL1_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetR (uint32_t PLL1R)

Function description

Set PLL1 division factor for PLL1R.

Parameters

- **PLL1R:** parameter can be a value between 1 and 128

Notes

- Used for trace

Reference Manual to LL API cross reference:

- PLL1DIVR PLL1R LL_RCC_PLL1_SetR

LL_RCC_PLL1_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetR (void)

Function description

Get Main PLL1 division factor for PLL1R.

Return values

- **Between:** 1 and 128

Notes

- Used for trace

Reference Manual to LL API cross reference:

- PLL1DIVR PLL1R LL_RCC_PLL1_GetR

LL_RCC_PLL1_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetM (uint32_t PLL1M)

Function description

Set Division factor for the main PLL and other PLL.

Parameters

- **PLL1M:** parameter can be a value between 1 and 63

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1M LL_RCC_PLL1_SetM

LL_RCC_PLL1_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetM (void)

Function description

Get Division factor for the main PLL and other PLL.

Return values

- **Between:** 0 and 63

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1M LL_RCC_PLL1_GetM

LL_RCC_PLL1FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Enable (void)

Function description

Enable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_Enable

LL_RCC_PLL1FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1FRACN_IsEnabled (void)

Function description

Check if PLL1 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_IsEnabled

LL_RCC_PLL1FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Disable (void)

Function description

Disable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_Disable

LL_RCC_PLL1_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetFRACN (uint32_t FRACN)

Function description

Set PLL1 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR PLL1FRACN LL_RCC_PLL1_SetFRACN

LL_RCC_PLL1_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetFRACN (void)

Function description

Get PLL1 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR PLL1FRACN LL_RCC_PLL1_GetFRACN

LL_RCC_PLL1_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL1 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1RGE LL_RCC_PLL1_SetVCOInputRange

LL_RCC_PLL1_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL1 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL1VCOSEL LL_RCC_PLL1_SetVCOOutputRange

LL_RCC_PLL2_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Enable (void)

Function description

Enable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Enable

LL_RCC_PLL2_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Disable (void)

Function description

Disable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Disable

LL_RCC_PLL2_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsReady (void)

Function description

Check if PLL2 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL2RDY LL_RCC_PLL2_IsReady

LL_RCC_PLL2_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetSource (uint32_t PLL2Source)

Function description

Configure PLL2 clock source.

Parameters

- **PLL2Source:** This parameter can be one of the following values:
  - LL_RCC_PLL2SOURCE_NONE
  - LL_RCC_PLL2SOURCE_CSI
  - LL_RCC_PLL2SOURCE_HSI
  - LL_RCC_PLL2SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_SetSource

LL_RCC_PLL2_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetSource (void)

Function description

Get the oscillator used as PLL2 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLL2SOURCE_NONE
  - LL_RCC_PLL2SOURCE_CSI
  - LL_RCC_PLL2SOURCE_HSI
  - LL_RCC_PLL2SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_GetSource

LL_RCC_PLL2_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetM (uint32_t PLL2M)

Function description

Set PLL2 Division factor M.

Parameters

- **PLL2M:** parameter can be a value between 1 and 63

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2M LL_RCC_PLL2_SetM

LL_RCC_PLL2_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetM (void)

Function description

Get PLL2 division factor M.

Return values

- **Between:** 1 and 63

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2M LL_RCC_PLL2_GetM

LL_RCC_PLL2_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetN (uint32_t PLL2N)

Function description

Set PLL2 multiplication factor N.

Parameters

- **PLL2N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2N LL_RCC_PLL2_SetN

LL_RCC_PLL2_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetN (void)

Function description

Get PLL2 multiplication factor N.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2N LL_RCC_PLL2_GetN

LL_RCC_PLL2_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetP (uint32_t PLL2P)

Function description

Set PLL2 division factor P.

Parameters

- **PLL2P:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2P LL_RCC_PLL2_SetP

LL_RCC_PLL2_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetP (void)

Function description

Get PLL2 division factor P.

Return values

- **Between:** 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2P LL_RCC_PLL2_GetP

LL_RCC_PLL2_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetQ (uint32_t PLL2Q)

Function description

Set PLL2 division factor Q.

Parameters

- **PLL2Q:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLLCFGR PLL2Q LL_RCC_PLL2_SetQ

LL_RCC_PLL2_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetQ (void)

Function description

Get PLL2 division factor Q.

Return values

- **Between:** 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2Q LL_RCC_PLL2_GetQ

LL_RCC_PLL2_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetR (uint32_t PLL2R)

Function description

Set PLL2 division factor R.

Parameters

- **PLL2R:** parameter can be a value between 1 and 128

Notes

- Used for PLL2CLK selected for peripherals clocks

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2Q LL_RCC_PLL2_SetR

LL_RCC_PLL2_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetR (void)

Function description

Get PLL2 division factor R.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL2CLK (system clock)

Reference Manual to LL API cross reference:

- PLL2DIVR PLL2R LL_RCC_PLL2_GetR

LL_RCC_PLL2P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Enable (void)

Function description

Enable PLL2 P output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2P_Enable

LL_RCC_PLL2P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Disable (void)

Function description

Disable PLL2 P output.

Return values

- **None:**

Notes

- In order to save power, when PLL2P output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2P_Disable

LL_RCC_PLL2Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2Q_Enable (void)

Function description

Enable PLL2 Q output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2Q_Enable

LL_RCC_PLL2Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2Q_Disable (void)

Function description

Disable PLL2 Q output.

Return values

- **None:**

Notes

- In order to save power, when PLL2Q output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2_Disable

LL_RCC_PLL2R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2R_Enable (void)

Function description

Enable PLL2 R output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2R_Enable

LL_RCC_PLL2R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2R_Disable (void)

Function description

Disable PLL2 R output.

Return values

- **None:**

Notes

- In order to save power, when PLL2R output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2R_Disable

LL_RCC_PLL2P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2P_IsEnabled (void)

Function description

Check if PLL2 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2P_IsEnabled

LL_RCC_PLL2Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2Q_IsEnabled (void)

Function description

Check if PLL2 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2Q_IsEnabled

LL_RCC_PLL2R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2R_IsEnabled (void)

Function description

Check if PLL2 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2R_IsEnabled

LL_RCC_PLL2FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Enable (void)

Function description

Enable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_Enable

LL_RCC_PLL2FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2FRACN_IsEnabled (void)

Function description

Check if PLL2 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_IsEnabled

LL_RCC_PLL2FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Disable (void)

Function description

Disable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_Disable

LL_RCC_PLL2_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetFRACN (uint32_t FRACN)

Function description

Set PLL2 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR PLL2FRACN LL_RCC_PLL2_SetFRACN

LL_RCC_PLL2_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetFRACN (void)

Function description

Get PLL2 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR PLL2FRACN LL_RCC_PLL2_GetFRACN

LL_RCC_PLL2_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL2 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2RGE LL_RCC_PLL2_SetVCOInputRange

LL_RCC_PLL2_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL2 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2VCOSEL LL_RCC_PLL2_SetVCOOutputRange

LL_RCC_PLL3_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Enable (void)

Function description

Enable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Enable

LL_RCC_PLL3_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Disable (void)

Function description

Disable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Disable

LL_RCC_PLL3_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsReady (void)

Function description

Check if PLL3 is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL3RDY LL_RCC_PLL3_IsReady

LL_RCC_PLL3_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetSource (uint32_t PLLSource)

Function description

Configure PLL3 clock source.

Parameters

- **PLLSource:** This parameter can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_CSI
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_SetSource

LL_RCC_PLL3_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetSource (void)

Function description

Get the oscillator used as PLL3 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_CSI
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_GetSource

LL_RCC_PLL3_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetN (uint32_t PLL3N)

Function description

Set PLL3 multiplication factor N.

Parameters

- **PLL3N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3N LL_RCC_PLL3_SetN

LL_RCC_PLL3_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetN (void)

Function description

Get PLL3 multiplication factor N.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3N LL_RCC_PLL3_GetN

LL_RCC_PLL3_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetP (uint32_t PLL3P)

Function description

Set PLL3 division factor P.

Parameters

- **PLL3P:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3P LL_RCC_PLL3_SetP

LL_RCC_PLL3_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetP (void)

Function description

Get PLL3 division factor P.

Return values

- **Between:** 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3P LL_RCC_PLL3_GetP

LL_RCC_PLL3_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetQ (uint32_t PLL3Q)

Function description

Set PLL3 division factor Q.

Parameters

- **PLL3Q:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLLCFGR PLL3Q LL_RCC_PLL3_SetQ

LL_RCC_PLL3_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetQ (void)

Function description

Get PLL3 division factor Q.

Return values

- **Between:** 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3Q LL_RCC_PLL3_GetQ

LL_RCC_PLL3_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetR (uint32_t PLL3R)

Function description

Set PLL3 division factor R.

Parameters

- **PLL3R:** parameter can be a value between 1 and 128

Notes

- Used for peripherals clocks

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3Q LL_RCC_PLL3_SetR

LL_RCC_PLL3_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetR (void)

Function description

Get PLL3 division factor R.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL3CLK (system clock)

Reference Manual to LL API cross reference:

- PLL3DIVR PLL3R LL_RCC_PLL3_GetR

LL_RCC_PLL3_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetM (uint32_t PLL3M)

Function description

Set PLL3 Division factor M.

Parameters

- **PLL3M:** parameter can be a value between 1 and 63

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3M LL_RCC_PLL3_SetM

LL_RCC_PLL3_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetM (void)

Function description

Get PLL3 Division factor M.

Return values

- **Between:** 1 and 63

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3M LL_RCC_PLL3_GetM

LL_RCC_PLL3P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Enable (void)

Function description

Enable PLL3 P output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3P_Enable

LL_RCC_PLL3P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Disable (void)

Function description

Disable PLL3 P output.

Return values

- **None:**

Notes

- In order to save power, when PLL3P output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3P_Disable

LL_RCC_PLL3Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3Q_Enable (void)

Function description

Enable PLL3 Q output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3Q_Enable

LL_RCC_PLL3Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3Q_Disable (void)

Function description

Disable PLL3 Q output.

Return values

- **None:**

Notes

- In order to save power, when PLL3Q output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3Q_Disable

LL_RCC_PLL3R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3R_Enable (void)

Function description

Enable PLL3 R output.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3R_Enable

LL_RCC_PLL3R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3R_Disable (void)

Function description

Disable PLL3 R output.

Return values

- **None:**

Notes

- In order to save power, when PLL3R output is not used, it should be disabled (at any time)

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3R_Disable

LL_RCC_PLL3P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3P_IsEnabled (void)

Function description

Check if PLL3 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3P_IsEnabled

LL_RCC_PLL3Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3Q_IsEnabled (void)

Function description

Check if PLL3 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3Q_IsEnabled

LL_RCC_PLL3R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3R_IsEnabled (void)

Function description

Check if PLL3 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3R_IsEnabled

LL_RCC_PLL3FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Enable (void)

Function description

Enable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_Enable

LL_RCC_PLL3FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3FRACN_IsEnabled (void)

Function description

Check if PLL3 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_IsEnabled

LL_RCC_PLL3FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Disable (void)

Function description

Disable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_Disable

LL_RCC_PLL3_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetFRACN (uint32_t FRACN)

Function description

Set PLL3 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR PLL3FRACN LL_RCC_PLL3_SetFRACN

LL_RCC_PLL3_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetFRACN (void)

Function description

Get PLL3 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR PLL3FRACN LL_RCC_PLL3_GetFRACN

LL_RCC_PLL3_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL3 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3RGE LL_RCC_PLL3_SetVCOInputRange

LL_RCC_PLL3_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL3 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3VCOSEL LL_RCC_PLL3_SetVCOOutputRange

LL_RCC_EnableSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_EnableSecPrivilegedMode (void)

Function description

Enable Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_EnableSecPrivilegedMode

LL_RCC_DisableSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_DisableSecPrivilegedMode (void)

Function description

Disable Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_DisableSecPrivilegedMode

LL_RCC_EnableNSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_EnableNSecPrivilegedMode (void)

Function description

Enable Non Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_EnableNSecPrivilegedMode

LL_RCC_DisableNSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_DisableNSecPrivilegedMode (void)

Function description

Disable Non Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_DisableNSecPrivilegedMode

LL_RCC_IsEnabledSecPrivilegedMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledSecPrivilegedMode (void)

Function description

Check if Secure Privileged mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_IsEnabledSecPrivilegedMode

LL_RCC_IsEnabledNSecPrivilegedMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledNSecPrivilegedMode (void)

Function description

Check if Non Secure Privileged mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_IsEnabledNSecPrivilegedMode

LL_RCC_ClearFlag_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)

Function description

Clear LSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)

Function description

Clear LSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_CSIRDY (void)

Function description

Clear CSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR CSIRDYC LL_RCC_ClearFlag_CSIRDY

LL_RCC_ClearFlag_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)

Function description

Clear HSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)

Function description

Clear HSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)

Function description

Clear HSI48 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL1RDY (void)

Function description

Clear PLL1 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL1RDYC LL_RCC_ClearFlag_PLL1RDY

LL_RCC_ClearFlag_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL2RDY (void)

Function description

Clear PLL2 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL2RDYC LL_RCC_ClearFlag_PLL2RDY

LL_RCC_ClearFlag_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL3RDY (void)

Function description

Clear PLL3 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL3RDYC LL_RCC_ClearFlag_PLL3RDY

LL_RCC_ClearFlag_HSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)

Function description

Clear Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSECSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)

Function description

Check if LSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)

Function description

Check if LSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_CSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_CSIRDY (void)

Function description

Check if CSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR CSIRDYF LL_RCC_IsActiveFlag_CSIRDY

LL_RCC_IsActiveFlag_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)

Function description

Check if HSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)

Function description

Check if HSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)

Function description

Check if HSI48 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL1RDY (void)

Function description

Check if PLL1 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL1RDYF LL_RCC_IsActiveFlag_PLL1RDY

LL_RCC_IsActiveFlag_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL2RDY (void)

Function description

Check if PLL2 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL2RDYF LL_RCC_IsActiveFlag_PLL2RDY

LL_RCC_IsActiveFlag_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL3RDY (void)

Function description

Check if PLL3 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL3RDYF LL_RCC_IsActiveFlag_PLL3RDY

LL_RCC_IsActiveFlag_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)

Function description

Check if Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSECSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_IWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)

Function description

Check if RCC flag Independent Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)

Function description

Check if RCC flag Window Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)

Function description

Enable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_CSIRDY (void)

Function description

Enable CSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_EnableIT_CSIRDY

LL_RCC_EnableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)

Function description

Enable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)

Function description

Enable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)

Function description

Enable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL1RDY (void)

Function description

Enable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_EnableIT_PLL1RDY

LL_RCC_EnableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL2RDY (void)

Function description

Enable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_EnableIT_PLL2RDY

LL_RCC_EnableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL3RDY (void)

Function description

Enable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_EnableIT_PLL3RDY

LL_RCC_DisableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)

Function description

Disable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)

Function description

Disable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_CSIRDY (void)

Function description

Disable CSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_DisableIT_CSIRDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)

Function description

Disable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)

Function description

Disable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL1RDY (void)

Function description

Disable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_DisableIT_PLL1RDY

LL_RCC_DisableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL2RDY (void)

Function description

Disable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_DisableIT_PLL2RDY

LL_RCC_DisableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL3RDY (void)

Function description

Disable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_DisableIT_PLL3RDY

LL_RCC_IsEnabledIT_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)

Function description

Checks if LSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)

Function description

Checks if LSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_CSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_CSIRDY (void)

Function description

Checks if CSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_IsEnabledIT_CSIRDY

LL_RCC_IsEnabledIT_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)

Function description

Checks if HSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

LL_RCC_IsEnabledIT_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)

Function description

Checks if HSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)

Function description

Checks if HSI48 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_IsEnabledIT_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL1RDY (void)

Function description

Checks if PLL1 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_IsEnabledIT_PLL1RDY

LL_RCC_IsEnabledIT_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL2RDY (void)

Function description

Checks if PLL2 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_IsEnabledIT_PLL2RDY

LL_RCC_IsEnabledIT_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL3RDY (void)

Function description

Checks if PLL3 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_IsEnabledIT_PLL3RDY

LL_RCC_ConfigSecure

Function name

__STATIC_INLINE void LL_RCC_ConfigSecure (uint32_t Configuration)

Function description

Configure RCC resources security.

Parameters

- **Configuration:** This parameter shall be the full combination of the following values:
  - LL_RCC_ALL_SEC or LL_RCC_ALL_NSEC
  - LL_RCC_HSI_SEC or LL_RCC_HSI_NSEC
  - LL_RCC_HSE_SEC or LL_RCC_HSE_NSEC
  - LL_RCC_CSI_SEC or LL_RCC_CSI_NSEC
  - LL_RCC_LSE_SEC or LL_RCC_LSE_NSEC
  - LL_RCC_LSI_SEC or LL_RCC_LSI_NSEC
  - LL_RCC_SYSCLK_SEC or LL_RCC_SYSCLK_NSEC
  - LL_RCC_PRESCALERS_SEC or LL_RCC_PRESCALERS_NSEC
  - LL_RCC_PLL1_SEC or LL_RCC_PLL1_NSEC
  - LL_RCC_PLL2_SEC or LL_RCC_PLL2_NSEC
  - LL_RCC_PLL3_SEC or LL_RCC_PLL3_NSEC
  - LL_RCC_HSI48_SEC or LL_RCC_HSI48_NSEC
  - LL_RCC_RESET_FLAGS_SEC or LL_RCC_RESET_FLAGS_NSEC
  - LL_RCC_CKPERSEL_SEC or LL_RCC_CKPERSEL_NSEC

Return values

- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR HSISEC LL_RCC_ConfigSecure
- SECCFGR HSESEC LL_RCC_ConfigSecure
- SECCFGR CSISEC LL_RCC_ConfigSecure
- SECCFGR LSISEC LL_RCC_ConfigSecure
- SECCFGR LSESEC LL_RCC_ConfigSecure
- SECCFGR SYSCLKSEC LL_RCC_ConfigSecure
- SECCFGR PRESCSEC LL_RCC_ConfigSecure
- SECCFGR PLL1SEC LL_RCC_ConfigSecure
- SECCFGR PLL2SEC LL_RCC_ConfigSecure
- SECCFGR PLL3SEC LL_RCC_ConfigSecure
- SECCFGR HSI48SEC LL_RCC_ConfigSecure
- SECCFGR RMVFSEC LL_RCC_ConfigSecure
- SECCFGR CKPERSELSEC LL_RCC_ConfigSecure

LL_RCC_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_RCC_GetConfigSecure (void)

Function description

Get RCC resources security status.

Return values

- **Returned:** value is the combination of the following values:
  - LL_RCC_ALL_SEC or LL_RCC_ALL_NSEC
  - LL_RCC_HSI_SEC or LL_RCC_HSI_NSEC
  - LL_RCC_HSE_SEC or LL_RCC_HSE_NSEC
  - LL_RCC_CSI_SEC or LL_RCC_CSI_NSEC
  - LL_RCC_LSE_SEC or LL_RCC_LSE_NSEC
  - LL_RCC_LSI_SEC or LL_RCC_LSI_NSEC
  - LL_RCC_SYSCLK_SEC or LL_RCC_SYSCLK_NSEC
  - LL_RCC_PRESCALERS_SEC or LL_RCC_PRESCALERS_NSEC
  - LL_RCC_PLL1_SEC or LL_RCC_PLL1_NSEC
  - LL_RCC_PLL2_SEC or LL_RCC_PLL2_NSEC
  - LL_RCC_PLL3_SEC or LL_RCC_PLL3_NSEC
  - LL_RCC_HSI48_SEC or LL_RCC_HSI48_NSEC
  - LL_RCC_RESET_FLAGS_SEC or LL_RCC_RESET_FLAGS_NSEC
  - LL_RCC_CKPERSEL_SEC or LL_RCC_CKPERSEL_NSEC
- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR HSISEC LL_RCC_GetConfigSecure
- SECCFGR HSESEC LL_RCC_GetConfigSecure
- SECCFGR CSISEC LL_RCC_GetConfigSecure
- SECCFGR LSISEC LL_RCC_GetConfigSecure
- SECCFGR LSESEC LL_RCC_GetConfigSecure
- SECCFGR SYSCLKSEC LL_RCC_GetConfigSecure
- SECCFGR PRESCSEC LL_RCC_GetConfigSecure
- SECCFGR PLL1SEC LL_RCC_GetConfigSecure
- SECCFGR PLL2SEC LL_RCC_GetConfigSecure
- SECCFGR PLL3SEC LL_RCC_GetConfigSecure
- SECCFGR HSI48SEC LL_RCC_GetConfigSecure
- SECCFGR RMVFSEC LL_RCC_GetConfigSecure
- SECCFGR CKPERSELSEC LL_RCC_GetConfigSecure

LL_RCC_DeInit

Function name

ErrorStatus LL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RCC registers are de-initialized
  - ERROR: not applicable

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, CSI, PLL1, PLL2 and PLL3 OFFAHB, APB1, APB2 and APB3 prescaler set to 1.CSS OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_CalcPLLClockFreq

Function name

uint32_t LL_RCC_CalcPLLClockFreq (uint32_t PLLInputFreq, uint32_t M, uint32_t N, uint32_t FRACN, uint32_t PQR)

Function description

Helper function to calculate the PLL1 frequency output.

Parameters

- **PLLInputFreq:** PLL Input frequency (based on HSE/(HSI/HSIDIV)/CSI)
- **M:** Between 1 and 63
- **N:** Between 4 and 512
- **FRACN:** Between 0 and 0x1FFF
- **PQR:** VCO output divider (P, Q or R) Between 1 and 128, except for PLL1P Odd value not allowed

Return values

- **PLL1:** output clock frequency (in Hz)

Notes

- ex: LL_RCC_CalcPLLClockFreq (HSE_VALUE, LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetFRACN (), LL_RCC_PLL1_GetP ());

LL_RCC_GetPLL1ClockFreq

Function name

void LL_RCC_GetPLL1ClockFreq (LL_PLL_ClocksTypeDef * pPLL_Clocks)

Function description

Return PLL1 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL2ClockFreq

Function name

void LL_RCC_GetPLL2ClockFreq (LL_PLL_ClocksTypeDef * pPLL_Clocks)

Function description

Return PLL2 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL3ClockFreq

Function name

void LL_RCC_GetPLL3ClockFreq (LL_PLL_ClocksTypeDef * pPLL_Clocks)

Function description

Return PLL3 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * pRCC_Clocks)

Function description

Return the frequencies of different on chip clocks; System, AHB, APB1, APB2 and APB3 buses clocks.

Parameters

- **pRCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK, PCLK1, PCLK2 and PCLK3 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name

uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)

Function description

Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE
  - LL_RCC_USART2_CLKSOURCE
  - LL_RCC_USART3_CLKSOURCE
  - LL_RCC_USART6_CLKSOURCE (*)
  - LL_RCC_USART10_CLKSOURCE (*)
  - LL_RCC_USART11_CLKSOURCE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **USART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetUARTClockFreq

Function name

uint32_t LL_RCC_GetUARTClockFreq (uint32_t UARTxSource)

Function description

Return UARTx clock frequency.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  - LL_RCC_UART4_CLKSOURCE
  - LL_RCC_UART5_CLKSOURCE
  - LL_RCC_UART7_CLKSOURCE
  - LL_RCC_UART8_CLKSOURCE
  - LL_RCC_UART9_CLKSOURCE
  - LL_RCC_UART12_CLKSOURCE

Return values

- **UART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetI2CClockFreq

Function name

uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)

Function description

Return I2Cx clock frequency.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE (*)
  - LL_RCC_I2C4_CLKSOURCE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetI3CClockFreq

Function name

uint32_t LL_RCC_GetI3CClockFreq (uint32_t I3CxSource)

Function description

Return I3Cx clock frequency.

Parameters

- **I3CxSource:** This parameter can be one of the following values:
  - LL_RCC_I3C1_CLKSOURCE
  - LL_RCC_I3C2_CLKSOURCE (*)

  (*) : For stm32h503xx family line only.

Return values

- **I3C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready or no clock is selected

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUARTx clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetLPTIMClockFreq

Function name

uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)

Function description

Return LPTIMx clock frequency.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM3_CLKSOURCE (*)
  - LL_RCC_LPTIM4_CLKSOURCE (*)
  - LL_RCC_LPTIM5_CLKSOURCE (*)
  - LL_RCC_LPTIM6_CLKSOURCE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **LPTIM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCAN kernel clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **FDCAN:** kernel clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready or no clock is selected

LL_RCC_GetSAIClockFreq

Function name

uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)

Function description

Return SAIx clock frequency.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE

Return values

- **SAI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that PLL is not ready

LL_RCC_GetSDMMCClockFreq

Function name

uint32_t LL_RCC_GetSDMMCClockFreq (uint32_t SDMMCxSource)

Function description

Return SDMMCx clock frequency.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC1_CLKSOURCE
  - LL_RCC_SDMMC2_CLKSOURCE (*)

Return values

- **SDMMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetRNGClockFreq

Function name

uint32_t LL_RCC_GetRNGClockFreq (uint32_t RNGxSource)

Function description

Return RNGx clock frequency.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **RNG:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetUSBClockFreq

Function name

uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)

Function description

Return USBx clock frequency.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **USB:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready or no clock is selected

LL_RCC_GetADCDACClockFreq

Function name

uint32_t LL_RCC_GetADCDACClockFreq (uint32_t ADCDACxSource)

Function description

Return ADCxDAC clock frequency.

Parameters

- **ADCDACxSource:** This parameter can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE

Return values

- **ADCDAC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetDACLPClockFreq

Function name

uint32_t LL_RCC_GetDACLPClockFreq (uint32_t DACLPxSource)

Function description

Return DAC low-power clock frequency.

Parameters

- **DACLPxSource:** This parameter can be one of the following values:
  - LL_RCC_DAC_LP_CLKSOURCE

Return values

- **DAC:** low-power clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that LSI or LSE oscillator is not ready

LL_RCC_GetOCTOSPIClockFreq

Function name

uint32_t LL_RCC_GetOCTOSPIClockFreq (uint32_t OCTOSPIxSource)

Function description

Return OCTOSPI clock frequency.

Parameters

- **OCTOSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_OCTOSPI_CLKSOURCE

Return values

- **OCTOSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetSPIClockFreq

Function name

uint32_t LL_RCC_GetSPIClockFreq (uint32_t SPIxSource)

Function description

Return SPIx clock frequency.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE
  - LL_RCC_SPI4_CLKSOURCE (*)
  - LL_RCC_SPI5_CLKSOURCE (*)
  - LL_RCC_SPI6_CLKSOURCE (*)

  (*) : For stm32h56xxx and stm32h57xxx family lines only.

Return values

- **SPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetCECClockFreq

Function name

uint32_t LL_RCC_GetCECClockFreq (uint32_t CECxSource)

Function description

Return CEC clock frequency.

Parameters

- **CECxSource:** This parameter can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE

Return values

- **CEC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready or no clock is selected

LL_RCC_GetCLKPClockFreq

Function name

uint32_t LL_RCC_GetCLKPClockFreq (uint32_t CLKPxSource)

Function description

Return CLKP clock frequency.

Parameters

- **CLKPxSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **CLKP:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready or no clock is selected

109.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

109.3.1 RCC

RCC

Peripheral ADCDAC get clock source

LL_RCC_ADCDAC_CLKSOURCE

ADCDACs Clock source selection

Peripheral ADCDAC clock source selection

LL_RCC_ADCDAC_CLKSOURCE_HCLK

AHB clock used as ADCDAC clock source

LL_RCC_ADCDAC_CLKSOURCE_SYSCLK

SYSCLK clock used as ADCDAC clock source

LL_RCC_ADCDAC_CLKSOURCE_PLL2R

PLL2 R clock used as ADCDAC clock source

LL_RCC_ADCDAC_CLKSOURCE_HSE

HSE clock used as ADCDAC clock source

LL_RCC_ADCDAC_CLKSOURCE_HSI

HSI clock used as ADCDAC clock source

LL_RCC_ADCDAC_CLKSOURCE_CSI

CSI clock used as ADCDAC clock source

AHB1 peripherals clock branch disable

LL_RCC_AHB1_PERIPH_DIS

Clock Branch disable for all AHB1 peripherals

AHB2 peripherals clock branch disable

LL_RCC_AHB2_PERIPH_DIS

Clock Branch disable for all AHB2 peripherals

AHB4 peripherals clock branch disable

LL_RCC_AHB4_PERIPH_DIS

Clock Branch disable for all AHB4 peripherals

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1

HCLK not divided

LL_RCC_APB1_DIV_2

HCLK divided by 2

LL_RCC_APB1_DIV_4

HCLK divided by 4

LL_RCC_APB1_DIV_8

HCLK divided by 8

LL_RCC_APB1_DIV_16

HCLK divided by 16

APB1 peripherals clock branch disable

LL_RCC_APB1_PERIPH_DIS

Clock Branch disable for all APB1 peripherals

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1

HCLK not divided

LL_RCC_APB2_DIV_2

HCLK divided by 2

LL_RCC_APB2_DIV_4

HCLK divided by 4

LL_RCC_APB2_DIV_8

HCLK divided by 8

LL_RCC_APB2_DIV_16

HCLK divided by 16

APB2 peripherals clock branch disable

LL_RCC_APB2_PERIPH_DIS

Clock Branch disable for all APB2 peripherals

APB high-speed prescaler (APB3)

LL_RCC_APB3_DIV_1

HCLK not divided

LL_RCC_APB3_DIV_2

HCLK divided by 2

LL_RCC_APB3_DIV_4

HCLK divided by 4

LL_RCC_APB3_DIV_8

HCLK divided by 8

LL_RCC_APB3_DIV_16

HCLK divided by 16

APB3 peripherals clock branch disable

LL_RCC_APB3_PERIPH_DIS

Clock Branch disable for all APB3 peripherals

Peripheral CEC get clock source

LL_RCC_CEC_CLKSOURCE

Peripheral CEC clock source selection

LL_RCC_CEC_CLKSOURCE_LSE

LSE clock used as CEC clock

LL_RCC_CEC_CLKSOURCE_LSI

LSI clock used as CEC clock

LL_RCC_CEC_CLKSOURCE_CSI_DIV122

CSI clock divied by 122 used as CEC clock

LL_RCC_CEC_CLKSOURCE_NONE

NO clock used as CEC clock source

Peripheral CLKP get clock source

LL_RCC_CLKP_CLKSOURCE

CLKP Clock source selection

Peripheral CLKP clock source selection

LL_RCC_CLKP_CLKSOURCE_HSI

HSI clock used as CLKP clock source

LL_RCC_CLKP_CLKSOURCE_CSI

CSI clock used as CLKP clock source

LL_RCC_CLKP_CLKSOURCE_HSE

HSE clock used as CLKP clock source

LL_RCC_CLKP_CLKSOURCE_NONE

No clock selected as CLKP clock source

Peripheral DAC get low-power clock source

LL_RCC_DAC_LP_CLKSOURCE

DAC low-power Clock source selection

Peripheral DAC low-power clock source selection

LL_RCC_DAC_LP_CLKSOURCE_LSE

LSE clock used as DAC low-power clock

LL_RCC_DAC_LP_CLKSOURCE_LSI

LSI clock used as DAC low-power clock

Peripheral FDCAN get kernel clock source

LL_RCC_FDCAN_CLKSOURCE

FDCAN kernel Clock source selection

Peripheral FDCAN kernel clock source selection

LL_RCC_FDCAN_CLKSOURCE_HSE

HSE clock used as FDCAN kernel clock source

LL_RCC_FDCAN_CLKSOURCE_PLL1Q

PLL1 Q clock used as FDCAN kernel clock source

LL_RCC_FDCAN_CLKSOURCE_PLL2Q

PLL2 Q clock used as FDCAN kernel clock source

LL_RCC_FDCAN_CLKSOURCE_NONE

NO clock used as FDCAN kernel clock source

EXTERNAL HSE clock Type

LL_RCC_HSE_ANALOG_TYPE

ANALOG clock used as HSE external clock source

LL_RCC_HSE_DIGITAL_TYPE

DIGITAL clock used as HSE external clock source

HSI oscillator divider

LL_RCC_HSI_DIV_1

HSI_DIV1 clock activation

LL_RCC_HSI_DIV_2

HSI_DIV2 clock activation

LL_RCC_HSI_DIV_4

HSI_DIV4 clock activation

LL_RCC_HSI_DIV_8

HSI_DIV8 clock activation

Peripheral I2Cx get clock source

LL_RCC_I2C1_CLKSOURCE

I2C1 Clock source selection

LL_RCC_I2C2_CLKSOURCE

I2C2 Clock source selection

LL_RCC_I2C3_CLKSOURCE

I2C3 Clock source selection

LL_RCC_I2C4_CLKSOURCE

I2C4 Clock source selection

Peripheral I2Cx clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1

PCLK1 clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_PLL3R

PLL3 R clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_HSI

HSI clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_CSI

CSI clock used as I2C1 clock source

LL_RCC_I2C2_CLKSOURCE_PCLK1

PCLK1 clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_PLL3R

PLL3 R clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_HSI

HSI clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_CSI

CSI clock used as I2C2 clock source

LL_RCC_I2C3_CLKSOURCE_PCLK3

PCLK3 clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_PLL3R

PLL3 R clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_HSI

HSI clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_CSI

CSI clock used as I2C3 clock source

LL_RCC_I2C4_CLKSOURCE_PCLK3

PCLK3 clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_PLL3R

PLL3 R clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_HSI

HSI clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_CSI

CSI clock used as I2C4 clock source

Peripheral I3Cx get clock source

LL_RCC_I3C1_CLKSOURCE

I3C1 Clock source selection

Peripheral I3Cx clock source selection

LL_RCC_I3C1_CLKSOURCE_PCLK1

PCLK1 clock used as I3C1 clock source

LL_RCC_I3C1_CLKSOURCE_PLL3R

PLL3 R clock used as I3C1 clock source

LL_RCC_I3C1_CLKSOURCE_HSI

HSI clock used as I3C1 clock source

LL_RCC_I3C1_CLKSOURCE_NONE

NONE clock used as I3C1 clock source

Kernel wakeup from stop clock source

LL_RCC_KERWAKEUP_CLKSOURCE_HSI

HSI selection as kernel clock after wake-up from STOP

LL_RCC_KERWAKEUP_CLKSOURCE_CSI

CSI selection as kernel clock after wake-up from STOP

Peripheral LPTIMx get clock source

LL_RCC_LPTIM1_CLKSOURCE

LPTIM1 Clock source selection

LL_RCC_LPTIM2_CLKSOURCE

LPTIM2 Clock source selection

LL_RCC_LPTIM3_CLKSOURCE

LPTIM3 Clock source selection

LL_RCC_LPTIM4_CLKSOURCE

LPTIM4 Clock source selection

LL_RCC_LPTIM5_CLKSOURCE

LPTIM5 Clock source selection

LL_RCC_LPTIM6_CLKSOURCE

LPTIM6 Clock source selection

Peripheral LPTIMx clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK3

PCLK3 clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSE

LSE clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSI

LSI clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_CLKP

CLKP clock used as LPTIM1 clock source

LL_RCC_LPTIM2_CLKSOURCE_PCLK1

PCLK1 clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_LSE

LSE clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_LSI

LSI clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_CLKP

CLKP clock used as LPTIM2 clock source

LL_RCC_LPTIM3_CLKSOURCE_PCLK3

PCLK3 clock used as LPTIM3 clock source

LL_RCC_LPTIM3_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM3 clock source

LL_RCC_LPTIM3_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM3 clock source

LL_RCC_LPTIM3_CLKSOURCE_LSE

LSE clock used as LPTIM3 clock source

LL_RCC_LPTIM3_CLKSOURCE_LSI

LSI clock used as LPTIM3 clock source

LL_RCC_LPTIM3_CLKSOURCE_CLKP

CLKP clock used as LPTIM3 clock source

LL_RCC_LPTIM4_CLKSOURCE_PCLK3

PCLK3 clock used as LPTIM4 clock source

LL_RCC_LPTIM4_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM4 clock source

LL_RCC_LPTIM4_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM4 clock source

LL_RCC_LPTIM4_CLKSOURCE_LSE

LSE clock used as LPTIM4 clock source

LL_RCC_LPTIM4_CLKSOURCE_LSI

LSI clock used as LPTIM4 clock source

LL_RCC_LPTIM4_CLKSOURCE_CLKP

CLKP clock used as LPTIM4 clock source

LL_RCC_LPTIM5_CLKSOURCE_PCLK3

PCLK3 clock used as LPTIM5 clock source

LL_RCC_LPTIM5_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM5 clock source

LL_RCC_LPTIM5_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM5 clock source

LL_RCC_LPTIM5_CLKSOURCE_LSE

LSE clock used as LPTIM5 clock source

LL_RCC_LPTIM5_CLKSOURCE_LSI

LSI clock used as LPTIM5 clock source

LL_RCC_LPTIM5_CLKSOURCE_CLKP

CLKP clock used as LPTIM5 clock source

LL_RCC_LPTIM6_CLKSOURCE_PCLK3

PCLK3 clock used as LPTIM6 clock source

LL_RCC_LPTIM6_CLKSOURCE_PLL2P

PLL2 P clock used as LPTIM6 clock source

LL_RCC_LPTIM6_CLKSOURCE_PLL3R

PLL3 R clock used as LPTIM6 clock source

LL_RCC_LPTIM6_CLKSOURCE_LSE

LSE clock used as LPTIM6 clock source

LL_RCC_LPTIM6_CLKSOURCE_LSI

LSI clock used as LPTIM6 clock source

LL_RCC_LPTIM6_CLKSOURCE_CLKP

CLKP clock used as LPTIM6 clock source

Peripheral LPUARTx get clock source

LL_RCC_LPUART1_CLKSOURCE

LPUART1 Clock source selection

Peripheral LPUARTx clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK3

PCLK3 clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_PLL2Q

PLL2Q clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_PLL3Q

PLL3Q clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_HSI

HSI clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_CSI

CSI clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_LSE

LSE clock used as LPUART1 clock source

LSCO Selection

LL_RCC_LSCO_CLKSOURCE_LSI

LSI selection for low speed clock

LL_RCC_LSCO_CLKSOURCE_LSE

LSE selection for low speed clock

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW

Xtal mode lower driving capability

LL_RCC_LSEDRIVE_MEDIUMLOW

Xtal mode medium low driving capability

LL_RCC_LSEDRIVE_MEDIUMHIGH

Xtal mode medium high driving capability

LL_RCC_LSEDRIVE_HIGH

Xtal mode higher driving capability

EXTERNAL LSE clock Type

LL_RCC_LSE_ANALOG_TYPE

ANALOG clock used as LSE external clock source

LL_RCC_LSE_DIGITAL_TYPE

DIGITAL clock used as LSE external clock source

MCO SOURCE selection

LL_RCC_MCO1SOURCE_HSI

LL_RCC_MCO1SOURCE_LSE

LL_RCC_MCO1SOURCE_HSE

LL_RCC_MCO1SOURCE_PLL1Q

LL_RCC_MCO1SOURCE_HSI48

LL_RCC_MCO2SOURCE_SYSCLK

LL_RCC_MCO2SOURCE_PLL2P

LL_RCC_MCO2SOURCE_HSE

LL_RCC_MCO2SOURCE_PLL1P

LL_RCC_MCO2SOURCE_CSI

LL_RCC_MCO2SOURCE_LSI

MCO prescaler

LL_RCC_MCO1_DIV_1

LL_RCC_MCO1_DIV_2

LL_RCC_MCO1_DIV_3

LL_RCC_MCO1_DIV_4

LL_RCC_MCO1_DIV_5

LL_RCC_MCO1_DIV_6

LL_RCC_MCO1_DIV_7

LL_RCC_MCO1_DIV_8

LL_RCC_MCO1_DIV_9

LL_RCC_MCO1_DIV_10

LL_RCC_MCO1_DIV_11

LL_RCC_MCO1_DIV_12

LL_RCC_MCO1_DIV_13

LL_RCC_MCO1_DIV_14

LL_RCC_MCO1_DIV_15

LL_RCC_MCO2_DIV_1

LL_RCC_MCO2_DIV_2

LL_RCC_MCO2_DIV_3

LL_RCC_MCO2_DIV_4

LL_RCC_MCO2_DIV_5

LL_RCC_MCO2_DIV_6

LL_RCC_MCO2_DIV_7

LL_RCC_MCO2_DIV_8

LL_RCC_MCO2_DIV_9

LL_RCC_MCO2_DIV_10

LL_RCC_MCO2_DIV_11

LL_RCC_MCO2_DIV_12

LL_RCC_MCO2_DIV_13

LL_RCC_MCO2_DIV_14

LL_RCC_MCO2_DIV_15

Peripheral OCTOSPI get clock source

LL_RCC_OCTOSPI_CLKSOURCE

OctoSPI Clock source selection

Peripheral OCTOSPI kernel clock source selection

LL_RCC_OSPI_CLKSOURCE_HCLK

AHB clock used as OctoSPI kernel clock source

LL_RCC_OSPI_CLKSOURCE_PLL1Q

PLL1 Q clock used as OctoSPI kernel clock source

LL_RCC_OSPI_CLKSOURCE_PLL2R

PLL2 R clock used as OctoSPI kernel clock source

LL_RCC_OSPI_CLKSOURCE_CLKP

CLKP clock used as OctoSPI clock source

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

LL_RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

PLL1 entry clock source

LL_RCC_PLL1SOURCE_NONE

No clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_HSI

HSI clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_CSI

CSI clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_HSE

HSE clock selected as main PLL1 entry clock source

PLL2 entry clock source

LL_RCC_PLL2SOURCE_NONE

No clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_HSI

HSI clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_CSI

CSI clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_HSE

HSE clock selected as main PLL2 entry clock source

PLL3 entry clock source

LL_RCC_PLL3SOURCE_NONE

No clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_HSI

HSI clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_CSI

CSI clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_HSE

HSE clock selected as main PLL3 entry clock source

All PLLs input ranges

LL_RCC_PLLINPUTRANGE_1_2

VCO input range: 1 to 2 MHz

LL_RCC_PLLINPUTRANGE_2_4

VCO input range: 2 to 4 MHz

LL_RCC_PLLINPUTRANGE_4_8

VCO input range: 4 to 8 MHz

LL_RCC_PLLINPUTRANGE_8_16

VCO input range: 8 to 16 MHz

All PLLs output ranges

LL_RCC_PLLVCORANGE_WIDE

VCO output range: 192 to 836 MHz

LL_RCC_PLLVCORANGE_MEDIUM

VCO output range: 150 to 420 MHz

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE

RNG Clock source selection

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_HSI48

HSI48 clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_PLL1Q

PLL1 Q clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_LSE

LSE clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_LSI

LSI clock used as RNG clock source

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE

No clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSE

LSE oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSI

LSI oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_HSE_DIV

HSE oscillator clock divided by RTCPRE used as RTC clock

RTC HSE Prescaler

LL_RCC_RTC_HSE_NOCLOCK

LL_RCC_RTC_HSE_DIV_2

LL_RCC_RTC_HSE_DIV_3

LL_RCC_RTC_HSE_DIV_4

LL_RCC_RTC_HSE_DIV_5

LL_RCC_RTC_HSE_DIV_6

LL_RCC_RTC_HSE_DIV_7

LL_RCC_RTC_HSE_DIV_8

LL_RCC_RTC_HSE_DIV_9

LL_RCC_RTC_HSE_DIV_10

LL_RCC_RTC_HSE_DIV_11

LL_RCC_RTC_HSE_DIV_12

LL_RCC_RTC_HSE_DIV_13

LL_RCC_RTC_HSE_DIV_14

LL_RCC_RTC_HSE_DIV_15

LL_RCC_RTC_HSE_DIV_16

LL_RCC_RTC_HSE_DIV_17

LL_RCC_RTC_HSE_DIV_18

LL_RCC_RTC_HSE_DIV_19

LL_RCC_RTC_HSE_DIV_20

LL_RCC_RTC_HSE_DIV_21

LL_RCC_RTC_HSE_DIV_22

LL_RCC_RTC_HSE_DIV_23

LL_RCC_RTC_HSE_DIV_24

LL_RCC_RTC_HSE_DIV_25

LL_RCC_RTC_HSE_DIV_26

LL_RCC_RTC_HSE_DIV_27

LL_RCC_RTC_HSE_DIV_28

LL_RCC_RTC_HSE_DIV_29

LL_RCC_RTC_HSE_DIV_30

LL_RCC_RTC_HSE_DIV_31

LL_RCC_RTC_HSE_DIV_32

LL_RCC_RTC_HSE_DIV_33

LL_RCC_RTC_HSE_DIV_34

LL_RCC_RTC_HSE_DIV_35

LL_RCC_RTC_HSE_DIV_36

LL_RCC_RTC_HSE_DIV_37

LL_RCC_RTC_HSE_DIV_38

LL_RCC_RTC_HSE_DIV_39

LL_RCC_RTC_HSE_DIV_40

LL_RCC_RTC_HSE_DIV_41

LL_RCC_RTC_HSE_DIV_42

LL_RCC_RTC_HSE_DIV_43

LL_RCC_RTC_HSE_DIV_44

LL_RCC_RTC_HSE_DIV_45

LL_RCC_RTC_HSE_DIV_46

LL_RCC_RTC_HSE_DIV_47

LL_RCC_RTC_HSE_DIV_48

LL_RCC_RTC_HSE_DIV_49

LL_RCC_RTC_HSE_DIV_50

LL_RCC_RTC_HSE_DIV_51

LL_RCC_RTC_HSE_DIV_52

LL_RCC_RTC_HSE_DIV_53

LL_RCC_RTC_HSE_DIV_54

LL_RCC_RTC_HSE_DIV_55

LL_RCC_RTC_HSE_DIV_56

LL_RCC_RTC_HSE_DIV_57

LL_RCC_RTC_HSE_DIV_58

LL_RCC_RTC_HSE_DIV_59

LL_RCC_RTC_HSE_DIV_60

LL_RCC_RTC_HSE_DIV_61

LL_RCC_RTC_HSE_DIV_62

LL_RCC_RTC_HSE_DIV_63

Peripheral SAIx get clock source

LL_RCC_SAI1_CLKSOURCE

SAI1 Clock source selection

LL_RCC_SAI2_CLKSOURCE

SAI2 Clock source selection

Peripheral SAIx clock source selection

LL_RCC_SAI1_CLKSOURCE_PLL1Q

PLL1 Q clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PLL2P

PLL2 P clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PLL3P

PLL3 P clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PIN

External input clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_CLKP

CLKP clock used as SAI1 clock source

LL_RCC_SAI2_CLKSOURCE_PLL1Q

PLL1 Q clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PLL2P

PLL2 P clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PLL3P

PLL3 P clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PIN

External input clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_CLKP

CLKP clock used as SAI2 clock source

Peripheral SDMMC get clock source

LL_RCC_SDMMC1_CLKSOURCE

SDMMC1 Kernel Clock source selection

LL_RCC_SDMMC2_CLKSOURCE

SDMMC2 Kernel Clock source selection

Peripheral SDMMCx kernel clock source selection

LL_RCC_SDMMC1_CLKSOURCE_PLL1Q

PLL1 Q used as SDMMC1 clock source

LL_RCC_SDMMC1_CLKSOURCE_PLL2R

PLL2 R used as SDMMC1 clock source

LL_RCC_SDMMC2_CLKSOURCE_PLL1Q

PLL1 Q used as SDMMC2 clock source

LL_RCC_SDMMC2_CLKSOURCE_PLL2R

PLL2 R used as SDMMC2 clock source

Secure attributes

LL_RCC_ALL_SEC

Security on all RCC resources

LL_RCC_ALL_NSEC

No security on RCC resources (default)

LL_RCC_HSI_SEC

HSI clock configuration secure-only access

LL_RCC_HSI_NSEC

HSI clock configuration secure/non-secure access

LL_RCC_HSE_SEC

HSE clock configuration secure-only access

LL_RCC_HSE_NSEC

HSE clock configuration secure/non-secure access

LL_RCC_CSI_SEC

CSI clock configuration secure-only access

LL_RCC_CSI_NSEC

CSI clock configuration secure/non-secure access

LL_RCC_LSI_SEC

LSI clock configuration secure-only access

LL_RCC_LSI_NSEC

LSI clock configuration secure/non-secure access

LL_RCC_LSE_SEC

LSE clock configuration secure-only access

LL_RCC_LSE_NSEC

LSE clock configuration secure/non-secure access

LL_RCC_SYSCLK_SEC

SYSCLK clock; STOPWUCK and MCO output configuration secure-only access

LL_RCC_SYSCLK_NSEC

SYSCLK clock; STOPWUCK and MCO output configuration secure/non-secure access

LL_RCC_PRESCALERS_SEC

AHBx/APBx prescaler configuration secure-only access

LL_RCC_PRESCALERS_NSEC

AHBx/APBx prescaler configuration secure/non-secure access

LL_RCC_PLL1_SEC

main PLL clock configuration secure-only access

LL_RCC_PLL1_NSEC

main PLL clock configuration secure/non-secure access

LL_RCC_PLL2_SEC

PLL2 clock configuration secure-only access

LL_RCC_PLL2_NSEC

PLL2 clock configuration secure/non-secure access

LL_RCC_PLL3_SEC

PLL3 clock configuration secure-only access

LL_RCC_PLL3_NSEC

PLL3 clock configuration secure/non-secure access

LL_RCC_HSI48_SEC

HSI48 clock configuration secure-only access

LL_RCC_HSI48_NSEC

HSI48 clock configuration secure/non-secure access

LL_RCC_RESET_FLAGS_SEC

Remove reset flag secure-ony access

LL_RCC_RESET_FLAGS_NSEC

Remove reset flag secure/non-secure access

LL_RCC_CKPERSEL_SEC

Periph clock configuration secure-ony access

LL_RCC_CKPERSEL_NSEC

Periph clock configuration secure/non-secure access

Peripheral SPIx get clock source

LL_RCC_SPI1_CLKSOURCE

SPI1 Clock source selection

LL_RCC_SPI2_CLKSOURCE

SPI2 Clock source selection

LL_RCC_SPI3_CLKSOURCE

SPI3 Clock source selection

LL_RCC_SPI4_CLKSOURCE

SPI4 Clock source selection

LL_RCC_SPI5_CLKSOURCE

SPI5 Clock source selection

LL_RCC_SPI6_CLKSOURCE

SPI6 Clock source selection

Peripheral SPIx clock source selection

LL_RCC_SPI1_CLKSOURCE_PLL1Q

PLL1 Q clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_PLL2P

PLL2 P clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_PLL3P

PLL3 P clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_PIN

PIN clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_CLKP

CLKP clock used as SPI1 clock source

LL_RCC_SPI2_CLKSOURCE_PLL1Q

PLL1 Q clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_PLL2P

PLL2 P clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_PLL3P

PLL3 P clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_PIN

PIN clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_CLKP

CLKP clock used as SPI2 clock source

LL_RCC_SPI3_CLKSOURCE_PLL1Q

PLL1 Q clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_PLL2P

PLL2 P clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_PLL3P

PLL3 P clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_PIN

PIN clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_CLKP

CLKP clock used as SPI3 clock source

LL_RCC_SPI4_CLKSOURCE_PCLK2

PCLK2 clock used as SPI4 clock source

LL_RCC_SPI4_CLKSOURCE_PLL2Q

PLL2 Q clock used as SPI4 clock source

LL_RCC_SPI4_CLKSOURCE_PLL3Q

PLL3 Q clock used as SPI4 clock source

LL_RCC_SPI4_CLKSOURCE_HSI

HSI clock used as SPI4 clock source

LL_RCC_SPI4_CLKSOURCE_CSI

CSI clock used as SPI4 clock source

LL_RCC_SPI4_CLKSOURCE_HSE

HSE clock used as SPI4 clock source

LL_RCC_SPI5_CLKSOURCE_PCLK3

PCLK2 clock used as SPI5 clock source

LL_RCC_SPI5_CLKSOURCE_PLL2Q

PLL2 Q clock used as SPI5 clock source

LL_RCC_SPI5_CLKSOURCE_PLL3Q

PLL3 Q clock used as SPI5 clock source

LL_RCC_SPI5_CLKSOURCE_HSI

HSI clock used as SPI5 clock source

LL_RCC_SPI5_CLKSOURCE_CSI

CSI clock used as SPI5 clock source

LL_RCC_SPI5_CLKSOURCE_HSE

HSE clock used as SPI5 clock source

LL_RCC_SPI6_CLKSOURCE_PCLK2

PCLK2 clock used as SPI6 clock source

LL_RCC_SPI6_CLKSOURCE_PLL2Q

PLL2 Q clock used as SPI6 clock source

LL_RCC_SPI6_CLKSOURCE_PLL3Q

PLL3 Q clock used as SPI6 clock source

LL_RCC_SPI6_CLKSOURCE_HSI

HSI clock used as SPI6 clock source

LL_RCC_SPI6_CLKSOURCE_CSI

CSI clock used as SPI6 clock source

LL_RCC_SPI6_CLKSOURCE_HSE

HSE clock used as SPI6 clock source

AHB prescaler

LL_RCC_SYSCLK_DIV_1

SYSCLK not divided

LL_RCC_SYSCLK_DIV_2

SYSCLK divided by 2

LL_RCC_SYSCLK_DIV_4

SYSCLK divided by 4

LL_RCC_SYSCLK_DIV_8

SYSCLK divided by 8

LL_RCC_SYSCLK_DIV_16

SYSCLK divided by 16

LL_RCC_SYSCLK_DIV_64

SYSCLK divided by 64

LL_RCC_SYSCLK_DIV_128

SYSCLK divided by 128

LL_RCC_SYSCLK_DIV_256

SYSCLK divided by 256

LL_RCC_SYSCLK_DIV_512

SYSCLK divided by 512

SYSTICK clock source selection

LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8

HCLKDIV8 clock used as SYSTICK clock source

LL_RCC_SYSTICK_CLKSOURCE_LSI

LSI clock used as SYSTICK clock source

LL_RCC_SYSTICK_CLKSOURCE_LSE

LSE clock used as SYSTICK clock source

System wakeup from stop and CSS backup clock selection

LL_RCC_SYSWAKEUP_CLKSOURCE_HSI

HSI selection as system clock after wake-up from STOP

LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

CSI selection as system clock after wake-up from STOP

System clock switch status

LL_RCC_SYS_CLKSOURCE_HSI

HSI oscillator selection as system clock

LL_RCC_SYS_CLKSOURCE_CSI

CSI oscillator selection as system clock

LL_RCC_SYS_CLKSOURCE_HSE

HSE oscillator selection as system clock

LL_RCC_SYS_CLKSOURCE_PLL1

PLL1 selection as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSI

HSI oscillator used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_CSI

CSI oscillator used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSE

HSE oscillator used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

PLL1 used as system clock

Timers clocks prescalers selection

LL_RCC_TIM_PRESCALER_TWICE

LL_RCC_TIM_PRESCALER_FOUR_TIMES

Peripheral UARTx get clock source

LL_RCC_UART4_CLKSOURCE

UART4 Clock source selection

LL_RCC_UART5_CLKSOURCE

UART5 Clock source selection

LL_RCC_UART7_CLKSOURCE

UART7 Clock source selection

LL_RCC_UART8_CLKSOURCE

UART8 Clock source selection

LL_RCC_UART9_CLKSOURCE

UART9 Clock source selection

LL_RCC_UART12_CLKSOURCE

UART12 Clock source selection

Peripheral UARTx clock source selection

LL_RCC_UART4_CLKSOURCE_PCLK1

PCLK1 clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_HSI

HSI clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_CSI

CSI clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_LSE

LSE clock used as UART4 clock source

LL_RCC_UART5_CLKSOURCE_PCLK1

PCLK1 clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_HSI

HSI clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_CSI

CSI clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_LSE

LSE clock used as UART5 clock source

LL_RCC_UART7_CLKSOURCE_PCLK1

PCLK1 clock used as UART7 clock source

LL_RCC_UART7_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART7 clock source

LL_RCC_UART7_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART7 clock source

LL_RCC_UART7_CLKSOURCE_HSI

HSI clock used as UART7 clock source

LL_RCC_UART7_CLKSOURCE_CSI

CSI clock used as UART7 clock source

LL_RCC_UART7_CLKSOURCE_LSE

LSE clock used as UART7 clock source

LL_RCC_UART8_CLKSOURCE_PCLK1

PCLK1 clock used as UART8 clock source

LL_RCC_UART8_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART8 clock source

LL_RCC_UART8_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART8 clock source

LL_RCC_UART8_CLKSOURCE_HSI

HSI clock used as UART8 clock source

LL_RCC_UART8_CLKSOURCE_CSI

CSI clock used as UART8 clock source

LL_RCC_UART8_CLKSOURCE_LSE

LSE clock used as UART8 clock source

LL_RCC_UART9_CLKSOURCE_PCLK1

PCLK1 clock used as UART9 clock source

LL_RCC_UART9_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART9 clock source

LL_RCC_UART9_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART9 clock source

LL_RCC_UART9_CLKSOURCE_HSI

HSI clock used as UART9 clock source

LL_RCC_UART9_CLKSOURCE_CSI

CSI clock used as UART9 clock source

LL_RCC_UART9_CLKSOURCE_LSE

LSE clock used as UART9 clock source

LL_RCC_UART12_CLKSOURCE_PCLK1

PCLK1 clock used as UART12 clock source

LL_RCC_UART12_CLKSOURCE_PLL2Q

PLL2 Q clock used as UART12 clock source

LL_RCC_UART12_CLKSOURCE_PLL3Q

PLL3 Q clock used as UART12 clock source

LL_RCC_UART12_CLKSOURCE_HSI

HSI clock used as UART12 clock source

LL_RCC_UART12_CLKSOURCE_CSI

CSI clock used as UART12 clock source

LL_RCC_UART12_CLKSOURCE_LSE

LSE clock used as UART12 clock source

Peripheral USARTx get clock source

LL_RCC_USART1_CLKSOURCE

USART1 Clock source selection

LL_RCC_USART2_CLKSOURCE

USART2 Clock source selection

LL_RCC_USART3_CLKSOURCE

USART3 Clock source selection

LL_RCC_USART6_CLKSOURCE

USART6 Clock source selection

LL_RCC_USART10_CLKSOURCE

USART10 Clock source selection

LL_RCC_USART11_CLKSOURCE

USART11 Clock source selection

Peripheral USARTx clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2

PCLK2 clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_HSI

HSI clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_CSI

CSI clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_LSE

LSE clock used as USART1 clock source

LL_RCC_USART2_CLKSOURCE_PCLK1

PCLK1 clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_HSI

HSI clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_CSI

CSI clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_LSE

LSE clock used as USART2 clock source

LL_RCC_USART3_CLKSOURCE_PCLK1

PCLK1 clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_HSI

HSI clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_CSI

CSI clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_LSE

LSE clock used as USART3 clock source

LL_RCC_USART6_CLKSOURCE_PCLK1

PCLK1 clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_HSI

HSI clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_CSI

CSI clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_LSE

LSE clock used as USART6 clock source

LL_RCC_USART10_CLKSOURCE_PCLK1

PCLK1 clock used as USART10 clock source

LL_RCC_USART10_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART10 clock source

LL_RCC_USART10_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART10 clock source

LL_RCC_USART10_CLKSOURCE_HSI

HSI clock used as USART10 clock source

LL_RCC_USART10_CLKSOURCE_CSI

CSI clock used as USART10 clock source

LL_RCC_USART10_CLKSOURCE_LSE

LSE clock used as USART10 clock source

LL_RCC_USART11_CLKSOURCE_PCLK1

PCLK1 clock used as USART11 clock source

LL_RCC_USART11_CLKSOURCE_PLL2Q

PLL2 Q clock used as USART11 clock source

LL_RCC_USART11_CLKSOURCE_PLL3Q

PLL3 Q clock used as USART11 clock source

LL_RCC_USART11_CLKSOURCE_HSI

HSI clock used as USART11 clock source

LL_RCC_USART11_CLKSOURCE_CSI

CSI clock used as USART11 clock source

LL_RCC_USART11_CLKSOURCE_LSE

LSE clock used as USART11 clock source

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE

USB Clock source selection

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_NONE

No clock used as USB clock source

LL_RCC_USB_CLKSOURCE_PLL1Q

PLL1 Q clock used as USB clock source

LL_RCC_USB_CLKSOURCE_PLL3Q

PLL3 Q clock used as USB clock source

LL_RCC_USB_CLKSOURCE_HSI48

HSI48 clock used as USB clock source

Calculate frequencies

__LL_RCC_CALC_PLL1CLK_P_FREQ

Description:

- Helper macro to calculate the PLL1P clock frequency.

Parameters:

- __INPUTFREQ__: PLL1 Input frequency (based on HSI/HSE/CSI)
- __PLL1M__: parameter can be a value between 1 and 63
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1P__: parameter can be a value between 1 and 128 (odd values not allowed)

Return value:

- PLL1P: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL1CLK_P_FREQ (HSE_VALUE,LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetP ());

__LL_RCC_CALC_PLL1CLK_Q_FREQ

Description:

- Helper macro to calculate the PLL1Q clock frequency.

Parameters:

- __INPUTFREQ__: PLL1 Input frequency (based on HSI/HSE/CSI)
- __PLL1M__: parameter can be a value between 1 and 63
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1Q__: parameter can be a value between 2 and 128

Return value:

- PLL1Q: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL1CLK_Q_FREQ (HSE_VALUE,LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetQ ());

__LL_RCC_CALC_PLL1CLK_R_FREQ

Description:

- Helper macro to calculate the PLL1R clock frequency.

Parameters:

- __INPUTFREQ__: PLL1 Input frequency (based on HSI/HSE/CSI)
- __PLL1M__: parameter can be a value between 1 and 63
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1R__: parameter can be a value between 1 and 128

Return value:

- PLL1R: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL1CLK_R_FREQ (HSE_VALUE,LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetN ());

__LL_RCC_CALC_PLL2CLK_P_FREQ

Description:

- Helper macro to calculate the PLL2P clock frequency.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSI/HSE/CSI)
- __PLL2M__: parameter can be a value between 1 and 63
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2P__: parameter can be a value between 2 and 128

Return value:

- PLL2P: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_P_FREQ (HSE_ALUE,LL_RCC_PLL2_GetM (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetP ());

__LL_RCC_CALC_PLL2CLK_Q_FREQ

Description:

- Helper macro to calculate the PLL2Q clock frequency.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on HSI/HSE/CSI)
- __PLL2M__: parameter can be a value between 1 and 63
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2Q__: parameter can be a value between 1 and 128

Return value:

- PLL2Q: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_Q_FREQ (HSE_VALUE,LL_RCC_PLL2_GetM (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetQ ());

__LL_RCC_CALC_PLL2CLK_R_FREQ

Description:

- Helper macro to calculate the PLL2R clock frequency.

Parameters:

- __INPUTFREQ__: PLL2 Input frequency (based on HSI/HSE/CSI)
- __PLL2M__: parameter can be a value between 1 and 63
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2R__: parameter can be a value between 1 and 128

Return value:

- PLL2R: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_R_FREQ (HSE_VALUE,LL_RCC_PLL2_GetM (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetR ());

__LL_RCC_CALC_PLL3CLK_P_FREQ

Description:

- Helper macro to calculate the PLL3P clock frequency.

Parameters:

- __INPUTFREQ__: PLL3 Input frequency (based on HSI/HSE/CSI)
- __PLL3M__: parameter can be a value between 1 and 63
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3P__: parameter can be a value between 2 and 128

Return value:

- PLL3P: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL3CLK_P_FREQ (HSE_VALUE,LL_RCC_PLL3_GetM (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetP ());

__LL_RCC_CALC_PLL3CLK_Q_FREQ

Description:

- Helper macro to calculate the PLL3 frequency.

Parameters:

- __INPUTFREQ__: PLL3 Input frequency (based on HSI/HSE/CSI)
- __PLL3M__: parameter can be a value between 1 and 63
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3Q__: parameter can be a value between 1 and 128

Return value:

- PLL3Q: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL3CLK_Q_FREQ (HSE_VALUE,LL_RCC_PLL3_GetM (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetQ ());

__LL_RCC_CALC_PLL3CLK_R_FREQ

Description:

- Helper macro to calculate the PLL3 frequency.

Parameters:

- __INPUTFREQ__: PLL3 Input frequency (based on HSI/HSE/CSI)
- __PLL3M__: parameter can be a value between 1 and 63
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3R__: parameter can be a value between 1 and 128

Return value:

- PLL3R: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL3CLK_R_FREQ (HSE_VALUE,LL_RCC_PLL3_GetM (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetR ());

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on HSI/HSE/CSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (APB1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (APB2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

__LL_RCC_CALC_PCLK3_FREQ

Description:

- Helper macro to calculate the PCLK3 frequency (APB3)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB3PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Return value:

- PCLK3: clock frequency (in Hz)

__LL_RCC_CALC_HSI_FREQ

Description:

- Helper macro to calculate the HSI frequency.

Parameters:

- __HSIDIV__: This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8

Return value:

- HSI: clock frequency (in Hz)

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

110 LL RNG Generic Driver

110.1 RNG Firmware driver registers structures

110.1.1 LL_RNG_InitTypeDef

LL_RNG_InitTypeDef is defined in the stm32h5xx_ll_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***. This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

110.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

110.2.1 Detailed description of functions

LL_RNG_Enable

Function name

__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name

__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabled (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Enable Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Disable RNG Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (const RNG_TypeDef * RNGx)

Function description

Check if RNG Clock Error Detection is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_EnableCondReset

Function name

__STATIC_INLINE void LL_RNG_EnableCondReset (RNG_TypeDef * RNGx)

Function description

Set RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_EnableCondReset

LL_RNG_DisableCondReset

Function name

__STATIC_INLINE void LL_RNG_DisableCondReset (RNG_TypeDef * RNGx)

Function description

Reset RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_DisableCondReset

LL_RNG_IsEnabledCondReset

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledCondReset (const RNG_TypeDef * RNGx)

Function description

Check if RNG Conditioning Soft Reset bit is set.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_IsEnabledCondReset

LL_RNG_ConfigLock

Function name

__STATIC_INLINE void LL_RNG_ConfigLock (RNG_TypeDef * RNGx)

Function description

Enable RNG Config Lock.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_ConfigLock

LL_RNG_IsConfigLocked

Function name

__STATIC_INLINE uint32_t LL_RNG_IsConfigLocked (const RNG_TypeDef * RNGx)

Function description

Check if RNG Config Lock is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_IsConfigLocked

LL_RNG_EnableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_EnableNistCompliance (RNG_TypeDef * RNGx)

Function description

Enable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_EnableNistCompliance

LL_RNG_DisableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_DisableNistCompliance (RNG_TypeDef * RNGx)

Function description

Disable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_DisableNistCompliance

LL_RNG_IsEnabledNistCompliance

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledNistCompliance (const RNG_TypeDef * RNGx)

Function description

Check if NIST Compliance is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_IsEnabledNistCompliance

LL_RNG_SetConfig1

Function name

__STATIC_INLINE void LL_RNG_SetConfig1 (RNG_TypeDef * RNGx, uint32_t Config1)

Function description

Set RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config1:** Value between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_SetConfig1

LL_RNG_GetConfig1

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig1 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 6 bits : Value between 0 and 0x3F

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_GetConfig1

LL_RNG_SetConfig2

Function name

__STATIC_INLINE void LL_RNG_SetConfig2 (RNG_TypeDef * RNGx, uint32_t Config2)

Function description

Set RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config2:** Value between 0 and 0x7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_SetConfig2

LL_RNG_GetConfig2

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig2 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 3 bits : Value between 0 and 0x7

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_GetConfig2

LL_RNG_SetConfig3

Function name

__STATIC_INLINE void LL_RNG_SetConfig3 (RNG_TypeDef * RNGx, uint32_t Config3)

Function description

Set RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config3:** Value between 0 and 0xF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_SetConfig3

LL_RNG_GetConfig3

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig3 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 4 bits : Value between 0 and 0xF

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_GetConfig3

LL_RNG_SetClockDivider

Function name

__STATIC_INLINE void LL_RNG_SetClockDivider (RNG_TypeDef * RNGx, uint32_t Divider)

Function description

Set RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance
- **Divider:** can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_SetClockDivider

LL_RNG_GetClockDivider

Function name

__STATIC_INLINE uint32_t LL_RNG_GetClockDivider (const RNG_TypeDef * RNGx)

Function description

Get RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_GetClockDivider

LL_RNG_IsActiveFlag_DRDY

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (const RNG_TypeDef * RNGx)

Function description

Indicate if the RNG Data ready Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Clear Clock Error interrupt Status (CEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Clear Seed Error interrupt Status (SEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name

__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name

__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name

__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (const RNG_TypeDef * RNGx)

Function description

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_EnableArdis

Function name

__STATIC_INLINE void LL_RNG_EnableArdis (RNG_TypeDef * RNGx)

Function description

Enable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_EnableArdis

LL_RNG_DisableArdis

Function name

__STATIC_INLINE void LL_RNG_DisableArdis (RNG_TypeDef * RNGx)

Function description

Disable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_DisableArdis

LL_RNG_IsEnabledArdis

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledArdis (const RNG_TypeDef * RNGx)

Function description

Check if RNG Auto reset is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_IsEnabledArdis

LL_RNG_SetHealthConfig

Function name

__STATIC_INLINE void LL_RNG_SetHealthConfig (RNG_TypeDef * RNGx, uint32_t HTCFG)

Function description

Set RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance
- **HTCFG:** can be values of 32 bits

Return values

- **None:**

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_SetHealthConfig

LL_RNG_GetHealthConfig

Function name

__STATIC_INLINE uint32_t LL_RNG_GetHealthConfig (const RNG_TypeDef * RNGx)

Function description

Get RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance

Return values

- **Return:** 32-bit RNG Health Test configuration

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_GetHealthConfig

LL_RNG_Init

Function name

ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Initialize RNG registers according to the specified parameters in RNG_InitStruct.

Parameters

- **RNGx:** RNG Instance
- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are initialized according to RNG_InitStruct content
  - ERROR: not applicable

LL_RNG_StructInit

Function name

void LL_RNG_StructInit (LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Set each LL_RNG_InitTypeDef field to default value.

Parameters

- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_RNG_DeInit

Function name

ErrorStatus LL_RNG_DeInit (const RNG_TypeDef * RNGx)

Function description

De-initialize RNG registers (Registers restored to their default values).

Parameters

- **RNGx:** RNG Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are de-initialized
  - ERROR: not applicable

110.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

110.3.1 RNG

RNG

Auto reset disable

LL_RNG_ARDIS_ENABLE

ARDIS enabled automatic reset to clear SECS bit

LL_RNG_ARDIS_DISABLE

ARDIS disabled no automatic reset to clear SECS bit

Clock Error Detection

LL_RNG_CED_ENABLE

Clock error detection enabled

LL_RNG_CED_DISABLE

Clock error detection disabled

Value used to configure an internal

LL_RNG_CLKDIV_BY_1

No clock division

LL_RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

Get Flags Defines

LL_RNG_SR_DRDY

Register contains valid random data

LL_RNG_SR_CECS

Clock error current status

LL_RNG_SR_SECS

Seed error current status

LL_RNG_SR_CEIS

Clock error interrupt status

LL_RNG_SR_SEIS

Seed error interrupt status

IT Defines

LL_RNG_CR_IE

RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

NIST Compliance configuration

LL_RNG_NIST_COMPLIANT

Default NIST compliant configuration

LL_RNG_CUSTOM_NIST

Custom NIST configuration

111 LL RTC Generic Driver

111.1 RTC Firmware driver registers structures

111.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32h5xx_ll_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

111.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32h5xx_ll_rtc.h

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT***This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected.This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected.This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

111.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32h5xx_ll_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Day***

- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***
  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***
  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

111.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32h5xx_ll_rtc.h

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B.
- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

111.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

111.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name

__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)

Function description

Set Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance
- **HourFormat:** This parameter can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (const RTC_TypeDef * RTCx)

Function description

Get Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (const RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)

Function description

Set RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance
- **Output:** This parameter can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (const RTC_TypeDef * RTCx)

Function description

Get RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnableInitMode

Function name

__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)

Function description

Enable initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name

__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)

Function description

Disable initialization mode (Free running mode)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_DisableInitMode

LL_RTC_SetBinaryMode

Function name

__STATIC_INLINE void LL_RTC_SetBinaryMode (RTC_TypeDef * RTCx, uint32_t BinaryMode)

Function description

Set Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance
- **BinaryMode:** can be one of the following values:
  - LL_RTC_BINARY_NONE
  - LL_RTC_BINARY_ONLY
  - LL_RTC_BINARY_MIX

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_SetBinaryMode

LL_RTC_GetBinaryMode

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinaryMode (const RTC_TypeDef * RTCx)

Function description

Get Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_BINARY_NONE
  - LL_RTC_BINARY_ONLY
  - LL_RTC_BINARY_MIX
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_GetBinaryMode

LL_RTC_SetBinMixBCDU

Function name

__STATIC_INLINE void LL_RTC_SetBinMixBCDU (RTC_TypeDef * RTCx, uint32_t BinMixBcdU)

Function description

Set Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance
- **BinMixBcdU:** can be one of the following values:
  - LL_RTC_BINARY_MIX_BCDU_0
  - LL_RTC_BINARY_MIX_BCDU_1
  - LL_RTC_BINARY_MIX_BCDU_2
  - LL_RTC_BINARY_MIX_BCDU_3
  - LL_RTC_BINARY_MIX_BCDU_4
  - LL_RTC_BINARY_MIX_BCDU_5
  - LL_RTC_BINARY_MIX_BCDU_6
  - LL_RTC_BINARY_MIX_BCDU_7

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_SetBinMixBCDU

LL_RTC_GetBinMixBCDU

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinMixBCDU (const RTC_TypeDef * RTCx)

Function description

Get Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_BINARY_MIX_BCDU_0
  - LL_RTC_BINARY_MIX_BCDU_1
  - LL_RTC_BINARY_MIX_BCDU_2
  - LL_RTC_BINARY_MIX_BCDU_3
  - LL_RTC_BINARY_MIX_BCDU_4
  - LL_RTC_BINARY_MIX_BCDU_5
  - LL_RTC_BINARY_MIX_BCDU_6
  - LL_RTC_BINARY_MIX_BCDU_7
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_GetBinMixBCDU

LL_RTC_SetOutputPolarity

Function name

__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)

Function description

Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (const RTC_TypeDef * RTCx)

Function description

Get Output polarity.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Enable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Disable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Shadow registers bypass is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name

__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)

Function description

Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name

__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)

Function description

Disable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)

Function description

Set Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)

Function description

Set Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7F

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7FFF

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_GetSynchPrescaler

LL_RTC_EnableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)

Function description

Enable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)

Function description

Disable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_EnableTamperOutput (RTC_TypeDef * RTCx)

Function description

Enable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- When the tamper output is enabled, all external and internal tamper flags are ORed and routed to the TAMPALRM output.

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_EnableTamperOutput

LL_RTC_DisableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_DisableTamperOutput (RTC_TypeDef * RTCx)

Function description

Disable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_DisableTamperOutput

LL_RTC_IsTamperOutputEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsTamperOutputEnabled (const RTC_TypeDef * RTCx)

Function description

Check if tamper output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_IsTamperOutputEnabled

LL_RTC_EnableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_EnableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Enable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_DisableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_DisableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Disable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_IsAlarmPullUpEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsAlarmPullUpEnabled (const RTC_TypeDef * RTCx)

Function description

Check if internal pull-up in output mode is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_IsAlarmPullUpEnabled

LL_RTC_EnableOutput2

Function name

__STATIC_INLINE void LL_RTC_EnableOutput2 (RTC_TypeDef * RTCx)

Function description

Enable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- RTC_OUT2 mapping depends on both OSEL (LL_RTC_SetAlarmOutEvent) and COE (LL_RTC_CAL_SetOutputFreq) settings.
- RTC_OUT2 is not available ins VBAT mode.

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_EnableOutput2

LL_RTC_DisableOutput2

Function name

__STATIC_INLINE void LL_RTC_DisableOutput2 (RTC_TypeDef * RTCx)

Function description

Disable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_DisableOutput2

LL_RTC_IsOutput2Enabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsOutput2Enabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC_OUT2 output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_IsOutput2Enabled

LL_RTC_TIME_SetFormat

Function name

__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (const RTC_TypeDef * RTCx)

Function description

Get time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name

__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_SetHour
- RTC_TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_GetHour
- RTC_TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name

__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_SetMinute
- RTC_TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_GetMinute
- RTC_TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name

__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_SetSecond
- RTC_TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_GetSecond
- RTC_TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name

__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- TimeFormat and Hours should follow the same format

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_Config
- RTC_TR HT LL_RTC_TIME_Config
- RTC_TR HU LL_RTC_TIME_Config
- RTC_TR MNT LL_RTC_TIME_Config
- RTC_TR MNU LL_RTC_TIME_Config
- RTC_TR ST LL_RTC_TIME_Config
- RTC_TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (const RTC_TypeDef * RTCx)

Function description

Get time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_Get
- RTC_TR HU LL_RTC_TIME_Get
- RTC_TR MNT LL_RTC_TIME_Get
- RTC_TR MNU LL_RTC_TIME_Get
- RTC_TR ST LL_RTC_TIME_Get
- RTC_TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)

Function description

Memorize whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)

Function description

Disable memorization whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC Day Light Saving stored operation has been enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name

__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)

Function description

Subtract 1 hour (winter time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name

__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)

Function description

Add 1 hour (summer time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Sub second value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Notes

- You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.

Reference Manual to LL API cross reference:

- RTC_SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name

__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)

Function description

Synchronize to a remote clock with a high degree of precision.

Parameters

- **RTCx:** RTC Instance
- **ShiftSecond:** This parameter can be one of the following values:
  - LL_RTC_SHIFT_SECOND_DELAY
  - LL_RTC_SHIFT_SECOND_ADVANCE
- **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)

Return values

- **None:**

Notes

- This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- When REFCKON is set, firmware must not write to Shift control register.

Reference Manual to LL API cross reference:

- RTC_SHIFTR ADD1S LL_RTC_TIME_Synchronize
- RTC_SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name

__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)

Function description

Set Year in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_SetYear
- RTC_DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (const RTC_TypeDef * RTCx)

Function description

Get Year in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x99

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_GetYear
- RTC_DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set Week day.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name

__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)

Function description

Set Month in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_SetMonth
- RTC_DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_GetMonth
- RTC_DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_SetDay
- RTC_DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_GetDay
- RTC_DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name

__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)

Function description

Set date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Config
- RTC_DR MT LL_RTC_DATE_Config
- RTC_DR MU LL_RTC_DATE_Config
- RTC_DR DT LL_RTC_DATE_Config
- RTC_DR DU LL_RTC_DATE_Config
- RTC_DR YT LL_RTC_DATE_Config
- RTC_DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (const RTC_TypeDef * RTCx)

Function description

Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Get
- RTC_DR MT LL_RTC_DATE_Get
- RTC_DR MU LL_RTC_DATE_Get
- RTC_DR DT LL_RTC_DATE_Get
- RTC_DR DU LL_RTC_DATE_Get
- RTC_DR YT LL_RTC_DATE_Get
- RTC_DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm A masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm A masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmA Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_SetDay
- RTC_ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_GetDay
- RTC_ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set Alarm A time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Alarm A time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_SetHour
- RTC_ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetHour
- RTC_ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_SetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_GetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_SetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_GetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm A Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR ST LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetTime
- RTC_ALRMAR HU LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNT LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNU LL_RTC_ALMA_GetTime
- RTC_ALRMAR ST LL_RTC_ALMA_GetTime
- RTC_ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_SetBinAutoClr

LL_RTC_ALMA_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_GetBinAutoClr

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm B masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm B masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmB Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_SetDay
- RTC_ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_GetDay
- RTC_ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_SetHour
- RTC_ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetHour
- RTC_ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_SetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_GetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_SetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_GetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR ST LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetTime
- RTC_ALRMBR HU LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNT LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNU LL_RTC_ALMB_GetTime
- RTC_ALRMBR ST LL_RTC_ALMB_GetTime
- RTC_ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL_RTC_ALMB_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_SetBinAutoClr

LL_RTC_ALMB_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_GetBinAutoClr

LL_RTC_ALMB_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)

Function description

Enable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)

Function description

Disable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name

__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)

Function description

Enable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name

__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)

Function description

Disable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name

__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)

Function description

Set Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance
- **Edge:** This parameter can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Timestamp AM/PM notation (AM or 24-hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TS_TIME_FORMAT_AM
  - LL_RTC_TS_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetHour
- RTC_TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR MNT LL_RTC_TS_GetMinute
- RTC_TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR ST LL_RTC_TS_GetSecond
- RTC_TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Timestamp time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetTime
- RTC_TSTR HU LL_RTC_TS_GetTime
- RTC_TSTR MNT LL_RTC_TS_GetTime
- RTC_TSTR MNU LL_RTC_TS_GetTime
- RTC_TSTR ST LL_RTC_TS_GetTime
- RTC_TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR MT LL_RTC_TS_GetMonth
- RTC_TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR DT LL_RTC_TS_GetDay
- RTC_TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (const RTC_TypeDef * RTCx)

Function description

Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of Weekday, Day and Month

Notes

- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetDate
- RTC_TSDR MT LL_RTC_TS_GetDate
- RTC_TSDR MU LL_RTC_TS_GetDate
- RTC_TSDR DT LL_RTC_TS_GetDate
- RTC_TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get time-stamp sub second value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_TSDR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)

Function description

Activate timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)

Function description

Disable timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Enable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Enable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Enable
-

LL_RTC_TAMPER_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Disable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Clear TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Disable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Enable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Notes

- Associated Tamper IT must not enabled when tamper mask is set.

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_EnableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Disable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_DisableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (const RTC_TypeDef * RTCx)

Function description

Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (const RTC_TypeDef * RTCx)

Function description

Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (const RTC_TypeDef * RTCx, uint32_t Duration)

Function description

Set RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance
- **Duration:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (const RTC_TypeDef * RTCx, uint32_t FilterCount)

Function description

Set RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance
- **FilterCount:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (const RTC_TypeDef * RTCx, uint32_t SamplingFreq)

Function description

Set Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance
- **SamplingFreq:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (const RTC_TypeDef * RTCx)

Function description

Get Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMPxTRG LL_RTC_TAMPER_EnableActiveLevel
-

LL_RTC_TAMPER_DisableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMPxTRG LL_RTC_TAMPER_DisableActiveLevel
-

LL_RTC_TAMPER_ITAMP_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Enable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP2E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMPxE.. LL_RTC_TAMPER_ITAMP_Enable
-

LL_RTC_TAMPER_ITAMP_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Disable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP2E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMPxE LL_RTC_TAMPER_ITAMP_Disable
-

LL_RTC_TAMPER_ITAMP_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_EnableEraseBKP (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable backup register erase after internal tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR3 ITAMP1NOER LL_RTC_TAMPER_ITAMP_EnableEraseBKP TAMP_CR3 ITAMP2NOER... LL_RTC_TAMPER_ITAMP_EnableEraseBKP

LL_RTC_TAMPER_ITAMP_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_DisableEraseBKP (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable backup register erase after internal tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR3 ITAMP1NOER LL_RTC_TAMPER_ITAMP_DisableEraseBKP TAMP_CR3 ITAMP2NOER... LL_RTC_TAMPER_ITAMP_DisableEraseBKP

LL_RTC_TAMPER_ATAMP_EnableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableActiveMode (uint32_t Tamper)

Function description

Enable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  - ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-

LL_RTC_TAMPER_ATAMP_DisableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableActiveMode (uint32_t Tamper)

Function description

Disable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  - ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-

LL_RTC_TAMPER_ATAMP_EnableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableFilter (void)

Function description

Enable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_EnableFilter
-

LL_RTC_TAMPER_ATAMP_DisableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableFilter (void)

Function description

Disable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_DisableFilter
-

LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod (uint32_t ActiveOutputChangePeriod)

Function description

Set Active tamper output change period.

Parameters

- **ActiveOutputChangePeriod:** This parameter can be a value from 0 to 7

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod (void)

Function description

Get Active tamper output change period.

Return values

- **Output:** change period. This parameter can be a value from 0 to 7.

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler (uint32_t ActiveAsynvPrescaler)

Function description

Set Active tamper asynchronous prescaler clock selection.

Parameters

- **ActiveAsynvPrescaler:** Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of the following values:
  - ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler (void)

Function description

Get Active tamper asynchronous prescaler clock selection.

Return values

- **One:** of
  - ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_EnableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableOutputSharing (void)

Function description

Enable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_EnableOutputSharing
-

LL_RTC_TAMPER_ATAMP_DisableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableOutputSharing (void)

Function description

Disable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_DisableOutputSharing
-

LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection (uint32_t OutputSelection)

Function description

Set Active tamper shared output selection.

Parameters

- **OutputSelection:** Specifies all the output selection of the Active Tamper. This parameter is a combinasation of the following values: One of
  - ACTIVE TAMPER OUTPUT SELECTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATOSELx LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection
-

LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection (void)

Function description

Get Active tamper shared output selection.

Return values

- **A:** combination of
  - ACTIVE TAMPER OUTPUT SELECTION

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATOSELx LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection
-

LL_RTC_TAMPER_ATAMP_WriteSeed

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_WriteSeed (uint32_t Seed)

Function description

Write active tamper seed.

Parameters

- **Seed:**

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATSEEDR SEED LL_RTC_TAMPER_ATAMP_WriteSeed
-

LL_RTC_IsActiveFlag_ATAMP_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_INITS (void)

Function description

Get active tamper initialization status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR INITS LL_RTC_IsActiveFlag_ATAMP_INITS

LL_RTC_IsActiveFlag_ATAMP_SEEDF

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_SEEDF (void)

Function description

Get active tamper seed running status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR SEEDF LL_RTC_IsActiveFlag_ATAMP_SEEDF

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ICSR WUTWF bit = 1

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (const RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (const RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BKP_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BKP_SetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_SetRegister

LL_RTC_BKP_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BKP_GetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_SetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (const RTC_TypeDef * RTCx)

Function description

Get Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_GetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (const RTC_TypeDef * RTCx)

Function description

Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)

Function description

Set the calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_SetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (const RTC_TypeDef * RTCx)

Function description

Get the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_GetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name

__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)

Function description

Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (const RTC_TypeDef * RTCx)

Function description

Get Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_CAL_LowPower_Enable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Enable (RTC_TypeDef * RTCx)

Function description

Enable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Enable

LL_RTC_CAL_LowPower_Disable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Disable (RTC_TypeDef * RTCx)

Function description

Disable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Disable

LL_RTC_CAL_LowPower_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_LowPower_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Calibration Low Power is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_IsEnabled

LL_RTC_IsActiveFlag_ITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (const RTC_TypeDef * RTCx)

Function description

Get Recalibration pending Flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TSOV

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (const RTC_TypeDef * RTCx)

Function description

Get Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (const RTC_TypeDef * RTCx)

Function description

Get Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_IsActiveFlag_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRU (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR SSRUF LL_RTC_IsActiveFlag_SSRU

LL_RTC_ClearFlag_ITS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)

Function description

Clear Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TSOV

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CWUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Clear Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Clear Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_ClearFlag_SSRU

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_SSRU (RTC_TypeDef * RTCx)

Function description

Clear SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CSSRUF LL_RTC_ClearFlag_SSRU

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (const RTC_TypeDef * RTCx)

Function description

Get Initialization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (const RTC_TypeDef * RTCx)

Function description

Get Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)

Function description

Clear Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (const RTC_TypeDef * RTCx)

Function description

Get Initialization status flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (const RTC_TypeDef * RTCx)

Function description

Get Shift operation pending flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRAM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAM (const RTC_TypeDef * RTCx)

Function description

Get Alarm A masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRAMF LL_RTC_IsActiveFlag_ALRAM

LL_RTC_IsActiveFlag_SSRUM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRUM (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR SSRUMF LL_RTC_IsActiveFlag_SSRUM

LL_RTC_IsActiveFlag_ALRBM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBM (const RTC_TypeDef * RTCx)

Function description

Get Alarm B masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRBMF LL_RTC_IsActiveFlag_ALRBM

LL_RTC_IsActiveFlag_WUTM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTM (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR WUTMF LL_RTC_IsActiveFlag_WUTM

LL_RTC_IsActiveFlag_TSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSMF LL_RTC_IsActiveFlag_TSM

LL_RTC_IsActiveFlag_TSOVM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOVM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSOVMF LL_RTC_IsActiveFlag_TSOVM

LL_RTC_IsActiveFlag_ITSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITSM (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ITSMF LL_RTC_IsActiveFlag_ITSM

LL_RTC_IsActiveFlag_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP4F LL_RTC_IsActiveFlag_TAMP4

LL_RTC_IsActiveFlag_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP5F LL_RTC_IsActiveFlag_TAMP5

LL_RTC_IsActiveFlag_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP6F LL_RTC_IsActiveFlag_TAMP6

LL_RTC_IsActiveFlag_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP7F LL_RTC_IsActiveFlag_TAMP7

LL_RTC_IsActiveFlag_TAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Get tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP8F LL_RTC_IsActiveFlag_TAMP8

LL_RTC_IsActiveFlag_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP1F LL_RTC_IsActiveFlag_ITAMP1

LL_RTC_IsActiveFlag_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP2F LL_RTC_IsActiveFlag_ITAMP2

LL_RTC_IsActiveFlag_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP3F LL_RTC_IsActiveFlag_ITAMP3

LL_RTC_IsActiveFlag_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP4F LL_RTC_IsActiveFlag_ITAMP4

LL_RTC_IsActiveFlag_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP5F LL_RTC_IsActiveFlag_ITAMP5

LL_RTC_IsActiveFlag_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP6F LL_RTC_IsActiveFlag_ITAMP6

LL_RTC_IsActiveFlag_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP7F LL_RTC_IsActiveFlag_ITAMP7

LL_RTC_IsActiveFlag_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP8F LL_RTC_IsActiveFlag_ITAMP8

LL_RTC_IsActiveFlag_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 9 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP9F LL_RTC_IsActiveFlag_ITAMP9

LL_RTC_IsActiveFlag_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 11 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP11F LL_RTC_IsActiveFlag_ITAMP11

LL_RTC_IsActiveFlag_ITAMP12

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 12 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP12F LL_RTC_IsActiveFlag_ITAMP12

LL_RTC_IsActiveFlag_ITAMP13

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 13 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP13F LL_RTC_IsActiveFlag_ITAMP13

LL_RTC_IsActiveFlag_ITAMP15

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP15 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 15 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP15F LL_RTC_IsActiveFlag_ITAMP15

LL_RTC_IsActiveFlag_TAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1M (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP1MF LL_RTC_IsActiveFlag_TAMP1M

LL_RTC_IsActiveFlag_TAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2M (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP2MF LL_RTC_IsActiveFlag_TAMP2M

LL_RTC_IsActiveFlag_TAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3M (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP3MF LL_RTC_IsActiveFlag_TAMP3M

LL_RTC_IsActiveFlag_TAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4M (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP4MF LL_RTC_IsActiveFlag_TAMP4M

LL_RTC_IsActiveFlag_TAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5M (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP5MF LL_RTC_IsActiveFlag_TAMP5M

LL_RTC_IsActiveFlag_TAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6M (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP6MF LL_RTC_IsActiveFlag_TAMP6M

LL_RTC_IsActiveFlag_TAMP7M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7M (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP7MF LL_RTC_IsActiveFlag_TAMP7M

LL_RTC_IsActiveFlag_TAMP8M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP8M (const RTC_TypeDef * RTCx)

Function description

Get tamper 8 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP8MF LL_RTC_IsActiveFlag_TAMP8M

LL_RTC_IsActiveFlag_ITAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP1MF LL_RTC_IsActiveFlag_ITAMP1M

LL_RTC_IsActiveFlag_ITAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP2MF LL_RTC_IsActiveFlag_ITAMP2M

LL_RTC_IsActiveFlag_ITAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP3MF LL_RTC_IsActiveFlag_ITAMP3M

LL_RTC_IsActiveFlag_ITAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP4M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP4MF LL_RTC_IsActiveFlag_ITAMP4M

LL_RTC_IsActiveFlag_ITAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP5MF LL_RTC_IsActiveFlag_ITAMP5M

LL_RTC_IsActiveFlag_ITAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP6MF LL_RTC_IsActiveFlag_ITAMP6M

LL_RTC_IsActiveFlag_ITAMP7M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP7M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 7 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP7MF LL_RTC_IsActiveFlag_ITAMP7M

LL_RTC_IsActiveFlag_ITAMP8M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP8MF LL_RTC_IsActiveFlag_ITAMP8M

LL_RTC_IsActiveFlag_ITAMP9M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP9M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 9 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP9MF LL_RTC_IsActiveFlag_ITAMP9M

LL_RTC_IsActiveFlag_ITAMP11M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP11M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 11 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP11MF LL_RTC_IsActiveFlag_ITAMP11M

LL_RTC_IsActiveFlag_ITAMP12M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP12M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 12 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP12MF LL_RTC_IsActiveFlag_ITAMP12M

LL_RTC_IsActiveFlag_ITAMP13M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP13M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 13 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP13MF LL_RTC_IsActiveFlag_ITAMP13M

LL_RTC_IsActiveFlag_ITAMP15M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP15M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 15 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP15MF LL_RTC_IsActiveFlag_ITAMP15M

LL_RTC_ClearFlag_TAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP4F LL_RTC_ClearFlag_TAMP4

LL_RTC_ClearFlag_TAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP5F LL_RTC_ClearFlag_TAMP5

LL_RTC_ClearFlag_TAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP6F LL_RTC_ClearFlag_TAMP6

LL_RTC_ClearFlag_TAMP7

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP7F LL_RTC_ClearFlag_TAMP7

LL_RTC_ClearFlag_TAMP8

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP8F LL_RTC_ClearFlag_TAMP8

LL_RTC_ClearFlag_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP1F LL_RTC_ClearFlag_ITAMP1

LL_RTC_ClearFlag_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP2F LL_RTC_ClearFlag_ITAMP2

LL_RTC_ClearFlag_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP3F LL_RTC_ClearFlag_ITAMP3

LL_RTC_ClearFlag_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP4F LL_RTC_ClearFlag_ITAMP4

LL_RTC_ClearFlag_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP5F LL_RTC_ClearFlag_ITAMP5

LL_RTC_ClearFlag_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP6F LL_RTC_ClearFlag_ITAMP6

LL_RTC_ClearFlag_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP7F LL_RTC_ClearFlag_ITAMP7

LL_RTC_ClearFlag_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP8F LL_RTC_ClearFlag_ITAMP8

LL_RTC_ClearFlag_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 9 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP9F LL_RTC_ClearFlag_ITAMP9

LL_RTC_ClearFlag_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 11 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP11F LL_RTC_ClearFlag_ITAMP11

LL_RTC_ClearFlag_ITAMP12

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 12 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP12F LL_RTC_ClearFlag_ITAMP12

LL_RTC_ClearFlag_ITAMP13

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 13 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP13F LL_RTC_ClearFlag_ITAMP13

LL_RTC_ClearFlag_ITAMP15

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP15 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 15 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP15F LL_RTC_ClearFlag_ITAMP15

LL_RTC_SetRtcSecure

Function name

__STATIC_INLINE void LL_RTC_SetRtcSecure (RTC_TypeDef * RTCx, uint32_t rtcSecure)

Function description

Set RTC secure level.

Parameters

- **RTCx:** RTC Instance
- **rtcSecure:** This parameter can be a combination of the following values:
  - LL_RTC_SECURE_FULL_YES
  - LL_RTC_SECURE_FULL_NO
  - LL_RTC_SECURE_FEATURE_INIT
  - LL_RTC_SECURE_FEATURE_CAL
  - LL_RTC_SECURE_FEATURE_TS
  - LL_RTC_SECURE_FEATURE_WUT
  - LL_RTC_SECURE_FEATURE_ALRA
  - LL_RTC_SECURE_FEATURE_ALRB

Return values

- **None:**

Notes

- secure features are relevant if LL_RTC_SECURE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INITSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_GetRtcSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcSecure (const RTC_TypeDef * RTCx)

Function description

Get RTC secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  - LL_RTC_SECURE_FULL_YES
  - LL_RTC_SECURE_FULL_NO
  - LL_RTC_SECURE_FEATURE_INIT
  - LL_RTC_SECURE_FEATURE_CAL
  - LL_RTC_SECURE_FEATURE_TS
  - LL_RTC_SECURE_FEATURE_WUT
  - LL_RTC_SECURE_FEATURE_ALRA
  - LL_RTC_SECURE_FEATURE_ALRB

Notes

- Secure features is relevant if LL_RTC_SECURE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INISEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_SetTampSecure

Function name

__STATIC_INLINE void LL_RTC_SetTampSecure (const RTC_TypeDef * RTCx, uint32_t tampSecure)

Function description

Set TAMPER secure level.

Parameters

- **RTCx:** RTC Instance
- **tampSecure:** This parameter can be one of the following values:
  - LL_TAMP_SECURE_FULL_YES
  - LL_TAMP_SECURE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_SetTampSecure

LL_RTC_GetTampSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampSecure (const RTC_TypeDef * RTCx)

Function description

Get TAMPER secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_TAMP_SECURE_FULL_YES
  - LL_TAMP_SECURE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_GetTampSecure

LL_RTC_SetRtcPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetRtcPrivilege (RTC_TypeDef * RTCx, uint32_t rtcPrivilege)

Function description

Set RTC privilege level.

Parameters

- **RTCx:** RTC Instance
- **rtcPrivilege:** This parameter can be a combination of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Return values

- **None:**

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_GetRtcPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcPrivilege (const RTC_TypeDef * RTCx)

Function description

Get RTC privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_SetTampPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetTampPrivilege (const RTC_TypeDef * RTCx, uint32_t tampPrivilege)

Function description

Set TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance
- **tampPrivilege:** This parameter can be one of the following values:
  - LL_TAMP_PRIVILEGE_FULL_YES
  - LL_TAMP_PRIVILEGE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR TAMPPRIV LL_RTC_SetTampPrivilege

LL_RTC_GetTampPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampPrivilege (const RTC_TypeDef * RTCx)

Function description

Get TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_TAMP_PRIVILEGE_FULL_YES
  - LL_TAMP_PRIVILEGE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR TAMPPRIV LL_RTC_GetTampPrivilege

LL_RTC_SetBackupRegisterPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegisterPrivilege (const RTC_TypeDef * RTCx, uint32_t bckupRegisterPrivilege)

Function description

Set Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance
- **bckupRegisterPrivilege:** This parameter can be one of the following values:
  - LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  - LL_RTC_PRIVILEGE_BKUP_ZONE_1
  - LL_RTC_PRIVILEGE_BKUP_ZONE_2
  - LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Return values

- **None:**

Notes

- bckupRegisterPrivilege is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_SetBackupRegisterPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_SetBackupRegisterPrivilege

LL_RTC_GetBackupRegisterPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegisterPrivilege (const RTC_TypeDef * RTCx)

Function description

Get Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  - LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  - LL_RTC_PRIVILEGE_BKUP_ZONE_1
  - LL_RTC_PRIVILEGE_BKUP_ZONE_2
  - LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_GetBackupRegisterPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_GetBackupRegisterPrivilege

LL_RTC_SetBackupRegProtection

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegProtection (const RTC_TypeDef * RTCx, uint32_t startZone2, uint32_t startZone3)

Function description

Set Backup registers protection level.

Parameters

- **RTCx:** RTC Instance
- **startZone2:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...
- **startZone3:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DRx ...

Return values

- **None:**

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection
- zone 1 : start from 0 to startZone2 start value
- zone 2 : start from startZone2 start value to startZone3 start value
- zone 3 : start from to startZone3 to the end of BACKUPREG
- Warning : this parameter is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_SetBackupRegProtection
- TAMP_SECCFGR BKPRWSEC LL_RTC_SetBackupRegProtection

LL_RTC_GetBackupRegProtectionStartZone2

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone2 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 2.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection/non-privile write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPRWSEC LL_RTC_GetBackupRegProtectionStartZone2

LL_RTC_GetBackupRegProtectionStartZone3

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone3 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 3.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_GetBackupRegProtectionStartZone3

LL_RTC_EnableIT_TS

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)

Function description

Enable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)

Function description

Disable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Enable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Disable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Enable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Disable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_EnableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Enable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_EnableIT_SSRU

LL_RTC_DisableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_DisableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Disable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_DisableIT_SSRU

LL_RTC_IsEnabledIT_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (const RTC_TypeDef * RTCx)

Function description

Check if Time-stamp interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (const RTC_TypeDef * RTCx)

Function description

Check if Alarm B interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (const RTC_TypeDef * RTCx)

Function description

Check if Alarm A interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_IsEnabledIT_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_SSRU (const RTC_TypeDef * RTCx)

Function description

Check if SSR Underflow interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_IsEnabledIT_SSRU

LL_RTC_EnableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_DisableIT_TAMP1

LL_RTC_EnableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_DisableIT_TAMP3

LL_RTC_EnableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_EnableIT_TAMP4

LL_RTC_DisableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_DisableIT_TAMP4

LL_RTC_EnableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_EnableIT_TAMP5

LL_RTC_DisableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_DisableIT_TAMP5

LL_RTC_EnableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_EnableIT_TAMP6

LL_RTC_DisableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_DisableIT_TAMP6

LL_RTC_EnableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_EnableIT_TAMP7

LL_RTC_DisableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_DisableIT_TAMP7

LL_RTC_EnableIT_TAMP8

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_EnableIT_TAMP8

LL_RTC_DisableIT_TAMP8

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_DisableIT_TAMP8

LL_RTC_EnableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_EnableIT_ITAMP1

LL_RTC_DisableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_DisableIT_ITAMP1

LL_RTC_EnableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_EnableIT_ITAMP2

LL_RTC_DisableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_DisableIT_ITAMP2

LL_RTC_EnableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_EnableIT_ITAMP3

LL_RTC_DisableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_DisableIT_ITAMP3

LL_RTC_EnableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_EnableIT_ITAMP4

LL_RTC_DisableIT_ITAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_DisableIT_ITAMP4

LL_RTC_EnableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_EnableIT_ITAMP5

LL_RTC_DisableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_DisableIT_ITAMP5

LL_RTC_EnableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_EnableIT_ITAMP6

LL_RTC_DisableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_DisableIT_ITAMP6

LL_RTC_EnableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_EnableIT_ITAMP7

LL_RTC_DisableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_DisableIT_ITAMP7

LL_RTC_EnableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_EnableIT_ITAMP8

LL_RTC_DisableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_DisableIT_ITAMP8

LL_RTC_EnableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_EnableIT_ITAMP9

LL_RTC_DisableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_DisableIT_ITAMP9

LL_RTC_EnableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_EnableIT_ITAMP11

LL_RTC_DisableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_DisableIT_ITAMP11

LL_RTC_EnableIT_ITAMP12

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 12 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP12IE LL_RTC_EnableIT_ITAMP12

LL_RTC_DisableIT_ITAMP12

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 12 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP12IE LL_RTC_DisableIT_ITAMP12

LL_RTC_EnableIT_ITAMP13

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 13 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP13IE LL_RTC_EnableIT_ITAMP13

LL_RTC_DisableIT_ITAMP13

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 13 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP13IE LL_RTC_DisableIT_ITAMP13

LL_RTC_EnableIT_ITAMP15

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP15 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 15 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP15IE LL_RTC_EnableIT_ITAMP15

LL_RTC_DisableIT_ITAMP15

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP15 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 15 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP15IE LL_RTC_DisableIT_ITAMP15

LL_RTC_IsEnabledIT_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_IsEnabledIT_TAMP4

LL_RTC_IsEnabledIT_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_IsEnabledIT_TAMP5

LL_RTC_IsEnabledIT_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_IsEnabledIT_TAMP6

LL_RTC_IsEnabledIT_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_IsEnabledIT_TAMP7

LL_RTC_IsEnabledIT_TAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 8 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_IsEnabledIT_TAMP8

LL_RTC_IsEnabledIT_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_IsEnabledIT_ITAMP1

LL_RTC_IsEnabledIT_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_IsEnabledIT_ITAMP2

LL_RTC_IsEnabledIT_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_IsEnabledIT_ITAMP3

LL_RTC_IsEnabledIT_ITAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP4 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP4IE LL_RTC_IsEnabledIT_ITAMP4

LL_RTC_IsEnabledIT_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_IsEnabledIT_ITAMP5

LL_RTC_IsEnabledIT_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_IsEnabledIT_ITAMP6

LL_RTC_IsEnabledIT_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_IsEnabledIT_ITAMP7

LL_RTC_IsEnabledIT_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 8 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_IsEnabledIT_ITAMP8

LL_RTC_IsEnabledIT_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 9 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_IsEnabledIT_ITAMP9

LL_RTC_IsEnabledIT_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 11 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_IsEnabledIT_ITAMP11

LL_RTC_IsEnabledIT_ITAMP12

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 12 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP12IE LL_RTC_IsEnabledIT_ITAMP12

LL_RTC_IsEnabledIT_ITAMP13

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 13 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP13IE LL_RTC_IsEnabledIT_ITAMP13

LL_RTC_IsEnabledIT_ITAMP15

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP15 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 15 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP15IE LL_RTC_IsEnabledIT_ITAMP15

LL_RTC_IncrementMonotonicCounter

Function name

__STATIC_INLINE void LL_RTC_IncrementMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TAMP_COUNT1R COUNT LL_RTC_IncrementMonotonicCounter

LL_RTC_GetMonotonicCounter

Function name

__STATIC_INLINE uint32_t LL_RTC_GetMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Monotonic:** counter value.

Reference Manual to LL API cross reference:

- TAMP_COUNT1R COUNT LL_RTC_GetMonotonicCounter

LL_RTC_DeInit

Function name

ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)

Function description

De-Initializes the RTC registers to their default reset values.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are de-initialized
  - ERROR: RTC registers are not de-initialized

Notes

- This function does not reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name

ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Initializes the RTC registers according to the specified parameters in RTC_InitStruct.

Parameters

- **RTCx:** RTC Instance
- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are initialized
  - ERROR: RTC registers are not initialized

Notes

- The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name

void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Set each LL_RTC_InitTypeDef field to default value.

Parameters

- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_TIME_Init

Function name

ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set the RTC current time.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Time register is configured
  - ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name

void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters

- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_DATE_Init

Function name

ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set the RTC current date.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name

void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)

Parameters

- **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMA_Init

Function name

ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm A.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMA registers are configured
  - ERROR: ALARMA registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name

ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm B.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMB registers are configured
  - ERROR: ALARMB registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name

void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMB_StructInit

Function name

void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_EnterInitMode

Function name

ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description

Enters the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes

- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name

ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description

Exit the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC exited from in Init mode
  - ERROR: Not applicable

Notes

- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name

ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)

Function description

Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are synchronised
  - ERROR: RTC registers are not synchronised

Notes

- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

111.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

111.3.1 RTC

RTC

ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

ACTIVE TAMPER MODE

LL_RTC_TAMPER_ATAMP_TAMP1AM

Tamper 1 is active

LL_RTC_TAMPER_ATAMP_TAMP2AM

Tamper 2 is active

LL_RTC_TAMPER_ATAMP_TAMP3AM

Tamper 3 is active

LL_RTC_TAMPER_ATAMP_TAMP4AM

Tamper 4 is active

LL_RTC_TAMPER_ATAMP_TAMP5AM

Tamper 5 is active

LL_RTC_TAMPER_ATAMP_TAMP6AM

Tamper 6 is active

LL_RTC_TAMPER_ATAMP_TAMP7AM

Tamper 7 is active

LL_RTC_TAMPER_ATAMP_TAMP8AM

Tamper 8 is active

ACTIVE TAMPER OUTPUT SELECTION

LL_RTC_TAMPER_ATAMP1IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP8OUT

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM is push-pull output

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM is open-drain output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE

No masks applied on Alarm A

LL_RTC_ALMA_MASK_DATEWEEKDAY

Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS

Hours do not care in Alarm A comparison

LL_RTC_ALMA_MASK_MINUTES

Minutes do not care in Alarm A comparison

LL_RTC_ALMA_MASK_SECONDS

Seconds do not care in Alarm A comparison

LL_RTC_ALMA_MASK_ALL

Masks all

RTC Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMABINR -> SS[31:0].

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMA_TIME_FORMAT_PM

PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE

Alarm A Date is selected

LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY

Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE

No masks applied on Alarm B

LL_RTC_ALMB_MASK_DATEWEEKDAY

Date/day do not care in Alarm B comparison

LL_RTC_ALMB_MASK_HOURS

Hours do not care in Alarm B comparison

LL_RTC_ALMB_MASK_MINUTES

Minutes do not care in Alarm B comparison

LL_RTC_ALMB_MASK_SECONDS

Seconds do not care in Alarm B comparison

LL_RTC_ALMB_MASK_ALL

Masks all

Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMBBINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMBBINR -> SS[31:0].

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMB_TIME_FORMAT_PM

PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE

Alarm B Date is selected

LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY

Alarm B WeekDay is selected

Calendar second incrementation in Binary mix mode

LL_RTC_BINARY_MIX_BCDU_0

1s calendar increment is generated each time SS[7:0] = 0

LL_RTC_BINARY_MIX_BCDU_1

1s calendar increment is generated each time SS[8:0] = 0

LL_RTC_BINARY_MIX_BCDU_2

1s calendar increment is generated each time SS[9:0] = 0

LL_RTC_BINARY_MIX_BCDU_3

1s calendar increment is generated each time SS[10:0] = 0

LL_RTC_BINARY_MIX_BCDU_4

1s calendar increment is generated each time SS[11:0] = 0

LL_RTC_BINARY_MIX_BCDU_5

1s calendar increment is generated each time SS[12:0] = 0

LL_RTC_BINARY_MIX_BCDU_6

1s calendar increment is generated each time SS[13:0] = 0

LL_RTC_BINARY_MIX_BCDU_7

1s calendar increment is generated each time SS[14:0] = 0

Binary mode (Sub Second Register)

LL_RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled)

LL_RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

LL_RTC_BINARY_MIX

Free running BCD calendar and Binary mode enable

BACKUP

LL_RTC_BKP_NUMBER

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE

No RTCCLK pulses are added

LL_RTC_CALIB_INSERTPULSE_SET

One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration low power

LL_RTC_CALIB_LOWPOWER_NONE

High conso mode

LL_RTC_CALIB_LOWPOWER_SET

Low power mode

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN

Binary data format

LL_RTC_FORMAT_BCD

BCD data format

Get Flags Defines

LL_RTC_SCR_SSRUF

LL_RTC_SCR_ITSF

LL_RTC_SCR_TSOVF

LL_RTC_SCR_TSF

LL_RTC_SCR_WUTF

LL_RTC_SCR_ALRBF

LL_RTC_SCR_ALRAF

LL_RTC_ICSR_RECALPF

LL_RTC_ICSR_BCDU_2

LL_RTC_ICSR_BCDU_1

LL_RTC_ICSR_BCDU_0

LL_RTC_ICSR_BIN_1

LL_RTC_ICSR_BIN_0

LL_RTC_ICSR_INITF

LL_RTC_ICSR_RSF

LL_RTC_ICSR_INITS

LL_RTC_ICSR_SHPF

LL_RTC_ICSR_WUTWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

INTERNAL TAMPER

LL_RTC_TAMPER_ITAMP1

Internal tamper 1: RTC supply voltage monitoring

LL_RTC_TAMPER_ITAMP2

Internal tamper 2: Temperature monitoring

LL_RTC_TAMPER_ITAMP3

Internal tamper 3: LSE monitoring

LL_RTC_TAMPER_ITAMP4

Internal tamper 4: HSE monitoring

LL_RTC_TAMPER_ITAMP5

Internal tamper 5: RTC calendar overflow

LL_RTC_TAMPER_ITAMP6

Internal tamper 6: JTAG/SWD access when RDP > 0

LL_RTC_TAMPER_ITAMP7

Internal tamper 7: Voltage monitoring (VCORE, VREF+), through ADC analog watchdog

LL_RTC_TAMPER_ITAMP8

Internal tamper 8: Monotonic counter overflow

LL_RTC_TAMPER_ITAMP9

Internal tamper 9: Cryptographic IPs fault

LL_RTC_TAMPER_ITAMP11

Internal tamper 11: IWDG reset when tamper flag is set

LL_RTC_TAMPER_ITAMP12

Internal tamper 12: Voltage monitoring (VCORE, VREF+), through ADC analog watchdog

LL_RTC_TAMPER_ITAMP13

Internal tamper 13: Voltage monitoring (VCORE, VREF+), through ADC analog watchdog

LL_RTC_TAMPER_ITAMP15

Internal tamper 15: System fault detection

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

INTERNAL TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_ITAMPER1

Internal tamper 1 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER2

Internal tamper 2 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER3

Internal tamper 3 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER4

Internal tamper 4 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER5

Internal tamper 5 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER6

Internal tamper 6 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER7

Internal tamper 7 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER8

Internal tamper 8 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER9

Internal tamper 9 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER11

Internal tamper 10 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER12

Internal tamper 11 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER13

Internal tamper 12 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_ITAMPER15

Internal tamper 13 event does not erase the backup registers

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH

March

LL_RTC_MONTH_APRIL

April

LL_RTC_MONTH_MAY

May

LL_RTC_MONTH_JUNE

June

LL_RTC_MONTH_JULY

July

LL_RTC_MONTH_AUGUST

August

LL_RTC_MONTH_SEPTEMBER

September

LL_RTC_MONTH_OCTOBER

October

LL_RTC_MONTH_NOVEMBER

November

LL_RTC_MONTH_DECEMBER

December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH

Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW

Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

Privilege Backup register privilege zone

LL_RTC_PRIVILEGE_BKUP_ZONE_NONE

LL_RTC_PRIVILEGE_BKUP_ZONE_1

LL_RTC_PRIVILEGE_BKUP_ZONE_2

LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Privilege rtc features in case of LL_RTC_PRIVILEGE_FULL_NO.

LL_RTC_PRIVILEGE_FEATURE_INIT

Initialization feature is privilege

LL_RTC_PRIVILEGE_FEATURE_CAL

Calibration feature is privilege

LL_RTC_PRIVILEGE_FEATURE_TS

Time stamp feature is privilege

LL_RTC_PRIVILEGE_FEATURE_WUT

Wake up timer feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRA

Alarm A feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRB

Alarm B feature is privilege

Privilege full rtc

LL_RTC_PRIVILEGE_FULL_YES

RTC full privilege

LL_RTC_PRIVILEGE_FULL_NO

RTC is not full privilege, features can be unprivilege. See RTC_LL_EC_PRIVILEGE_RTC_FEATURE

Privilege full tamp

LL_TAMP_PRIVILEGE_FULL_YES

TAMP full privilege

LL_TAMP_PRIVILEGE_FULL_NO

TAMP is not privilege

Secure features rtc in case of LL_RTC_SECURE_FULL_NO.

LL_RTC_SECURE_FEATURE_INIT

Initialization feature is secure

LL_RTC_SECURE_FEATURE_CAL

Calibration feature is secure

LL_RTC_SECURE_FEATURE_TS

Time stamp feature is secure

LL_RTC_SECURE_FEATURE_WUT

Wake up timer feature is secure

LL_RTC_SECURE_FEATURE_ALRA

Alarm A feature is secure

LL_RTC_SECURE_FEATURE_ALRB

Alarm B feature is secure

Secure full rtc

LL_RTC_SECURE_FULL_YES

RTC full secure

LL_RTC_SECURE_FULL_NO

RTC is not full secure, features can be secure. See RTC_LL_EC_SECURE_RTC_FEATURE

Secure tamp

LL_TAMP_SECURE_FULL_YES

TAMP full secure

LL_TAMP_SECURE_FULL_NO

TAMP is not secure

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

Delay (seconds) = SUBFS / (PREDIV_S + 1)

LL_RTC_SHIFT_SECOND_ADVANCE

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1)))

TAMPER

LL_RTC_TAMPER_1

Tamper 1 input detection

LL_RTC_TAMPER_2

Tamper 2 input detection

LL_RTC_TAMPER_3

Tamper 3 input detection

LL_RTC_TAMPER_4

Tamper 4 input detection

LL_RTC_TAMPER_5

Tamper 5 input detection

LL_RTC_TAMPER_6

Tamper 6 input detection

LL_RTC_TAMPER_7

Tamper 7 input detection

LL_RTC_TAMPER_8

Tamper 8 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1

Tamper 1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Tamper 2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Tamper 3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP4

Tamper 4 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP5

Tamper 5 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP6

Tamper 6 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP7

Tamper 7 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP8

Tamper 8 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE

Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1

Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2

Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER3

Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1

Tamper 1 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER2

Tamper 2 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER3

Tamper 3 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER4

Tamper 4 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER5

Tamper 5 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER6

Tamper 6 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER7

Tamper 7 event does not erase the backup registers

LL_RTC_TAMPER_NOERASE_TAMPER8

Tamper 8 event does not erase the backup registers

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPLFREQDIV_16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPLFREQDIV_8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPLFREQDIV_4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPLFREQDIV_2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

LL_RTC_TAMPER_SAMPLFREQDIV_512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

LL_RTC_TAMPER_SAMPLFREQDIV_256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING

RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING

RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24

AM or 24-hour format

LL_RTC_TIME_FORMAT_PM

PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_TS_TIME_FORMAT_PM

PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16

RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8

RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4

RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2

RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE

ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY

Monday

LL_RTC_WEEKDAY_TUESDAY

Tuesday

LL_RTC_WEEKDAY_WEDNESDAY

Wednesday

LL_RTC_WEEKDAY_THURSDAY

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

112 LL SPI Generic Driver

112.1 SPI Firmware driver registers structures

112.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32h5xx_ll_spi.h

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***
  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***
  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***
  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit.This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***
  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

  This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.
- ***uint32_t LL_SPI_InitTypeDef::BitOrder***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.

- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

112.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

112.2.1 Detailed description of functions

LL_SPI_Enable

Function name

__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)

Function description

Enable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name

__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)

Function description

Disable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- When disabling the SPI, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled (const SPI_TypeDef * SPIx)

Function description

Check if SPI peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_IsEnabled

LL_SPI_EnableIOSwap

Function name

__STATIC_INLINE void LL_SPI_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_EnableIOSwap

LL_SPI_DisableIOSwap

Function name

__STATIC_INLINE void LL_SPI_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_DisableIOSwap

LL_SPI_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOSwap (const SPI_TypeDef * SPIx)

Function description

Check if MOSI and MISO pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_IsEnabledIOSwap

LL_SPI_EnableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_EnableGPIOControl

LL_SPI_DisableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_DisableGPIOControl

LL_SPI_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledGPIOControl (const SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_IsEnabledGPIOControl

LL_SPI_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI Mode to Master or Slave.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (const SPI_TypeDef * SPIx)

Function description

Get SPI Mode (Master or Slave)

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_GetMode

LL_SPI_SetMasterSSIdleness

Function name

__STATIC_INLINE void LL_SPI_SetMasterSSIdleness (SPI_TypeDef * SPIx, uint32_t MasterSSIdleness)

Function description

Configure the Idleness applied by master between active edge of SS and first send data.

Parameters

- **SPIx:** SPI Instance
- **MasterSSIdleness:** This parameter can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_SetMasterSSIdleness

LL_SPI_GetMasterSSIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMasterSSIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured Idleness applied by master.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_GetMasterSSIdleness

LL_SPI_SetInterDataIdleness

Function name

__STATIC_INLINE void LL_SPI_SetInterDataIdleness (SPI_TypeDef * SPIx, uint32_t MasterInterDataIdleness)

Function description

Configure the idleness applied by master between data frame.

Parameters

- **SPIx:** SPI Instance
- **MasterInterDataIdleness:** This parameter can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_GetInterDataIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInterDataIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured inter data idleness.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_SetTransferSize

Function name

__STATIC_INLINE void LL_SPI_SetTransferSize (SPI_TypeDef * SPIx, uint32_t Count)

Function description

Set transfer size.

Parameters

- **SPIx:** SPI Instance
- **Count:** 0..0xFFFF

Return values

- **None:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_SetTransferSize

LL_SPI_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferSize (const SPI_TypeDef * SPIx)

Function description

Get transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_GetTransferSize

LL_SPI_EnableIOLock

Function name

__STATIC_INLINE void LL_SPI_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the AF configuration remains locked until a hardware reset occurs. the reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_SPI_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOLock (const SPI_TypeDef * SPIx)

Function description

Check if the AF configuration is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_IsEnabledIOLock

LL_SPI_SetTxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetTxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t TXCRCInitAll)

Function description

Set Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **TXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_SetTxCRCInitPattern

LL_SPI_GetTxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_GetTxCRCInitPattern

LL_SPI_SetRxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetRxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t RXCRCInitAll)

Function description

Set Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **RXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_SetRxCRCInitPattern

LL_SPI_GetRxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_GetRxCRCInitPattern

LL_SPI_SetInternalSSLevel

Function name

__STATIC_INLINE void LL_SPI_SetInternalSSLevel (SPI_TypeDef * SPIx, uint32_t SSLevel)

Function description

Set internal SS input level ignoring what comes from PIN.

Parameters

- **SPIx:** SPI Instance
- **SSLevel:** This parameter can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Return values

- **None:**

Notes

- This configuration has effect only with config LL_SPI_NSS_SOFT

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_SetInternalSSLevel

LL_SPI_GetInternalSSLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInternalSSLevel (const SPI_TypeDef * SPIx)

Function description

Get internal SS input level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_GetInternalSSLevel

LL_SPI_EnableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_EnableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_EnableFullSizeCRC

LL_SPI_DisableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_DisableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_DisableFullSizeCRC

LL_SPI_IsEnabledFullSizeCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledFullSizeCRC (const SPI_TypeDef * SPIx)

Function description

Check if Enable CRC computation on 33/17 bits is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_IsEnabledFullSizeCRC

LL_SPI_SuspendMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_SuspendMasterTransfer (SPI_TypeDef * SPIx)

Function description

Suspend an ongoing transfer for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSUSP LL_SPI_SuspendMasterTransfer

LL_SPI_StartMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_StartMasterTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_StartMasterTransfer

LL_SPI_IsActiveMasterTransfer

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveMasterTransfer (const SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished master transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_IsActiveMasterTransfer

LL_SPI_EnableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_EnableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Enable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_EnableMasterRxAutoSuspend

LL_SPI_DisableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_DisableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Disable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_DisableMasterRxAutoSuspend

LL_SPI_IsEnabledMasterRxAutoSuspend

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledMasterRxAutoSuspend (const SPI_TypeDef * SPIx)

Function description

Check if Master Rx auto suspend is activated.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_IsEnabledMasterRxAutoSuspend

LL_SPI_SetUDRConfiguration

Function name

__STATIC_INLINE void LL_SPI_SetUDRConfiguration (SPI_TypeDef * SPIx, uint32_t UDRConfig)

Function description

Set Underrun behavior.

Parameters

- **SPIx:** SPI Instance
- **UDRConfig:** This parameter can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_SetUDRConfiguration

LL_SPI_GetUDRConfiguration

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRConfiguration (const SPI_TypeDef * SPIx)

Function description

Get Underrun behavior.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_GetUDRConfiguration

LL_SPI_SetStandard

Function name

__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set Serial protocol used.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_SetStandard

LL_SPI_GetStandard

Function name

__STATIC_INLINE uint32_t LL_SPI_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get Serial protocol used.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name

__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description

Set Clock phase.

Parameters

- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (const SPI_TypeDef * SPIx)

Function description

Get Clock phase.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name

__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set Clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get Clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetNSSPolarity

Function name

__STATIC_INLINE void LL_SPI_SetNSSPolarity (SPI_TypeDef * SPIx, uint32_t NSSPolarity)

Function description

Set NSS polarity.

Parameters

- **SPIx:** SPI Instance
- **NSSPolarity:** This parameter can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_SetNSSPolarity

LL_SPI_GetNSSPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSPolarity (const SPI_TypeDef * SPIx)

Function description

Get NSS polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_GetNSSPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t Baudrate)

Function description

Set Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance
- **Baudrate:** This parameter can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_BYPASS
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. SPI BaudRate = fPCLK/Pescaler.

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_SetBaudRatePrescaler
- CFG1 BPASS LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (const SPI_TypeDef * SPIx)

Function description

Get Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_BYPASS
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_GetBaudRatePrescaler
- CFG1 BPASS LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (const SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set Transfer Mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled except for half duplex direction using LL_SPI_SetHalfDuplexDirection.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetTransferDirection
- CFG2 COMM LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (const SPI_TypeDef * SPIx)

Function description

Get Transfer Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetTransferDirection
- CFG2 COMM LL_SPI_GetTransferDirection

LL_SPI_SetHalfDuplexDirection

Function name

__STATIC_INLINE void LL_SPI_SetHalfDuplexDirection (SPI_TypeDef * SPIx, uint32_t HalfDuplexDirection)

Function description

Set direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance
- **HalfDuplexDirection:** This parameter can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetHalfDuplexDirection

LL_SPI_GetHalfDuplexDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetHalfDuplexDirection (const SPI_TypeDef * SPIx)

Function description

Get direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetHalfDuplexDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set Frame Data Size.

Parameters

- **SPIx:** SPI Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (const SPI_TypeDef * SPIx)

Function description

Get Frame Data Size.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_GetDataWidth

LL_SPI_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_SetFIFOThreshold

LL_SPI_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_GetFIFOThreshold

LL_SPI_EnableCRC

Function name

__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name

__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (const SPI_TypeDef * SPIx)

Function description

Check if CRC is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name

__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)

Function description

Set CRC Length.

Parameters

- **SPIx:** SPI Instance
- **CRCLength:** This parameter can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (const SPI_TypeDef * SPIx)

Function description

Get CRC Length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_GetCRCWidth

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_SetNSSMode
- CFG2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (const SPI_TypeDef * SPIx)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_GetNSSMode
- CFG2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (const SPI_TypeDef * SPIx)

Function description

Check if NSS pulse is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_SPI_IsActiveFlag_RXP

LL_SPI_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_SPI_IsActiveFlag_TXP

LL_SPI_IsActiveFlag_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_DXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet, AND enough data to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR DXP LL_SPI_IsActiveFlag_DXP

LL_SPI_IsActiveFlag_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_EOT (const SPI_TypeDef * SPIx)

Function description

Check that end of transfer event occurred.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EOT LL_SPI_IsActiveFlag_EOT

LL_SPI_IsActiveFlag_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXTF (const SPI_TypeDef * SPIx)

Function description

Check that all required data has been filled in the fifo according to transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXTF LL_SPI_IsActiveFlag_TXTF

LL_SPI_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_SPI_IsActiveFlag_UDR

LL_SPI_IsActiveFlag_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (const SPI_TypeDef * SPIx)

Function description

Get CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CRCE LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (const SPI_TypeDef * SPIx)

Function description

Get Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_SPI_IsActiveFlag_FRE

LL_SPI_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_SUSP (const SPI_TypeDef * SPIx)

Function description

Check if a suspend operation is done.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR SUSP LL_SPI_IsActiveFlag_SUSP

LL_SPI_IsActiveFlag_TXC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXC (const SPI_TypeDef * SPIx)

Function description

Check if last TxFIFO or CRC frame transmission is completed.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXC LL_SPI_IsActiveFlag_TXC

LL_SPI_IsActiveFlag_RXWNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXWNE (const SPI_TypeDef * SPIx)

Function description

Check if at least one 32-bit data is available in RxFIFO.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXWNE LL_SPI_IsActiveFlag_RXWNE

LL_SPI_GetRemainingDataFrames

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRemainingDataFrames (const SPI_TypeDef * SPIx)

Function description

Get number of data framed remaining in current TSIZE.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- SR CTSIZE LL_SPI_GetRemainingDataFrames

LL_SPI_GetRxFIFOPackingLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOPackingLevel (const SPI_TypeDef * SPIx)

Function description

Get RxFIFO packing Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_0PACKET
  - LL_SPI_RX_FIFO_1PACKET
  - LL_SPI_RX_FIFO_2PACKET
  - LL_SPI_RX_FIFO_3PACKET

Reference Manual to LL API cross reference:

- SR RXPLVL LL_SPI_GetRxFIFOPackingLevel

LL_SPI_ClearFlag_EOT

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_EOT (SPI_TypeDef * SPIx)

Function description

Clear End Of Transfer flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR EOTC LL_SPI_ClearFlag_EOT

LL_SPI_ClearFlag_TXTF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_TXTF (SPI_TypeDef * SPIx)

Function description

Clear TXTF flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TXTFC LL_SPI_ClearFlag_TXTF

LL_SPI_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_SPI_ClearFlag_UDR

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_CRCERR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Clear CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CRCEC LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)

Function description

Clear Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR MODFC LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_SPI_ClearFlag_FRE

LL_SPI_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_SUSP (SPI_TypeDef * SPIx)

Function description

Clear SUSP flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR SUSPC LL_SPI_ClearFlag_SUSP

LL_SPI_EnableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_EnableIT_RXP

LL_SPI_EnableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_EnableIT_TXP

LL_SPI_EnableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_DXP (SPI_TypeDef * SPIx)

Function description

Enable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_EnableIT_DXP

LL_SPI_EnableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_EnableIT_EOT (SPI_TypeDef * SPIx)

Function description

Enable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_EnableIT_EOT

LL_SPI_EnableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Enable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_EnableIT_TXTF

LL_SPI_EnableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_EnableIT_UDR

LL_SPI_EnableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_EnableIT_OVR

LL_SPI_EnableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Enable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_EnableIT_CRCERR

LL_SPI_EnableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_EnableIT_FRE

LL_SPI_EnableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_MODF (SPI_TypeDef * SPIx)

Function description

Enable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_EnableIT_MODF

LL_SPI_DisableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_DisableIT_RXP

LL_SPI_DisableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_DisableIT_TXP

LL_SPI_DisableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_DXP (SPI_TypeDef * SPIx)

Function description

Disable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_DisableIT_DXP

LL_SPI_DisableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_DisableIT_EOT (SPI_TypeDef * SPIx)

Function description

Disable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_DisableIT_EOT

LL_SPI_DisableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Disable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_DisableIT_TXTF

LL_SPI_DisableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_DisableIT_UDR

LL_SPI_DisableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_DisableIT_OVR

LL_SPI_DisableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Disable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_DisableIT_CRCERR

LL_SPI_DisableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_DisableIT_FRE

LL_SPI_DisableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_MODF (SPI_TypeDef * SPIx)

Function description

Disable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_DisableIT_MODF

LL_SPI_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXP (const SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_IsEnabledIT_RXP

LL_SPI_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXP (const SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_IsEnabledIT_TXP

LL_SPI_IsEnabledIT_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_DXP (const SPI_TypeDef * SPIx)

Function description

Check if Duplex Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_IsEnabledIT_DXP

LL_SPI_IsEnabledIT_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_EOT (const SPI_TypeDef * SPIx)

Function description

Check if End Of Transfer IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_IsEnabledIT_EOT

LL_SPI_IsEnabledIT_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXTF (const SPI_TypeDef * SPIx)

Function description

Check if TXTF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_IsEnabledIT_TXTF

LL_SPI_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_UDR (const SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_IsEnabledIT_UDR

LL_SPI_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_OVR (const SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_IsEnabledIT_OVR

LL_SPI_IsEnabledIT_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_CRCERR (const SPI_TypeDef * SPIx)

Function description

Check if CRC Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_IsEnabledIT_CRCERR

LL_SPI_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_FRE (const SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_IsEnabledIT_FRE

LL_SPI_IsEnabledIT_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_MODF (const SPI_TypeDef * SPIx)

Function description

Check if MODF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_IsEnabledIT_MODF

LL_SPI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetTxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetTxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDR LL_SPI_DMA_GetTxRegAddr

LL_SPI_DMA_GetRxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RXDR RXDR LL_SPI_DMA_GetRxRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData16

LL_SPI_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_SPI_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData32

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData16

LL_SPI_TransmitData32

Function name

__STATIC_INLINE void LL_SPI_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData32

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (const SPI_TypeDef * SPIx)

Function description

Get polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_SetUDRPattern

Function name

__STATIC_INLINE void LL_SPI_SetUDRPattern (SPI_TypeDef * SPIx, uint32_t Pattern)

Function description

Set the underrun pattern.

Parameters

- **SPIx:** SPI Instance
- **Pattern:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_SetUDRPattern

LL_SPI_GetUDRPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRPattern (const SPI_TypeDef * SPIx)

Function description

Get the underrun pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_GetUDRPattern

LL_SPI_GetRxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

112.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

112.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_BYPASS

LL_SPI_BAUDRATEPRESCALER_DIV2

LL_SPI_BAUDRATEPRESCALER_DIV4

LL_SPI_BAUDRATEPRESCALER_DIV8

LL_SPI_BAUDRATEPRESCALER_DIV16

LL_SPI_BAUDRATEPRESCALER_DIV32

LL_SPI_BAUDRATEPRESCALER_DIV64

LL_SPI_BAUDRATEPRESCALER_DIV128

LL_SPI_BAUDRATEPRESCALER_DIV256

Bit Order

LL_SPI_LSB_FIRST

LL_SPI_MSB_FIRST

CRC

LL_SPI_CRC_4BIT

LL_SPI_CRC_5BIT

LL_SPI_CRC_6BIT

LL_SPI_CRC_7BIT

LL_SPI_CRC_8BIT

LL_SPI_CRC_9BIT

LL_SPI_CRC_10BIT

LL_SPI_CRC_11BIT

LL_SPI_CRC_12BIT

LL_SPI_CRC_13BIT

LL_SPI_CRC_14BIT

LL_SPI_CRC_15BIT

LL_SPI_CRC_16BIT

LL_SPI_CRC_17BIT

LL_SPI_CRC_18BIT

LL_SPI_CRC_19BIT

LL_SPI_CRC_20BIT

LL_SPI_CRC_21BIT

LL_SPI_CRC_22BIT

LL_SPI_CRC_23BIT

LL_SPI_CRC_24BIT

LL_SPI_CRC_25BIT

LL_SPI_CRC_26BIT

LL_SPI_CRC_27BIT

LL_SPI_CRC_28BIT

LL_SPI_CRC_29BIT

LL_SPI_CRC_30BIT

LL_SPI_CRC_31BIT

LL_SPI_CRC_32BIT

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

Data Width

LL_SPI_DATAWIDTH_4BIT

LL_SPI_DATAWIDTH_5BIT

LL_SPI_DATAWIDTH_6BIT

LL_SPI_DATAWIDTH_7BIT

LL_SPI_DATAWIDTH_8BIT

LL_SPI_DATAWIDTH_9BIT

LL_SPI_DATAWIDTH_10BIT

LL_SPI_DATAWIDTH_11BIT

LL_SPI_DATAWIDTH_12BIT

LL_SPI_DATAWIDTH_13BIT

LL_SPI_DATAWIDTH_14BIT

LL_SPI_DATAWIDTH_15BIT

LL_SPI_DATAWIDTH_16BIT

LL_SPI_DATAWIDTH_17BIT

LL_SPI_DATAWIDTH_18BIT

LL_SPI_DATAWIDTH_19BIT

LL_SPI_DATAWIDTH_20BIT

LL_SPI_DATAWIDTH_21BIT

LL_SPI_DATAWIDTH_22BIT

LL_SPI_DATAWIDTH_23BIT

LL_SPI_DATAWIDTH_24BIT

LL_SPI_DATAWIDTH_25BIT

LL_SPI_DATAWIDTH_26BIT

LL_SPI_DATAWIDTH_27BIT

LL_SPI_DATAWIDTH_28BIT

LL_SPI_DATAWIDTH_29BIT

LL_SPI_DATAWIDTH_30BIT

LL_SPI_DATAWIDTH_31BIT

LL_SPI_DATAWIDTH_32BIT

FIFO Threshold

LL_SPI_FIFO_TH_01DATA

LL_SPI_FIFO_TH_02DATA

LL_SPI_FIFO_TH_03DATA

LL_SPI_FIFO_TH_04DATA

LL_SPI_FIFO_TH_05DATA

LL_SPI_FIFO_TH_06DATA

LL_SPI_FIFO_TH_07DATA

LL_SPI_FIFO_TH_08DATA

LL_SPI_FIFO_TH_09DATA

LL_SPI_FIFO_TH_10DATA

LL_SPI_FIFO_TH_11DATA

LL_SPI_FIFO_TH_12DATA

LL_SPI_FIFO_TH_13DATA

LL_SPI_FIFO_TH_14DATA

LL_SPI_FIFO_TH_15DATA

LL_SPI_FIFO_TH_16DATA

Get Flags Defines

LL_SPI_SR_RXP

LL_SPI_SR_TXP

LL_SPI_SR_DXP

LL_SPI_SR_EOT

LL_SPI_SR_TXTF

LL_SPI_SR_UDR

LL_SPI_SR_CRCE

LL_SPI_SR_MODF

LL_SPI_SR_OVR

LL_SPI_SR_TIFRE

LL_SPI_SR_SUSP

LL_SPI_SR_TXC

LL_SPI_SR_RXWNE

Master Inter-Data Idleness

LL_SPI_ID_IDLENESS_00CYCLE

LL_SPI_ID_IDLENESS_01CYCLE

LL_SPI_ID_IDLENESS_02CYCLE

LL_SPI_ID_IDLENESS_03CYCLE

LL_SPI_ID_IDLENESS_04CYCLE

LL_SPI_ID_IDLENESS_05CYCLE

LL_SPI_ID_IDLENESS_06CYCLE

LL_SPI_ID_IDLENESS_07CYCLE

LL_SPI_ID_IDLENESS_08CYCLE

LL_SPI_ID_IDLENESS_09CYCLE

LL_SPI_ID_IDLENESS_10CYCLE

LL_SPI_ID_IDLENESS_11CYCLE

LL_SPI_ID_IDLENESS_12CYCLE

LL_SPI_ID_IDLENESS_13CYCLE

LL_SPI_ID_IDLENESS_14CYCLE

LL_SPI_ID_IDLENESS_15CYCLE

IT Defines

LL_SPI_IER_RXPIE

LL_SPI_IER_TXPIE

LL_SPI_IER_DXPIE

LL_SPI_IER_EOTIE

LL_SPI_IER_TXTFIE

LL_SPI_IER_UDRIE

LL_SPI_IER_OVRIE

LL_SPI_IER_CRCEIE

LL_SPI_IER_TIFREIE

LL_SPI_IER_MODFIE

Mode

LL_SPI_MODE_MASTER

LL_SPI_MODE_SLAVE

NSS Mode

LL_SPI_NSS_SOFT

LL_SPI_NSS_HARD_INPUT

LL_SPI_NSS_HARD_OUTPUT

NSS Polarity

LL_SPI_NSS_POLARITY_LOW

LL_SPI_NSS_POLARITY_HIGH

Phase

LL_SPI_PHASE_1EDGE

LL_SPI_PHASE_2EDGE

Polarity

LL_SPI_POLARITY_LOW

LL_SPI_POLARITY_HIGH

Protocol

LL_SPI_PROTOCOL_MOTOROLA

LL_SPI_PROTOCOL_TI

RXCRC Init All

LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

RxFIFO Packing LeVel

LL_SPI_RX_FIFO_0PACKET

LL_SPI_RX_FIFO_1PACKET

LL_SPI_RX_FIFO_2PACKET

LL_SPI_RX_FIFO_3PACKET

SS Idleness

LL_SPI_SS_IDLENESS_00CYCLE

LL_SPI_SS_IDLENESS_01CYCLE

LL_SPI_SS_IDLENESS_02CYCLE

LL_SPI_SS_IDLENESS_03CYCLE

LL_SPI_SS_IDLENESS_04CYCLE

LL_SPI_SS_IDLENESS_05CYCLE

LL_SPI_SS_IDLENESS_06CYCLE

LL_SPI_SS_IDLENESS_07CYCLE

LL_SPI_SS_IDLENESS_08CYCLE

LL_SPI_SS_IDLENESS_09CYCLE

LL_SPI_SS_IDLENESS_10CYCLE

LL_SPI_SS_IDLENESS_11CYCLE

LL_SPI_SS_IDLENESS_12CYCLE

LL_SPI_SS_IDLENESS_13CYCLE

LL_SPI_SS_IDLENESS_14CYCLE

LL_SPI_SS_IDLENESS_15CYCLE

SS Level

LL_SPI_SS_LEVEL_HIGH

LL_SPI_SS_LEVEL_LOW

Transfer Mode

LL_SPI_FULL_DUPLEX

LL_SPI_SIMPLEX_TX

LL_SPI_SIMPLEX_RX

LL_SPI_HALF_DUPLEX_RX

LL_SPI_HALF_DUPLEX_TX

TXCRC Init All

LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

UDR Config Register

LL_SPI_UDR_CONFIG_REGISTER_PATTERN

LL_SPI_UDR_CONFIG_LAST_RECEIVED

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

113 LL SYSTEM Generic Driver

113.1 SYSTEM Firmware driver API description

The following section lists the various functions of the SYSTEM library.

113.1.1 Detailed description of functions

LL_SBS_SetPHYInterface

Function name

__STATIC_INLINE void LL_SBS_SetPHYInterface (uint32_t Interface)

Function description

Select Ethernet PHY interface.

Parameters

- **Interface:** This parameter can be one of the following values:
  - LL_SBS_ETH_MII
  - LL_SBS_ETH_RMII

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR EPIS_SEL LL_SBS_SetPHYInterface

LL_SBS_GetPHYInterface

Function name

__STATIC_INLINE uint32_t LL_SBS_GetPHYInterface (void)

Function description

Get Ethernet PHY interface.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_ETH_MII
  - LL_SBS_ETH_RMII

Reference Manual to LL API cross reference:

- PMCR EPIS_SEL LL_SBS_GetPHYInterface

LL_SBS_EnableFastModePlus

Function name

__STATIC_INLINE void LL_SBS_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SBS_FASTMODEPLUS_PB6
  - LL_SBS_FASTMODEPLUS_PB7
  - LL_SBS_FASTMODEPLUS_PB8
  - LL_SBS_FASTMODEPLUS_PB9

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR PBx_FMP LL_SBS_EnableFastModePlus
- PMCR PBx_FMP LL_SBS_EnableFastModePlus

LL_SBS_DisableFastModePlus

Function name

__STATIC_INLINE void LL_SBS_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SBS_FASTMODEPLUS_PB6
  - LL_SBS_FASTMODEPLUS_PB7
  - LL_SBS_FASTMODEPLUS_PB8
  - LL_SBS_FASTMODEPLUS_PB9

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR PBx_FMP LL_SBS_DisableFastModePlus
- PMCR PBx_FMP LL_SBS_DisableFastModePlus

LL_SBS_EnableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_IOC (void)

Function description

Enable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_0 LL_SBS_EnableIT_FPU_IOC

LL_SBS_EnableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_DZC (void)

Function description

Enable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_1 LL_SBS_EnableIT_FPU_DZC

LL_SBS_EnableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_UFC (void)

Function description

Enable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_2 LL_SBS_EnableIT_FPU_UFC

LL_SBS_EnableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_OFC (void)

Function description

Enable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_3 LL_SBS_EnableIT_FPU_OFC

LL_SBS_EnableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_IDC (void)

Function description

Enable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_4 LL_SBS_EnableIT_FPU_IDC

LL_SBS_EnableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SBS_EnableIT_FPU_IXC (void)

Function description

Enable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_5 LL_SBS_EnableIT_FPU_IXC

LL_SBS_DisableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_IOC (void)

Function description

Disable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_0 LL_SBS_DisableIT_FPU_IOC

LL_SBS_DisableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_DZC (void)

Function description

Disable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_1 LL_SBS_DisableIT_FPU_DZC

LL_SBS_DisableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_UFC (void)

Function description

Disable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_2 LL_SBS_DisableIT_FPU_UFC

LL_SBS_DisableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_OFC (void)

Function description

Disable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_3 LL_SBS_DisableIT_FPU_OFC

LL_SBS_DisableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_IDC (void)

Function description

Disable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_4 LL_SBS_DisableIT_FPU_IDC

LL_SBS_DisableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SBS_DisableIT_FPU_IXC (void)

Function description

Disable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_5 LL_SBS_DisableIT_FPU_IXC

LL_SBS_IsEnabledIT_FPU_IOC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_IOC (void)

Function description

Check if Floating Point Unit Invalid operation Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_0 LL_SBS_IsEnabledIT_FPU_IOC

LL_SBS_IsEnabledIT_FPU_DZC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_DZC (void)

Function description

Check if Floating Point Unit Divide-by-zero Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_1 LL_SBS_IsEnabledIT_FPU_DZC

LL_SBS_IsEnabledIT_FPU_UFC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_UFC (void)

Function description

Check if Floating Point Unit Underflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_2 LL_SBS_IsEnabledIT_FPU_UFC

LL_SBS_IsEnabledIT_FPU_OFC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_OFC (void)

Function description

Check if Floating Point Unit Overflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_3 LL_SBS_IsEnabledIT_FPU_OFC

LL_SBS_IsEnabledIT_FPU_IDC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_IDC (void)

Function description

Check if Floating Point Unit Input denormal Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_4 LL_SBS_IsEnabledIT_FPU_IDC

LL_SBS_IsEnabledIT_FPU_IXC

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabledIT_FPU_IXC (void)

Function description

Check if Floating Point Unit Inexact Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FPUIMR FPU_IE_5 LL_SBS_IsEnabledIT_FPU_IXC

LL_SBS_SetTIMBreakInputs

Function name

__STATIC_INLINE void LL_SBS_SetTIMBreakInputs (uint32_t Break)

Function description

Set connections to TIM1/8/15/16/17 Break inputs.

Parameters

- **Break:** This parameter can be a combination of the following values: where non selected TIMBREAK input is disconnected.
  - LL_SBS_TIMBREAK_ECC
  - LL_SBS_TIMBREAK_PVD
  - LL_SBS_TIMBREAK_SRAM_ECC
  - LL_SBS_TIMBREAK_LOCKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 CLL LL_SBS_SetTIMBreakInputs
- CFGR2 SEL LL_SBS_SetTIMBreakInputs
- CFGR2 PVDL LL_SBS_SetTIMBreakInputs
- CFGR2 ECCL LL_SBS_SetTIMBreakInputs

LL_SBS_GetTIMBreakInputs

Function name

__STATIC_INLINE uint32_t LL_SBS_GetTIMBreakInputs (void)

Function description

Get connections to TIM1/8/15/16/17 Break inputs.

Return values

- **Returned:** value can be a combination of the following values:
  - LL_SBS_TIMBREAK_ECC
  - LL_SBS_TIMBREAK_PVD
  - LL_SBS_TIMBREAK_SRAM_ECC
  - LL_SBS_TIMBREAK_LOCKUP

Reference Manual to LL API cross reference:

- CFGR2 CLL LL_SBS_GetTIMBreakInputs
- CFGR2 SEL LL_SBS_GetTIMBreakInputs
- CFGR2 PVDL LL_SBS_GetTIMBreakInputs
- CFGR2 ECCL LL_SBS_GetTIMBreakInputs

LL_SBS_EPOCHSelection

Function name

__STATIC_INLINE void LL_SBS_EPOCHSelection (uint32_t Epoch_Selection)

Function description

Select EPOCH security sent to SAES IP to encrypt/decrypt keys.

Parameters

- **Epoch_Selection:** Select EPOCH security This parameter can be one of the following values:
  - LL_SBS_EPOCH_SEL_SECURE : EPOCH secure selected.
  - LL_SBS_EPOCH_SEL_NONSECURE : EPOCH non secure selected.
  - LL_SBS_EPOCH_SEL_PUFCHECK : EPOCH all zeros for PUF integrity check.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EPOCHSELCR EPOCH_SEL LL_SBS_EPOCHSelection

LL_SBS_GetEPOCHSelection

Function name

__STATIC_INLINE uint32_t LL_SBS_GetEPOCHSelection (void)

Function description

Get EPOCH security selection.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_EPOCH_SEL_SECURE : EPOCH secure selected.
  - LL_SBS_EPOCH_SEL_NONSECURE : EPOCH non secure selected.
  - LL_SBS_EPOCH_SEL_PUFCHECK : EPOCH all zeros for PUF integrity check.

Reference Manual to LL API cross reference:

- EPOCHSELCR EPOCH_SEL LL_SBS_GetEPOCHSelection

LL_SBS_FLASH_DisableECCNMI

Function name

__STATIC_INLINE void LL_SBS_FLASH_DisableECCNMI (void)

Function description

Disable the NMI in case of double ECC error in FLASH Interface.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ECCNMIR SBS_ECCNMIR_ECCNMI_MASK_EN LL_SBS_FLASH_DisableECCNMI

LL_SBS_FLASH_EnableECCNMI

Function name

__STATIC_INLINE void LL_SBS_FLASH_EnableECCNMI (void)

Function description

Enable the NMI in case of double ECC error in FLASH Interface.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ECCNMIR SBS_ECCNMIR_ECCNMI_MASK_EN LL_SBS_FLASH_EnableECCNMI

LL_SBS_IncrementHDPLValue

Function name

__STATIC_INLINE void LL_SBS_IncrementHDPLValue (void)

Function description

Increment by 1 the HDPL value.

Return values

- **None:**

Reference Manual to LL API cross reference:

- HDPLCR HDPL_INCR LL_SBS_IncrementHDPLValue

LL_SBS_GetHDPLValue

Function name

__STATIC_INLINE uint32_t LL_SBS_GetHDPLValue (void)

Function description

Get the HDPL Value.

Return values

- **Returns:** the HDPL value This return value can be one of the following values:
  - LL_SBS_HDPL_VALUE_0: HDPL0
  - LL_SBS_HDPL_VALUE_1: HDPL1
  - LL_SBS_HDPL_VALUE_2: HDPL2
  - LL_SBS_HDPL_VALUE_3: HDPL3

Reference Manual to LL API cross reference:

- HDPLSR HDPL LL_SBS_GetHDPLValue

LL_SBS_SetOBKHDPL

Function name

__STATIC_INLINE void LL_SBS_SetOBKHDPL (uint32_t OBKHDPL_Value)

Function description

Set the OBK-HDPL Value.

Parameters

- **OBKHDPL_Value:** Value of increment to add to HDPL value to generate the OBK-HDPL. This parameter can be one of the following values:
  - LL_SBS_OBKHDPL_INCR_0 : HDPL
  - LL_SBS_OBKHDPL_INCR_1 : HDPL + 1
  - LL_SBS_OBKHDPL_INCR_2 : HDPL + 2
  - LL_SBS_OBKHDPL_INCR_3 : HDPL + 3

Return values

- **None:**

Reference Manual to LL API cross reference:

- NEXTHDPLCR NEXTHDPL LL_SBS_SetOBKHDPL

LL_SBS_GetOBKHDPL

Function name

__STATIC_INLINE uint32_t LL_SBS_GetOBKHDPL (void)

Function description

Get the OBK-HDPL Value.

Return values

- **Returns:** the incremement to add to HDPL value to generate OBK-HDPL This return value can be one of the following values:
  - LL_SBS_OBKHDPL_INCR_0: HDPL
  - LL_SBS_OBKHDPL_INCR_1: HDPL + 1
  - LL_SBS_OBKHDPL_INCR_2: HDPL + 2
  - LL_SBS_OBKHDPL_INCR_3: HDPL + 3

Reference Manual to LL API cross reference:

- NEXTHDPLCR NEXTHDPL LL_SBS_GetOBKHDPL

LL_SBS_SetAuthDbgHDPL

Function name

__STATIC_INLINE void LL_SBS_SetAuthDbgHDPL (uint32_t Level)

Function description

Set the authenticated debug hide protection level.

Parameters

- **Level:** This parameter can be one of the following values:
  - LL_SBS_HDPL_VALUE_1
  - LL_SBS_HDPL_VALUE_2
  - LL_SBS_HDPL_VALUE_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_AUTH_HDPL LL_SBS_SetAuthDbgHDPL

LL_SBS_GetAuthDbgHDPL

Function name

__STATIC_INLINE uint32_t LL_SBS_GetAuthDbgHDPL (void)

Function description

Get current hide protection level.

Return values

- **Returned:** value is the hide protection level where the authenticated debug is opened:
  - LL_SBS_HDPL_VALUE_1
  - LL_SBS_HDPL_VALUE_2
  - LL_SBS_HDPL_VALUE_3

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_AUTH_HDPL LL_SBS_GetAuthDbgHDPL

LL_SBS_SetAuthDbgSec

Function name

__STATIC_INLINE void LL_SBS_SetAuthDbgSec (uint32_t Security)

Function description

Configure the authenticated debug security access.

Parameters

- **Control:** debug opening secure/non-secure or non-secure only This parameter can be one of the following values:
  - LL_SBS_DEBUG_SEC_NSEC: debug opening for secure and non-secure.
  - LL_SBS_DEBUG_NSEC: debug opening for non-secure only.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_AUTH_SEC LL_SBS_SetAuthDbgSec

LL_SBS_GetAuthDbgSec

Function name

__STATIC_INLINE uint32_t LL_SBS_GetAuthDbgSec (void)

Function description

Get the current value of the hide protection level.

Return values

- **Returned:** value can be one of the following values:
  - SBS_DEBUG_SEC_NSEC: debug opening for secure and non-secure.
  - any other value: debug opening for non-secure only.

Notes

- This function can be only used when device state is Closed.

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_AUTH_SEC LL_SBS_GetAuthDbgSec

LL_SBS_UnlockDebug

Function name

__STATIC_INLINE void LL_SBS_UnlockDebug (void)

Function description

Unlock the debug.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_UNLOCK LL_SBS_UnlockDebug

LL_SBS_IsUnlockedDebug

Function name

__STATIC_INLINE uint32_t LL_SBS_IsUnlockedDebug (void)

Function description

Check if the debug is unlocked.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SBS_DBGCR DBG_UNLOCK LL_SBS_IsUnlockedDebug

LL_SBS_UnlockAccessPort

Function name

__STATIC_INLINE void LL_SBS_UnlockAccessPort (void)

Function description

Unlock the access port.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SBS_DBGCR AP_UNLOCK LL_SBS_UnlockAccessPort

LL_SBS_IsUnlockedAccessPort

Function name

__STATIC_INLINE uint32_t LL_SBS_IsUnlockedAccessPort (void)

Function description

Check if the access port is unlocked.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SBS_DBGCR AP_UNLOCK LL_SBS_IsUnlockedAccessPort

LL_SBS_LockDebugConfig

Function name

__STATIC_INLINE void LL_SBS_LockDebugConfig (void)

Function description

Lock the debug configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SBS_DBGLOCKR DBGCFG_LOCK LL_SBS_LockDebugConfig

LL_SBS_IsLockedDebugConfig

Function name

__STATIC_INLINE uint32_t LL_SBS_IsLockedDebugConfig (void)

Function description

Check if the debug configuration is locked.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SBS_DBGLOCKR DBGCFG_LOCK LL_SBS_IsLockedDebugConfig

LL_SBS_NonSecureLock

Function name

__STATIC_INLINE void LL_SBS_NonSecureLock (uint32_t Item)

Function description

Non-secure Lock of SBS item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be one of the following values :
  - LL_SBS_VTOR_NSEC : VTOR_NS register lock
  - LL_SBS_MPU_NSEC : Non-secure MPU registers lock
  - LL_SBS_LOCK_ALL_NSEC : Non-secure MPU and VTOR_NS lock

Return values

- **None:**

Notes

- Setting lock(s) depends on privilege mode in secure/non-secure code Lock(s) cleared only at system reset

Reference Manual to LL API cross reference:

- CNSLCKR LOCKNSVTOR LL_SBS_NonSecureLock
- CNSLCKR LOCKNSMPU LL_SBS_NonSecureLock

LL_SBS_GetNonSecureLock

Function name

__STATIC_INLINE uint32_t LL_SBS_GetNonSecureLock (void)

Function description

Get the non secure lock state of SBS items.

Return values

- **the:** return value can be one of the following values :
  - LL_SBS_VTOR_NSEC : VTOR_NS register lock
  - LL_SBS_MPU_NSEC : Non-secure MPU registers lock
  - LL_SBS_LOCK_ALL_NSEC : VTOR_NS and Non-secure MPU registers lock

Notes

- Getting lock(s) depends on privilege mode in secure/non-secure code

Reference Manual to LL API cross reference:

- CNSLCKR LOCKNSVTOR LL_SBS_NonSecureLock
- CNSLCKR LOCKNSMPU LL_SBS_NonSecureLock

LL_SBS_SecureLock

Function name

__STATIC_INLINE void LL_SBS_SecureLock (uint32_t Item)

Function description

Secure Lock of System item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be a combination of the following values :
  - LL_SBS_VTOR_AIRCR_SEC : VTOR_S and AIRCR registers lock
  - LL_SBS_MPU_SEC : Secure MPU registers lock
  - LL_SBS_SAU : SAU registers lock
  - LL_SBS_LOCK_ALL_SEC : VTOR_S, AIRCR, Secure MPU and SAU registers lock

Return values

- **None:**

Notes

- Setting lock(s) depends on privilege mode in secure code Lock(s) cleared only at system reset

Reference Manual to LL API cross reference:

- CSLCKR LOCKSVTAIRCR LL_SBS_SecureLock
- CSLCKR LOCKSMPU LL_SBS_SecureLock
- CSLCKR LOCKSAU LL_SBS_SecureLock

LL_SBS_GetSecureLock

Function name

__STATIC_INLINE uint32_t LL_SBS_GetSecureLock (void)

Function description

Get the secure lock state of System items.

Return values

- **the:** return value is a combination of the following values :
  - LL_SBS_VTOR_AIRCR_SEC : VTOR_S and AIRCR registers lock
  - LL_SBS_MPU_SEC : Secure MPU registers lock
  - LL_SBS_SAU : SAU registers lock
  - LL_SBS_LOCK_ALL_SEC : VTOR_S, AIRCR, Secure MPU and SAU registers lock

Notes

- Getting lock(s) depends on privilege mode in secure code

Reference Manual to LL API cross reference:

- CSLCKR LOCKSVTAIRCR LL_SBS_GetSecureLock
- CSLCKR LOCKSMPU LL_SBS_GetSecureLock
- CSLCKR LOCKSAU LL_SBS_GetSecureLock

LL_SBS_ConfigSecure

Function name

__STATIC_INLINE void LL_SBS_ConfigSecure (uint32_t Configuration)

Function description

Configure Secure mode.

Parameters

- **Configuration:** This parameter shall be the full combination of the following values:
  - LL_SBS_CLOCK_SEC or LL_SBS_CLOCK_NSEC
  - LL_SBS_CLASSB_SEC or LL_SBS_CLASSB_NSEC
  - LL_SBS_FPU_SEC or LL_SBS_FPU_NSEC
  - LL_SBS_SMPS_SEC or LL_SBS_SMPS_NSEC

Return values

- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR SBSSEC LL_SBS_ConfigSecure
- SECCFGR CLASSBSEC LL_SBS_ConfigSecure
- SECCFGR FPUSEC LL_SBS_ConfigSecure
- SECCFGR SDCE_SEC_EN LL_SBS_ConfigSecure

LL_SBS_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_SBS_GetConfigSecure (void)

Function description

Get Secure mode configuration.

Return values

- **Returned:** value is the combination of the following values:
  - LL_SBS_CLOCK_SEC or LL_SBS_CLOCK_NSEC
  - LL_SBS_CLASSB_SEC or LL_SBS_CLASSB_NSEC
  - LL_SBS_FPU_SEC or LL_SBS_FPU_NSEC
  - LL_SBS_SMPS_SEC or LL_SBS_SMPS_NSEC

Notes

- Only available when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR SBSSEC LL_SBS_ConfigSecure
- SECCFGR CLASSBSEC LL_SBS_ConfigSecure
- SECCFGR FPUSEC LL_SBS_ConfigSecure
- SECCFGR SDCE_SEC_EN LL_SBS_ConfigSecure

LL_SBS_GetPMOSVddCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SBS_GetPMOSVddCompensationValue (void)

Function description

Get the compensation cell value of the GPIO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCVALR PCV1 LL_SBS_GetPMOSVddCompensationValue

LL_SBS_GetNMOSVddCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SBS_GetNMOSVddCompensationValue (void)

Function description

Get the compensation cell value of the GPIO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the NMOS compensation cell

Reference Manual to LL API cross reference:

- CCVALR NCV1 LL_SBS_GetNMOSVddCompensationValue

LL_SBS_GetPMOSVddIO2CompensationValue

Function name

__STATIC_INLINE uint32_t LL_SBS_GetPMOSVddIO2CompensationValue (void)

Function description

Get the compensation cell value of the GPIO PMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCVALR PCV2 LL_SBS_GetPMOSVddIO2CompensationValue

LL_SBS_GetNMOSVddIO2CompensationValue

Function name

__STATIC_INLINE uint32_t LL_SBS_GetNMOSVddIO2CompensationValue (void)

Function description

Get the compensation cell value of the GPIO NMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the NMOS compensation cell

Reference Manual to LL API cross reference:

- CCVALR NCV2 LL_SBS_GetNMOSVddIO2CompensationValue

LL_SBS_SetPMOSVddCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetPMOSVddCompensationCode (uint32_t PMOSCode)

Function description

Set the compensation cell code of the GPIO PMOS transistor supplied by VDD.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the PMOS compensation cell when the CS1 bit of the SBS_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCSWCR PCC1 LL_SBS_SetPMOSVddCompensationCode

LL_SBS_GetPMOSVddCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetPMOSVddCompensationCode (void)

Function description

Get the compensation cell code of the GPIO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCSWCR PCC1 LL_SBS_GetPMOSVddCompensationCode

LL_SBS_SetPMOSVddIOCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetPMOSVddIOCompensationCode (uint32_t PMOSCode)

Function description

Set the compensation cell code of the GPIO PMOS transistor supplied by VDDIO.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the PMOS compensation cell when the CS2 bit of the SBS_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCSWCR PCC2 LL_SBS_SetPMOSVddIOCompensationCode

LL_SBS_GetPMOSVddIOCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetPMOSVddIOCompensationCode (void)

Function description

Get the compensation cell code of the GPIO PMOS transistor supplied by VDDIO.

Return values

- **Returned:** value is the PMOS compensation

Reference Manual to LL API cross reference:

- CCSWCR PCC2 LL_SBS_GetPMOSVddIOCompensationCode

LL_SBS_SetNMOSVddCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetNMOSVddCompensationCode (uint32_t NMOSCode)

Function description

Set the compensation cell code of the GPIO NMOS transistor supplied by VDD.

Parameters

- **NMOSCode:** NMOS compensation code This code is applied to the NMOS compensation cell when the CS2 bit of the SBS_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCSWCR PCC2 LL_SBS_SetNMOSVddCompensationCode

LL_SBS_GetNMOSVddCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetNMOSVddCompensationCode (void)

Function description

Get the compensation cell code of the GPIO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the Vdd compensation cell code for NMOS transistors

Reference Manual to LL API cross reference:

- CCSWCR NCC1 LL_SBS_GetNMOSVddCompensationCode

LL_SBS_SetNMOSVddIOCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetNMOSVddIOCompensationCode (uint32_t NMOSCode)

Function description

Set the compensation cell code of the GPIO NMOS transistor supplied by VDDIO.

Parameters

- **NMOSCode:** NMOS compensation cell code This code is applied to the NMOS compensation cell when the CS2 bit of the SBS_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCSWCR NCC2 LL_SBS_SetNMOSVddIOCompensationCode

LL_SBS_GetNMOSVddIOCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetNMOSVddIOCompensationCode (void)

Function description

Get the compensation cell code of the GPIO NMOS transistor supplied by VDDIO.

Return values

- **Returned:** value is the NMOS compensation cell code

Reference Manual to LL API cross reference:

- CCSWCR NCC2 LL_SBS_GetNMOSVddIOCompensationCode

LL_SBS_EnableVddCompensationCell

Function name

__STATIC_INLINE void LL_SBS_EnableVddCompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SBS_EnableVddCompensationCell

LL_SBS_EnableVddIOCompensationCell

Function name

__STATIC_INLINE void LL_SBS_EnableVddIOCompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDDIO.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SBS_EnableVddIOCompensationCell

LL_SBS_DisableVddCompensationCell

Function name

__STATIC_INLINE void LL_SBS_DisableVddCompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The Vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SBS_DisableVddCompensationCell

LL_SBS_DisableVddIOCompensationCell

Function name

__STATIC_INLINE void LL_SBS_DisableVddIOCompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDDIO.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SBS_DisableVddIOCompensationCell

LL_SBS_IsEnabled_VddCompensationCell

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabled_VddCompensationCell (void)

Function description

Check if the Compensation Cell of GPIO supplied by VDD is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SBS_IsEnabled_VddCompensationCell

LL_SBS_IsEnabled_VddIOCompensationCell

Function name

__STATIC_INLINE uint32_t LL_SBS_IsEnabled_VddIOCompensationCell (void)

Function description

Check if the Compensation Cell of GPIO supplied by VDDIO is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SBS_IsEnabled_VddIOCompensationCell

LL_SBS_IsActiveFlag_VddCMPCR

Function name

__STATIC_INLINE uint32_t LL_SBS_IsActiveFlag_VddCMPCR (void)

Function description

Get Compensation Cell ready Flag of GPIO supplied by VDD.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR RDY1 LL_SBS_IsActiveFlag_VddCMPCR

LL_SBS_IsActiveFlag_VddIOCMPCR

Function name

__STATIC_INLINE uint32_t LL_SBS_IsActiveFlag_VddIOCMPCR (void)

Function description

Get Compensation Cell ready Flag of GPIO supplied by VDDIO.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR RDY1 LL_SBS_IsActiveFlag_VddIOCMPCR

LL_SBS_SetVddCellCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetVddCellCompensationCode (uint32_t CompCode)

Function description

Set the compensation cell code selection of GPIO supplied by VDD.

Parameters

- **CompCode:** Selects the code to be applied for the Vdd compensation cell This parameter can be one of the following values:
  - LL_SBS_VDD_CELL_CODE : Select Code from the cell (available in the SBS_CCVALR)
  - LL_SBS_VDD_REGISTER_CODE: Select Code from the SBS compensation cell code register (SBS_CCSWCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS1 LL_SBS_SetVddCellCompensationCode

LL_SBS_SetVddIOCellCompensationCode

Function name

__STATIC_INLINE void LL_SBS_SetVddIOCellCompensationCode (uint32_t CompCode)

Function description

Set the compensation cell code selection of GPIO supplied by VDDIO.

Parameters

- **CompCode:** Selects the code to be applied for the VddIO compensation cell This parameter can be one of the following values:
  - LL_SBS_VDDIO_CELL_CODE : Select Code from the cell (available in the SBS_CCVALR)
  - LL_SBS_VDDIO_REGISTER_CODE: Select Code from the SBS compensation cell code register (SBS_CCSWCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS2 LL_SBS_SetVddIOCellCompensationCode

LL_SBS_GetVddCellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetVddCellCompensationCode (void)

Function description

Get the compensation cell code selection of GPIO supplied by VDD.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_VDD_CELL_CODE : Selected Code is from the cell (available in the SBS_CCVALR)
  - LL_SBS_VDD_REGISTER_CODE: Selected Code is from the SBS compensation cell code register (SBS_CCSWCR)

Reference Manual to LL API cross reference:

- CCCSR CS1 LL_SBS_GetVddCellCompensationCode

LL_SBS_GetVddIOCellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SBS_GetVddIOCellCompensationCode (void)

Function description

Get the compensation cell code selection of GPIO supplied by VDDIO.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_VDDIO_CELL_CODE : Selected Code is from the cell (available in the SBS_CCVALR)
  - LL_SBS_VDDIO_REGISTER_CODE: Selected Code is from the SBS compensation cell code register (SBS_CCSWCR)

Reference Manual to LL API cross reference:

- CCCSR CS2 LL_SBS_GetVddIOCellCompensationCode

LL_DBGMCU_GetDeviceID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)

Function description

Return the device identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF (ex: device ID is 0x6415)

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)

Function description

Return the device revision identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- This field indicates the revision of the device.

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_EnableTraceClock

Function name

__STATIC_INLINE void LL_DBGMCU_EnableTraceClock (void)

Function description

Enable the Debug Clock Trace.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_CLKEN LL_DBGMCU_EnableTraceClock

LL_DBGMCU_DisableTraceClock

Function name

__STATIC_INLINE void LL_DBGMCU_DisableTraceClock (void)

Function description

Disable the Debug Clock Trace.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_CLKEN LL_DBGMCU_DisableTraceClock

LL_DBGMCU_IsEnabledTraceClock

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_IsEnabledTraceClock (void)

Function description

Check if clock trace is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DBGMCU_CR_TRACE_CLKEN LL_DBGMCU_IsEnabledTraceClock

LL_DBGMCU_SetTracePinAssignment

Function name

__STATIC_INLINE void LL_DBGMCU_SetTracePinAssignment (uint32_t PinAssignment)

Function description

Set Trace pin assignment control.

Parameters

- **PinAssignment:** This parameter can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)

Function description

Get Trace pin assignment control.

Return values

- **Returned:** value can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I3C1_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_LPTIM2_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I3C1_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_LPTIM2_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZR DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph

LL_DBGMCU_APB3_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB3_GRP1_I2C3_STOP
  - LL_DBGMCU_APB3_GRP1_I2C4_STOP
  - LL_DBGMCU_APB3_GRP1_LPTIM1_STOP
  - LL_DBGMCU_APB3_GRP1_RTC_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZ DBG_TIMx_STOP LL_DBGMCU_APB3_GRP1_FreezePeriph

LL_DBGMCU_APB3_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB3_GRP1_I2C3_STOP
  - LL_DBGMCU_APB3_GRP1_I2C4_STOP
  - LL_DBGMCU_APB3_GRP1_LPTIM1_STOP
  - LL_DBGMCU_APB3_GRP1_RTC_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZR DBG_TIMx_STOP LL_DBGMCU_APB3_GRP1_UnFreezePeriph

LL_VREFBUF_Enable

Function name

__STATIC_INLINE void LL_VREFBUF_Enable (void)

Function description

Enable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Enable

LL_VREFBUF_Disable

Function name

__STATIC_INLINE void LL_VREFBUF_Disable (void)

Function description

Disable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Disable

LL_VREFBUF_EnableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_EnableHIZ (void)

Function description

Enable high impedance (VREF+pin is high impedance)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_EnableHIZ

LL_VREFBUF_DisableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_DisableHIZ (void)

Function description

Disable high impedance (VREF+pin is internally connected to the voltage reference buffer output)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_DisableHIZ

LL_VREFBUF_SetVoltageScaling

Function name

__STATIC_INLINE void LL_VREFBUF_SetVoltageScaling (uint32_t Scale)

Function description

Set the Voltage reference scale.

Parameters

- **Scale:** This parameter can be one of the following values:
  - LL_VREFBUF_VOLTAGE_SCALE0
  - LL_VREFBUF_VOLTAGE_SCALE1
  - LL_VREFBUF_VOLTAGE_SCALE2
  - LL_VREFBUF_VOLTAGE_SCALE3

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_SetVoltageScaling

LL_VREFBUF_GetVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetVoltageScaling (void)

Function description

Get the Voltage reference scale.

Return values

- **Returned:** value can be one of the following values:
  - LL_VREFBUF_VOLTAGE_SCALE0
  - LL_VREFBUF_VOLTAGE_SCALE1
  - LL_VREFBUF_VOLTAGE_SCALE2
  - LL_VREFBUF_VOLTAGE_SCALE3

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_GetVoltageScaling

LL_VREFBUF_IsVREFReady

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_IsVREFReady (void)

Function description

Check if Voltage reference buffer is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRR LL_VREFBUF_IsVREFReady

LL_VREFBUF_GetTrimming

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetTrimming (void)

Function description

Get the trimming code for VREFBUF calibration.

Return values

- **Between:** 0 and 0x3F

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_GetTrimming

LL_VREFBUF_SetTrimming

Function name

__STATIC_INLINE void LL_VREFBUF_SetTrimming (uint32_t Value)

Function description

Set the trimming code for VREFBUF calibration (Tune the internal reference buffer voltage)

Parameters

- **Value:** Between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_SetTrimming

LL_FLASH_SetLatency

Function name

__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)

Function description

Set FLASH Latency.

Parameters

- **Latency:** This parameter can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7
  - LL_FLASH_LATENCY_8
  - LL_FLASH_LATENCY_9
  - LL_FLASH_LATENCY_10
  - LL_FLASH_LATENCY_11
  - LL_FLASH_LATENCY_12
  - LL_FLASH_LATENCY_13
  - LL_FLASH_LATENCY_14
  - LL_FLASH_LATENCY_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)

Function description

Get FLASH Latency.

Return values

- **Returned:** value can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7
  - LL_FLASH_LATENCY_8
  - LL_FLASH_LATENCY_9
  - LL_FLASH_LATENCY_10
  - LL_FLASH_LATENCY_11
  - LL_FLASH_LATENCY_12
  - LL_FLASH_LATENCY_13
  - LL_FLASH_LATENCY_14
  - LL_FLASH_LATENCY_15

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_SBS_ClearEraseEndStatus

Function name

__STATIC_INLINE void LL_SBS_ClearEraseEndStatus (void)

Function description

Clear Status of End of Erase for ICACHE and PKA RAMs.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MESR IPMEE LL_SBS_ClearEraseEndStatus

LL_SBS_GetEraseEndStatus

Function name

__STATIC_INLINE uint32_t LL_SBS_GetEraseEndStatus (void)

Function description

Get Status of End of Erase for ICACHE and PKA RAMs.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_MEMORIES_ERASE_IPMEE_ON_GOING : Erase of ICACHE and PKA RAMs on going or flag cleared by SW
  - LL_SBS_MEMORIES_ERASE_IPMEE_ENDED: Erase of ICACHE and PKA RAMs ended

Reference Manual to LL API cross reference:

- MESR IPMEE LL_SBS_GetEraseEndStatus

LL_SBS_ClearEraseAfterResetStatus

Function name

__STATIC_INLINE void LL_SBS_ClearEraseAfterResetStatus (void)

Function description

Clear Status of End of Erase after Power-on Reset for SRAM2, BKPRAM, ICACHE, DCACHE and PKA RAMs.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MESR MCLR LL_SBS_ClearEraseAfterResetStatus

LL_SBS_GetEraseAfterResetStatus

Function name

__STATIC_INLINE uint32_t LL_SBS_GetEraseAfterResetStatus (void)

Function description

Get Status of End of Erase after Power-on Reset for SRAM2, BKPRAM, ICACHE, DCACHE and PKA RAMs.

Return values

- **Returned:** value can be one of the following values:
  - LL_SBS_MEMORIES_ERASE_MCLR_ON_GOING : Erase of memories on going or flag cleared by SW
  - LL_SBS_MEMORIES_ERASE_MCLR_ENDED: Erase of memories ended

Reference Manual to LL API cross reference:

- MESR MCLR LL_SBS_GetEraseAfterResetStatus

113.2 SYSTEM Firmware driver defines

The following section lists the various define and macros of the module.

113.2.1 SYSTEM

SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

The counter clock of TIM2 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM3_STOP

The counter clock of TIM3 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM4_STOP

The counter clock of TIM4 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM5_STOP

The counter clock of TIM5 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM6_STOP

The counter clock of TIM6 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM7_STOP

The counter clock of TIM7 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM12_STOP

The counter clock of TIM12 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM13_STOP

The counter clock of TIM13 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM14_STOP

The counter clock of TIM14 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_WWDG_STOP

The window watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_IWDG_STOP

The independent watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_I2C1_STOP

The I2C1 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I2C2_STOP

The I2C2 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I3C1_STOP

The I3C1 SMBus timeout is frozen

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_LPTIM2_STOP

The counter clock of LPTIM2 is stopped when the core is halted

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP

The counter clock of TIM1 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM8_STOP

The counter clock of TIM8 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM15_STOP

The counter clock of TIM15 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM16_STOP

The counter clock of TIM16 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM17_STOP

The counter clock of TIM17 is stopped when the core is halted

DBGMCU APB3 GRP1 STOP IP

LL_DBGMCU_APB3_GRP1_I2C3_STOP

The counter clock of I2C3 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_I2C4_STOP

The counter clock of I2C4 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM1_STOP

The counter clock of LPTIM1 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM3_STOP

The counter clock of LPTIM3 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM4_STOP

The counter clock of LPTIM4 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM5_STOP

The counter clock of LPTIM5 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM6_STOP

The counter clock of LPTIM6 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_RTC_STOP

The counter clock of RTC is stopped when the core is halted

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE

TRACE pins not assigned (default state)

LL_DBGMCU_TRACE_ASYNCH

TRACE pin assignment for Asynchronous Mode

LL_DBGMCU_TRACE_SYNCH_SIZE1

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1

LL_DBGMCU_TRACE_SYNCH_SIZE2

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2

LL_DBGMCU_TRACE_SYNCH_SIZE4

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

EXTI EXTI LINE

LL_EXTI_EXTI_LINE0

EXTI_POSITION_0 | EXTICR[0]

LL_EXTI_EXTI_LINE1

EXTI_POSITION_8 | EXTICR[0]

LL_EXTI_EXTI_LINE2

EXTI_POSITION_16 | EXTICR[0]

LL_EXTI_EXTI_LINE3

EXTI_POSITION_24 | EXTICR[0]

LL_EXTI_EXTI_LINE4

EXTI_POSITION_0 | EXTICR[1]

LL_EXTI_EXTI_LINE5

EXTI_POSITION_8 | EXTICR[1]

LL_EXTI_EXTI_LINE6

EXTI_POSITION_16 | EXTICR[1]

LL_EXTI_EXTI_LINE7

EXTI_POSITION_24 | EXTICR[1]

LL_EXTI_EXTI_LINE8

EXTI_POSITION_0 | EXTICR[2]

LL_EXTI_EXTI_LINE9

EXTI_POSITION_8 | EXTICR[2]

LL_EXTI_EXTI_LINE10

EXTI_POSITION_16 | EXTICR[2]

LL_EXTI_EXTI_LINE11

EXTI_POSITION_24 | EXTICR[2]

LL_EXTI_EXTI_LINE12

EXTI_POSITION_0 | EXTICR[3]

LL_EXTI_EXTI_LINE13

EXTI_POSITION_8 | EXTICR[3]

LL_EXTI_EXTI_LINE14

EXTI_POSITION_16 | EXTICR[3]

LL_EXTI_EXTI_LINE15

EXTI_POSITION_24 | EXTICR[3]

EXTI EXTI PORT

LL_EXTI_EXTI_PORTA

EXTI PORT A

LL_EXTI_EXTI_PORTB

EXTI PORT B

LL_EXTI_EXTI_PORTC

EXTI PORT C

LL_EXTI_EXTI_PORTD

EXTI PORT D

LL_EXTI_EXTI_PORTE

EXTI PORT E

LL_EXTI_EXTI_PORTF

EXTI PORT F

LL_EXTI_EXTI_PORTG

EXTI PORT G

LL_EXTI_EXTI_PORTH

EXTI PORT H

LL_EXTI_EXTI_PORTI

EXTI PORT I

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH zero wait state

LL_FLASH_LATENCY_1

FLASH one wait state

LL_FLASH_LATENCY_2

FLASH two wait states

LL_FLASH_LATENCY_3

FLASH three wait states

LL_FLASH_LATENCY_4

FLASH four wait states

LL_FLASH_LATENCY_5

FLASH five wait states

LL_FLASH_LATENCY_6

FLASH six wait state

LL_FLASH_LATENCY_7

FLASH Seven wait states

LL_FLASH_LATENCY_8

FLASH Eight wait states

LL_FLASH_LATENCY_9

FLASH nine wait states

LL_FLASH_LATENCY_10

FLASH ten wait states

LL_FLASH_LATENCY_11

FLASH eleven wait states

LL_FLASH_LATENCY_12

FLASH twelve wait states

LL_FLASH_LATENCY_13

FLASH thirteen wait states

LL_FLASH_LATENCY_14

FLASH fourteen wait states

LL_FLASH_LATENCY_15

FLASH fifteen wait states

SBS Vdd compensation cell Code selection

LL_SBS_VDD_CELL_CODE

VDD I/Os code from the cell (available in the SBS_CCVALR)

LL_SBS_VDD_REGISTER_CODE

VDD I/Os code from the SBS compensation cell code register (SBS_CCSWCR)

SBS VddIO compensation cell Code selection

LL_SBS_VDDIO_CELL_CODE

VDDIO I/Os code from the cell (available in the SBS_CCVALR)

LL_SBS_VDDIO_REGISTER_CODE

VDDIO I/Os code from the SBS compensation cell code register (SBS_CCSWCR)

SBS FASTMODEPLUS

LL_SBS_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

LL_SBS_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

LL_SBS_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

LL_SBS_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

Secure attributes

LL_SBS_CLOCK_SEC

SBS clock configuration secure-only access

LL_SBS_CLOCK_NSEC

SBS clock configuration secure/non-secure access

LL_SBS_CLASSB_SEC

Class B configuration secure-only access

LL_SBS_CLASSB_NSEC

Class B configuration secure/non-secure access

LL_SBS_FPU_SEC

FPU configuration secure-only access

LL_SBS_FPU_NSEC

FPU configuration secure/non-secure access

LL_SBS_SMPS_SEC

SMPS configuration secure-only access

LL_SBS_SMPS_NSEC

SMPS configuration secure/non-secure access

SBS TIMER BREAK

LL_SBS_TIMBREAK_ECC

Enables and locks the Flash ECC double error signal with Break Input of TIM1/8/15/16/17

LL_SBS_TIMBREAK_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface

LL_SBS_TIMBREAK_SRAM_ECC

Enables and locks the SRAM ECC double error signal with Break Input of TIM1/8/15/16/17

LL_SBS_TIMBREAK_LOCKUP

Enables and locks the LOCKUP (Hardfault) output of Cortex-M33 with Break Input of TIM1/15/16/17

EPOCH Selection

LL_SBS_EPOCH_SEL_SECURE

EPOCH secure selected

LL_SBS_EPOCH_SEL_NONSECURE

EPOCH non secure selected

LL_SBS_EPOCH_SEL_PUFCHECK

EPOCH all zeros for PUF integrity check

ETHENET CONFIG

LL_SBS_ETH_MII

Select the Media Independent Interface (MII) or GMII

LL_SBS_ETH_RMII

Select the Reduced Media Independent Interface (RMII)

HDPL Value

LL_SBS_HDPL_VALUE_0

Hide protection level 0

LL_SBS_HDPL_VALUE_1

Hide protection level 1

LL_SBS_HDPL_VALUE_2

Hide protection level 2

LL_SBS_HDPL_VALUE_3

Hide protection level 3

Next HDPL Selection

LL_SBS_OBKHDPL_INCR_0

Index to add to the current HDPL to point (through OBK-HDPL) to the next secure storage areas

LL_SBS_OBKHDPL_INCR_1

Index to add to the current HDPL to point (through OBK-HDPL) to the next secure storage areas

LL_SBS_OBKHDPL_INCR_2

Index to add to the current HDPL to point (through OBK-HDPL) to the next secure storage areas

LL_SBS_OBKHDPL_INCR_3

Index to add to the current HDPL to point (through OBK-HDPL) to the next secure storage areas

Lock items

LL_SBS_MPU_NSEC

Non-secure MPU lock (privileged secure or non-secure only)

LL_SBS_VTOR_NSEC

Non-secure VTOR lock (privileged secure or non-secure only)

LL_SBS_LOCK_ALL_NSEC

lock all Non-secure (privileged secure or non-secure only)

SBS Lock items

LL_SBS_SAU

SAU lock (privileged secure code only)

LL_SBS_MPU_SEC

Secure MPU lock (privileged secure code only)

LL_SBS_VTOR_AIRCR_SEC

VTOR_S and AIRCR lock (privileged secure code only)

LL_SBS_LOCK_ALL_SEC

lock all secure (privileged secure only)

VREFBUF VOLTAGE

LL_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREF_OUT1)

LL_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREF_OUT2)

LL_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREF_OUT3)

LL_VREFBUF_VOLTAGE_SCALE3

Voltage reference scale 3 (VREF_OUT4)

114 LL TIM Generic Driver

114.1 TIM Firmware driver registers structures

114.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***
  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint32_t LL_TIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

114.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***
  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***
  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***
  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***
  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

114.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

114.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
  Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
  Specifies the TI1 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
  Specifies the active edge of TI2 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
  Specifies the TI2 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
  Specifies the TI2 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

114.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***
  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

114.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32h5xx_ll_tim.h

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***
  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of ***TIM_LL_EC_OSSI***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of ***TIM_LL_EC_LOCKLEVEL***
  Note:
  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.
- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***
  Specifies the TIM Break Filter. This parameter can be a value of ***TIM_LL_EC_BREAK_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input. This parameter can be a value of ***TIM_LL_EC_BREAK_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***
  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK2_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***
  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK2_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***
  Specifies the TIM Break2 Filter. This parameter can be a value of ***TIM_LL_EC_BREAK2_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input. This parameter can be a value of ***TIM_LL_EC_BREAK2_AFMODE***This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK2()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_LL_EC_AUTOMATICOUTPUT_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

114.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

114.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name

__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)

Function description

Enable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name

__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)

Function description

Disable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (const TIM_TypeDef * TIMx)

Function description

Indicates whether the timer counter is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Enable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Disable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (const TIM_TypeDef * TIMx)

Function description

Indicates whether update event generation is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name

__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)

Function description

Set update event source.

Parameters

- **TIMx:** Timer instance
- **UpdateSource:** This parameter can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Return values

- **None:**

Notes

- Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (const TIM_TypeDef * TIMx)

Function description

Get actual event update source.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name

__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)

Function description

Set one pulse mode (one shot v.s.

Parameters

- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (const TIM_TypeDef * TIMx)

Function description

Get actual one pulse mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)

Function description

Set the timer counter counting mode.

Parameters

- **TIMx:** Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_SetCounterMode
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (const TIM_TypeDef * TIMx)

Function description

Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name

__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)

Function description

Enable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name

__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)

Function description

Disable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether auto-reload (ARR) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name

__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)

Function description

Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name

__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (const TIM_TypeDef * TIMx)

Function description

Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name

__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)

Function description

Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Return values

- **None:**

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounter (const TIM_TypeDef * TIMx)

Function description

Get the counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetDirection (const TIM_TypeDef * TIMx)

Function description

Get the current direction of the counter.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERDIRECTION_UP
  - LL_TIM_COUNTERDIRECTION_DOWN

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)

Function description

Set the prescaler value.

Parameters

- **TIMx:** Timer instance
- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name

__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)

Function description

Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter In case dithering is activated,macro __LL_TIM_CALC_ARR_DITHER can be used instead, to calculate the AutoReload parameter.

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (const TIM_TypeDef * TIMx)

Function description

Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the returned value interpretation

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name

__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)

Function description

Set the repetition counter value.

Parameters

- **TIMx:** Timer instance
- **RepetitionCounter:** between Min_Data=0 and Max_Data=255 or 65535 for advanced timer.

Return values

- **None:**

Notes

- For advanced timer instances RepetitionCounter can be up to 65535.
- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (const TIM_TypeDef * TIMx)

Function description

Get the repetition counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Repetition:** counter value

Notes

- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)

Function description

Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)

Function description

Disable update interrupt flag (UIF) remapping.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_IsActiveUIFCPY

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveUIFCPY (const uint32_t Counter)

Function description

Indicate whether update interrupt flag (UIF) copy is set.

Parameters

- **Counter:** Counter value

Return values

- **State:** of bit (1 or 0).

LL_TIM_EnableDithering

Function name

__STATIC_INLINE void LL_TIM_EnableDithering (TIM_TypeDef * TIMx)

Function description

Enable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_EnableDithering

LL_TIM_DisableDithering

Function name

__STATIC_INLINE void LL_TIM_DisableDithering (TIM_TypeDef * TIMx)

Function description

Disable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_DisableDithering

LL_TIM_IsEnabledDithering

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDithering (const TIM_TypeDef * TIMx)

Function description

Indicates whether dithering is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_IsEnabledDithering

LL_TIM_CC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)

Function description

Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.
- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)

Function description

Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_SetUpdate

Function name

__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)

Function description

Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters

- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name

__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)

Function description

Set the trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (const TIM_TypeDef * TIMx)

Function description

Get actual trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name

__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description

Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters

- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  - LL_TIM_LOCKLEVEL_OFF
  - LL_TIM_LOCKLEVEL_1
  - LL_TIM_LOCKLEVEL_2
  - LL_TIM_LOCKLEVEL_3

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:

- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Enable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel
- CCER CC4NE LL_TIM_CC_EnableChannel
- CCER CC5E LL_TIM_CC_EnableChannel
- CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Disable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC4NE LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (const TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Indicate whether channel(s) is(are) enabled.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC4NE LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name

__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCMR3 CC5S LL_TIM_OC_ConfigOutput
- CCMR3 CC6S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CCER CC5P LL_TIM_OC_ConfigOutput
- CCER CC6P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput
- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput
- CR2 OIS5 LL_TIM_OC_ConfigOutput
- CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name

__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description

Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the output compare mode of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode
- CCMR3 OC5M LL_TIM_OC_GetMode
- CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity
- CCER CC4NP LL_TIM_OC_SetPolarity
- CCER CC5P LL_TIM_OC_SetPolarity
- CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity
- CCER CC4NP LL_TIM_OC_GetPolarity
- CCER CC5P LL_TIM_OC_GetPolarity
- CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name

__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)

Function description

Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **IdleState:** This parameter can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS4N LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState
- CR2 OIS4N LL_TIM_OC_GetIdleState
- CR2 OIS5 LL_TIM_OC_GetIdleState
- CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name

__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast
- CCMR3 OC5FE LL_TIM_OC_EnableFast
- CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name

__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_DisableFast
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast
- CCMR3 OC5FE LL_TIM_OC_DisableFast
- CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether fast mode is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC5FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload
- CCMR3 OC5PE LL_TIM_OC_EnablePreload
- CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload
- CCMR3 OC5PE LL_TIM_OC_DisablePreload
- CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name

__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_EnableClear
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear
- CCMR3 OC5CE LL_TIM_OC_EnableClear
- CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name

__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear
- CCMR3 OC5CE LL_TIM_OC_DisableClear
- CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Notes

- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC5CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name

__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 2 (TIMx_CCR2).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 3 (TIMx_CCR3).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 4 (TIMx_CCR4).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 5 (TIMx_CCR5).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 6 (TIMx_CCR6).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR1) set for output channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR2) set for output channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR5) set for output channel 5.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR6) set for output channel 6.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name

__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)

Function description

Select on which reference signal the OC5REF is combined to.

Parameters

- **TIMx:** Timer instance
- **GroupCH5:** This parameter can be a combination of the following values:
  - LL_TIM_GROUPCH5_NONE
  - LL_TIM_GROUPCH5_OC1REFC
  - LL_TIM_GROUPCH5_OC2REFC
  - LL_TIM_GROUPCH5_OC3REFC

Return values

- **None:**

Notes

- Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.

Reference Manual to LL API cross reference:

- CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_OC_SetPulseWidthPrescaler

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidthPrescaler (TIM_TypeDef * TIMx, uint32_t PulseWidthPrescaler)

Function description

Set the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance
- **PulseWidthPrescaler:** This parameter can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_SetPulseWidthPrescaler

LL_TIM_OC_GetPulseWidthPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidthPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_PWPRSC_X1
  - LL_TIM_PWPRSC_X2
  - LL_TIM_PWPRSC_X4
  - LL_TIM_PWPRSC_X8
  - LL_TIM_PWPRSC_X16
  - LL_TIM_PWPRSC_X32
  - LL_TIM_PWPRSC_X64
  - LL_TIM_PWPRSC_X128

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_GetPulseWidthPrescaler

LL_TIM_OC_SetPulseWidth

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidth (TIM_TypeDef * TIMx, uint32_t PulseWidth)

Function description

Set the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance
- **PulseWidth:** This parameter can be between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_SetPulseWidth

LL_TIM_OC_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidth (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_GetPulseWidth

LL_TIM_IC_Config

Function name

__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name

__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description

Set the active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description

Set the prescaler of input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current prescaler value acting on an input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)

Function description

Set the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)

Function description

Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
- CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
- CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)

Function description

Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)

Function description

Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (const TIM_TypeDef * TIMx)

Function description

Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name

__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)

Function description

Enable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name

__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)

Function description

Disable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (const TIM_TypeDef * TIMx)

Function description

Indicate whether external clock mode 2 is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name

__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)

Function description

Set the clock source of the counter clock.

Parameters

- **TIMx:** Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_TIM_CLOCKSOURCE_INTERNAL
  - LL_TIM_CLOCKSOURCE_EXT_MODE1
  - LL_TIM_CLOCKSOURCE_EXT_MODE2

Return values

- **None:**

Notes

- when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetClockSource
- SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)

Function description

Set the encoder interface mode.

Parameters

- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12
  - LL_TIM_ENCODERMODE_X1_TI1
  - LL_TIM_ENCODERMODE_X1_TI2

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description

Set the trigger output (TRGO) used for timer synchronization .

Parameters

- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF
  - LL_TIM_TRGO_ENCODERCLK

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:

- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)

Function description

Set the trigger output 2 (TRGO2) used for ADC synchronization .

Parameters

- **TIMx:** Timer Instance
- **ADCSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO2_RESET
  - LL_TIM_TRGO2_ENABLE
  - LL_TIM_TRGO2_UPDATE
  - LL_TIM_TRGO2_CC1F
  - LL_TIM_TRGO2_OC1
  - LL_TIM_TRGO2_OC2
  - LL_TIM_TRGO2_OC3
  - LL_TIM_TRGO2_OC4
  - LL_TIM_TRGO2_OC5
  - LL_TIM_TRGO2_OC6
  - LL_TIM_TRGO2_OC4_RISINGFALLING
  - LL_TIM_TRGO2_OC6_RISINGFALLING
  - LL_TIM_TRGO2_OC4_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
  - LL_TIM_TRGO2_OC5_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values

- **None:**

Notes

- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name

__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description

Set the synchronization mode of a slave timer.

Parameters

- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  - LL_TIM_SLAVEMODE_DISABLED
  - LL_TIM_SLAVEMODE_RESET
  - LL_TIM_SLAVEMODE_GATED
  - LL_TIM_SLAVEMODE_TRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description

Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_ITR4
  - LL_TIM_TS_ITR5
  - LL_TIM_TS_ITR6
  - LL_TIM_TS_ITR7
  - LL_TIM_TS_ITR8
  - LL_TIM_TS_ITR9
  - LL_TIM_TS_ITR10
  - LL_TIM_TS_ITR11
  - LL_TIM_TS_ITR12
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (const TIM_TypeDef * TIMx)

Function description

Indicates whether the Master/Slave mode is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name

__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)

Function description

Configure the external trigger (ETR) input.

Parameters

- **TIMx:** Timer instance
- **ETRPolarity:** This parameter can be one of the following values:
  - LL_TIM_ETR_POLARITY_NONINVERTED
  - LL_TIM_ETR_POLARITY_INVERTED
- **ETRPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ETR_PRESCALER_DIV1
  - LL_TIM_ETR_PRESCALER_DIV2
  - LL_TIM_ETR_PRESCALER_DIV4
  - LL_TIM_ETR_PRESCALER_DIV8
- **ETRFilter:** This parameter can be one of the following values:
  - LL_TIM_ETR_FILTER_FDIV1
  - LL_TIM_ETR_FILTER_FDIV1_N2
  - LL_TIM_ETR_FILTER_FDIV1_N4
  - LL_TIM_ETR_FILTER_FDIV1_N8
  - LL_TIM_ETR_FILTER_FDIV2_N6
  - LL_TIM_ETR_FILTER_FDIV2_N8
  - LL_TIM_ETR_FILTER_FDIV4_N6
  - LL_TIM_ETR_FILTER_FDIV4_N8
  - LL_TIM_ETR_FILTER_FDIV8_N6
  - LL_TIM_ETR_FILTER_FDIV8_N8
  - LL_TIM_ETR_FILTER_FDIV16_N5
  - LL_TIM_ETR_FILTER_FDIV16_N6
  - LL_TIM_ETR_FILTER_FDIV16_N8
  - LL_TIM_ETR_FILTER_FDIV32_N5
  - LL_TIM_ETR_FILTER_FDIV32_N6
  - LL_TIM_ETR_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:

- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name

__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)

Function description

Select the external trigger (ETR) input source.

Parameters

- **TIMx:** Timer instance
- **ETRSource:** This parameter can be one of the following values:

Return values

- **None:**

Notes

- Macro IS_TIM_ETRSEL_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.

Reference Manual to LL API cross reference:

- AF1 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_EnableSMSPreload (TIM_TypeDef * TIMx)

Function description

Enable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_EnableSMSPreload

LL_TIM_DisableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_DisableSMSPreload (TIM_TypeDef * TIMx)

Function description

Disable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_DisableSMSPreload

LL_TIM_IsEnabledSMSPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledSMSPreload (const TIM_TypeDef * TIMx)

Function description

Indicate whether SMS preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_IsEnabledSMSPreload

LL_TIM_SetSMSPreloadSource

Function name

__STATIC_INLINE void LL_TIM_SetSMSPreloadSource (TIM_TypeDef * TIMx, uint32_t PreloadSource)

Function description

Set the preload source of SMS.

Parameters

- **TIMx:** Timer instance
- **PreloadSource:** This parameter can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_SetSMSPreloadSource
-

LL_TIM_GetSMSPreloadSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetSMSPreloadSource (const TIM_TypeDef * TIMx)

Function description

Get the preload source of SMS.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_SMSPS_TIMUPDATE
  - LL_TIM_SMSPS_INDEX

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_GetSMSPreloadSource
-

LL_TIM_EnableBRK

Function name

__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)

Function description

Enable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name

__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)

Function description

Disable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter, uint32_t BreakAFMode)

Function description

Configure the break input.

Parameters

- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK_POLARITY_LOW
  - LL_TIM_BREAK_POLARITY_HIGH
- **BreakFilter:** This parameter can be one of the following values:
  - LL_TIM_BREAK_FILTER_FDIV1
  - LL_TIM_BREAK_FILTER_FDIV1_N2
  - LL_TIM_BREAK_FILTER_FDIV1_N4
  - LL_TIM_BREAK_FILTER_FDIV1_N8
  - LL_TIM_BREAK_FILTER_FDIV2_N6
  - LL_TIM_BREAK_FILTER_FDIV2_N8
  - LL_TIM_BREAK_FILTER_FDIV4_N6
  - LL_TIM_BREAK_FILTER_FDIV4_N8
  - LL_TIM_BREAK_FILTER_FDIV8_N6
  - LL_TIM_BREAK_FILTER_FDIV8_N8
  - LL_TIM_BREAK_FILTER_FDIV16_N5
  - LL_TIM_BREAK_FILTER_FDIV16_N6
  - LL_TIM_BREAK_FILTER_FDIV16_N8
  - LL_TIM_BREAK_FILTER_FDIV32_N5
  - LL_TIM_BREAK_FILTER_FDIV32_N6
  - LL_TIM_BREAK_FILTER_FDIV32_N8
- **BreakAFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK_AFMODE_INPUT
  - LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BKBID bit set), the Break input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, BreakAFMode must be set to LL_TIM_BREAK_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BKP LL_TIM_ConfigBRK
- BDTR BKF LL_TIM_ConfigBRK
- BDTR BKBID LL_TIM_ConfigBRK

LL_TIM_DisarmBRK

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK (TIM_TypeDef * TIMx)

Function description

Disarm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_DisarmBRK

LL_TIM_ReArmBRK

Function name

__STATIC_INLINE void LL_TIM_ReArmBRK (TIM_TypeDef * TIMx)

Function description

Re-arm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The Break input is automatically armed as soon as MOE bit is set.

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_ReArmBRK

LL_TIM_EnableBRK2

Function name

__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)

Function description

Enable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name

__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)

Function description

Disable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter, uint32_t Break2AFMode)

Function description

Configure the break 2 input.

Parameters

- **TIMx:** Timer instance
- **Break2Polarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_POLARITY_LOW
  - LL_TIM_BREAK2_POLARITY_HIGH
- **Break2Filter:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_FILTER_FDIV1
  - LL_TIM_BREAK2_FILTER_FDIV1_N2
  - LL_TIM_BREAK2_FILTER_FDIV1_N4
  - LL_TIM_BREAK2_FILTER_FDIV1_N8
  - LL_TIM_BREAK2_FILTER_FDIV2_N6
  - LL_TIM_BREAK2_FILTER_FDIV2_N8
  - LL_TIM_BREAK2_FILTER_FDIV4_N6
  - LL_TIM_BREAK2_FILTER_FDIV4_N8
  - LL_TIM_BREAK2_FILTER_FDIV8_N6
  - LL_TIM_BREAK2_FILTER_FDIV8_N8
  - LL_TIM_BREAK2_FILTER_FDIV16_N5
  - LL_TIM_BREAK2_FILTER_FDIV16_N6
  - LL_TIM_BREAK2_FILTER_FDIV16_N8
  - LL_TIM_BREAK2_FILTER_FDIV32_N5
  - LL_TIM_BREAK2_FILTER_FDIV32_N6
  - LL_TIM_BREAK2_FILTER_FDIV32_N8
- **Break2AFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_AFMODE_INPUT
  - LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BK2BID bit set), the Break 2 input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break 2 input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, Break2AFMode must be set to LL_TIM_BREAK2_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BK2P LL_TIM_ConfigBRK2
- BDTR BK2F LL_TIM_ConfigBRK2
- BDTR BK2BID LL_TIM_ConfigBRK2

LL_TIM_DisarmBRK2

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK2 (TIM_TypeDef * TIMx)

Function description

Disarm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break 2 input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_DisarmBRK2

LL_TIM_ReArmBRK2

Function name

__STATIC_INLINE void LL_TIM_ReArmBRK2 (TIM_TypeDef * TIMx)

Function description

Re-arm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The Break 2 input is automatically armed as soon as MOE bit is set.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_ReArmBRK2

LL_TIM_SetOffStates

Function name

__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)

Function description

Select the outputs off state (enabled v.s.

Parameters

- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  - LL_TIM_OSSI_DISABLE
  - LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  - LL_TIM_OSSR_DISABLE
  - LL_TIM_OSSR_ENABLE

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Disable automatic output (MOE can be set only by software).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (const TIM_TypeDef * TIMx)

Function description

Indicate whether automatic output is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)

Function description

Enable the outputs (set the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)

Function description

Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (const TIM_TypeDef * TIMx)

Function description

Indicates whether outputs are enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Enable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1 (*)

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_EnableBreakInputSource
- AF1 BKCMP1E LL_TIM_EnableBreakInputSource
- AF2 BK2INE LL_TIM_EnableBreakInputSource
- AF2 BK2CMP1E LL_TIM_EnableBreakInputSource
-

LL_TIM_DisableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Disable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1 (*)

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_DisableBreakInputSource
- AF1 BKCMP1E LL_TIM_DisableBreakInputSource
- AF2 BK2INE LL_TIM_DisableBreakInputSource
- AF2 BK2CMP1E LL_TIM_DisableBreakInputSource
-

LL_TIM_SetBreakInputSourcePolarity

Function name

__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)

Function description

Set the polarity of the break signal for the timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1 (*)
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_BKIN_POLARITY_LOW
  - LL_TIM_BKIN_POLARITY_HIGH

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINP LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP1P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP1P LL_TIM_SetBreakInputSourcePolarity
-

LL_TIM_EnableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_EnableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Enable asymmetrical deadtime.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_EnableAsymmetricalDeadTime

LL_TIM_DisableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_DisableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Disable asymmetrical dead-time.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_DisableAsymmetricalDeadTime

LL_TIM_IsEnabledAsymmetricalDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAsymmetricalDeadTime (const TIM_TypeDef * TIMx)

Function description

Indicates whether asymmetrical deadtime is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_IsEnabledAsymmetricalDeadTime

LL_TIM_SetFallingDeadTime

Function name

__STATIC_INLINE void LL_TIM_SetFallingDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_SetFallingDeadTime

LL_TIM_GetFallingDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_GetFallingDeadTime (const TIM_TypeDef * TIMx)

Function description

Get the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_GetFallingDeadTime

LL_TIM_EnableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_EnableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Enable deadtime preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_EnableDeadTimePreload

LL_TIM_DisableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_DisableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Disable dead-time preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_DisableDeadTimePreload

LL_TIM_IsEnabledDeadTimePreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDeadTimePreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether deadtime preload is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_IsEnabledDeadTimePreload

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength, uint32_t DMABurstSource)

Function description

Configures the timer DMA burst feature.

Parameters

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_DTR2
  - LL_TIM_DMABURST_BASEADDR_ECR
  - LL_TIM_DMABURST_BASEADDR_TISEL
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2

- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS
  - LL_TIM_DMABURST_LENGTH_15TRANSFERS
  - LL_TIM_DMABURST_LENGTH_16TRANSFERS
  - LL_TIM_DMABURST_LENGTH_17TRANSFERS
  - LL_TIM_DMABURST_LENGTH_18TRANSFERS
  - LL_TIM_DMABURST_LENGTH_19TRANSFERS
  - LL_TIM_DMABURST_LENGTH_20TRANSFERS
  - LL_TIM_DMABURST_LENGTH_21TRANSFERS
  - LL_TIM_DMABURST_LENGTH_22TRANSFERS
  - LL_TIM_DMABURST_LENGTH_23TRANSFERS
  - LL_TIM_DMABURST_LENGTH_24TRANSFERS
  - LL_TIM_DMABURST_LENGTH_25TRANSFERS
  - LL_TIM_DMABURST_LENGTH_26TRANSFERS
- **DMABurstSource:** This parameter can be one of the following values:
  - LL_TIM_DMA_UPDATE
  - LL_TIM_DMA_CC1
  - LL_TIM_DMA_CC2
  - LL_TIM_DMA_CC3
  - LL_TIM_DMA_CC4
  - LL_TIM_DMA_COM
  - LL_TIM_DMA_TRIGGER

Return values

- **None:**

Notes

- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:

- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_EnableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_EnableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Enable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_EnableEncoderIndex

LL_TIM_DisableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_DisableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Disable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_DisableEncoderIndex

LL_TIM_IsEnabledEncoderIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledEncoderIndex (const TIM_TypeDef * TIMx)

Function description

Indicate whether encoder index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_IsEnabledEncoderIndex

LL_TIM_SetIndexDirection

Function name

__STATIC_INLINE void LL_TIM_SetIndexDirection (TIM_TypeDef * TIMx, uint32_t IndexDirection)

Function description

Set index direction.

Parameters

- **TIMx:** Timer instance
- **IndexDirection:** This parameter can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_SetIndexDirection

LL_TIM_GetIndexDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexDirection (const TIM_TypeDef * TIMx)

Function description

Get actual index direction.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_GetIndexDirection

LL_TIM_SetIndexblanking

Function name

__STATIC_INLINE void LL_TIM_SetIndexblanking (TIM_TypeDef * TIMx, uint32_t Indexblanking)

Function description

Set index blanking.

Parameters

- **TIMx:** Timer instance
- **Indexblanking:** This parameter can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_SetIndexblanking

LL_TIM_GetIndexblanking

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexblanking (const TIM_TypeDef * TIMx)

Function description

Get actual index blanking.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_GetIndexblanking

LL_TIM_EnableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_EnableFirstIndex (TIM_TypeDef * TIMx)

Function description

Enable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_EnableFirstIndex

LL_TIM_DisableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_DisableFirstIndex (TIM_TypeDef * TIMx)

Function description

Disable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_DisableFirstIndex

LL_TIM_IsEnabledFirstIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledFirstIndex (const TIM_TypeDef * TIMx)

Function description

Indicates whether first index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_IsEnabledFirstIndex

LL_TIM_SetIndexPositionning

Function name

__STATIC_INLINE void LL_TIM_SetIndexPositionning (TIM_TypeDef * TIMx, uint32_t IndexPositionning)

Function description

Set index positioning.

Parameters

- **TIMx:** Timer instance
- **IndexPositionning:** This parameter can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_SetIndexPositionning

LL_TIM_GetIndexPositionning

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexPositionning (const TIM_TypeDef * TIMx)

Function description

Get actual index positioning.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_POSITION_DOWN_DOWN
  - LL_TIM_INDEX_POSITION_DOWN_UP
  - LL_TIM_INDEX_POSITION_UP_DOWN
  - LL_TIM_INDEX_POSITION_UP_UP
  - LL_TIM_INDEX_POSITION_DOWN
  - LL_TIM_INDEX_POSITION_UP

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_GetIndexPositionning

LL_TIM_ConfigIDX

Function name

__STATIC_INLINE void LL_TIM_ConfigIDX (TIM_TypeDef * TIMx, uint32_t Configuration)

Function description

Configure encoder index.

Parameters

- **TIMx:** Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_INDEX_UP or LL_TIM_INDEX_DOWN or LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_BLANK_ALWAYS or LL_TIM_INDEX_BLANK_TI3 or LL_TIM_INDEX_BLANK_TI4
  - LL_TIM_INDEX_ALL or LL_TIM_INDEX_FIRST_ONLY
  - LL_TIM_INDEX_POSITION_DOWN_DOWN or ... or LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_ConfigIDX
- ECR IBLK LL_TIM_ConfigIDX
- ECR FIDX LL_TIM_ConfigIDX
- ECR IPOS LL_TIM_ConfigIDX

LL_TIM_SetRemap

Function name

__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)

Function description

Remap TIM inputs (input channel, internal/external triggers).

Parameters

- **TIMx:** Timer instance
- **Remap:** Remap param depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of TISEL registers.

Return values

- **None:**

Notes

- Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.

Reference Manual to LL API cross reference:

- TIM1_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI2SEL LL_TIM_SetRemap
- TIM2_TISEL TI4SEL LL_TIM_SetRemap
- TIM3_TISEL TI1SEL LL_TIM_SetRemap
- TIM3_TISEL TI2SEL LL_TIM_SetRemap
- TIM4_TISEL TI1SEL LL_TIM_SetRemap
- TIM4_TISEL TI2SEL LL_TIM_SetRemap
- TIM5_TISEL TI1SEL LL_TIM_SetRemap
- TIM5_TISEL TI2SEL LL_TIM_SetRemap
- TIM8_TISEL TI1SEL LL_TIM_SetRemap
- TIM12_TISEL TI1SEL LL_TIM_SetRemap
- TIM13_TISEL TI1SEL LL_TIM_SetRemap
- TIM14_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI2SEL LL_TIM_SetRemap
- TIM16_TISEL TI1SEL LL_TIM_SetRemap
- TIM17_TISEL TI1SEL LL_TIM_SetRemap
-

LL_TIM_SetOCRefClearInputSource

Function name

__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)

Function description

Set the OCREF clear input source.

Parameters

- **TIMx:** Timer instance
- **OCRefClearInputSource:** This parameter can be one of the following values:
  - LL_TIM_OCREF_CLR_INT_OCREF_CLR
  - LL_TIM_OCREF_CLR_INT_ETR

Return values

- **None:**

Notes

- The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT
- This function can only be used in Output compare and PWM modes.

Reference Manual to LL API cross reference:

- SMCR OCCS LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Clear the update interrupt flag (UIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 interrupt flag (CC1F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 interrupt flag (CC2F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 interrupt flag (CC3F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 interrupt flag (CC4F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 5 interrupt flag (CC5F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 6 interrupt flag (CC6F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)

Function description

Clear the commutation interrupt flag (COMIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (const TIM_TypeDef * TIMx)

Function description

Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Clear the trigger interrupt flag (TIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)

Function description

Clear the break interrupt flag (BIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Clear the break 2 interrupt flag (B2IF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Clear the system break interrupt flag (SBIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether system break interrupt flag (SBIF) is set (system break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_ClearFlag_TERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TERR (TIM_TypeDef * TIMx)

Function description

Clear the transition error interrupt flag (TERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_ClearFlag_TERR

LL_TIM_IsActiveFlag_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether transition error interrupt flag (TERRF) is set (transition error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_IsActiveFlag_TERR

LL_TIM_ClearFlag_IERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IERR (TIM_TypeDef * TIMx)

Function description

Clear the index error interrupt flag (IERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_ClearFlag_IERR

LL_TIM_IsActiveFlag_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether index error interrupt flag (IERRF) is set (index error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_IsActiveFlag_IERR

LL_TIM_ClearFlag_DIR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_DIR (TIM_TypeDef * TIMx)

Function description

Clear the direction change interrupt flag (DIRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_ClearFlag_DIR

LL_TIM_IsActiveFlag_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_DIR (const TIM_TypeDef * TIMx)

Function description

Indicate whether direction change interrupt flag (DIRF) is set (direction change interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_IsActiveFlag_DIR

LL_TIM_ClearFlag_IDX

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IDX (TIM_TypeDef * TIMx)

Function description

Clear the index interrupt flag (IDXF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_ClearFlag_IDX

LL_TIM_IsActiveFlag_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IDX (const TIM_TypeDef * TIMx)

Function description

Indicate whether index interrupt flag (IDXF) is set (index interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_IsActiveFlag_IDX

LL_TIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update interrupt (UIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name

__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name

__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation interrupt (COMIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)

Function description

Enable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)

Function description

Disable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (const TIM_TypeDef * TIMx)

Function description

Indicates whether the break interrupt (BIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TERR (TIM_TypeDef * TIMx)

Function description

Enable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_EnableIT_TERR

LL_TIM_DisableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TERR (TIM_TypeDef * TIMx)

Function description

Disable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_DisableIT_TERR

LL_TIM_IsEnabledIT_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the transition error interrupt (TERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_IsEnabledIT_TERR

LL_TIM_EnableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IERR (TIM_TypeDef * TIMx)

Function description

Enable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_EnableIT_IERR

LL_TIM_DisableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IERR (TIM_TypeDef * TIMx)

Function description

Disable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_DisableIT_IERR

LL_TIM_IsEnabledIT_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index error interrupt (IERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_IsEnabledIT_IERR

LL_TIM_EnableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_DIR (TIM_TypeDef * TIMx)

Function description

Enable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_EnableIT_DIR

LL_TIM_DisableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_DIR (TIM_TypeDef * TIMx)

Function description

Disable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_DisableIT_DIR

LL_TIM_IsEnabledIT_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_DIR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the direction change interrupt (DIRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_IsEnabledIT_DIR

LL_TIM_EnableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IDX (TIM_TypeDef * TIMx)

Function description

Enable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_EnableIT_IDX

LL_TIM_DisableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IDX (TIM_TypeDef * TIMx)

Function description

Disable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_DisableIT_IDX

LL_TIM_IsEnabledIT_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IDX (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index interrupt (IDXIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_IsEnabledIT_IDX

LL_TIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update DMA request (UDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation DMA request (COMDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)

Function description

Generate break 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name

ErrorStatus LL_TIM_DeInit (const TIM_TypeDef * TIMx)

Function description

Set TIMx registers to their reset values.

Parameters

- **TIMx:** Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name

void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Set the fields of the time base unit configuration data structure to their default values.

Parameters

- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values

- **None:**

LL_TIM_Init

Function name

ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, const LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Configure the TIMx time base unit.

Parameters

- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_OC_StructInit

Function name

void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters

- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values

- **None:**

LL_TIM_OC_Init

Function name

ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Configure the TIMx output channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name

void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)

Function description

Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters

- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values

- **None:**

LL_TIM_IC_Init

Function name

ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)

Function description

Configure the TIMx input channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name

void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Fills each TIM_EncoderInitStruct field with its default value.

Parameters

- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values

- **None:**

LL_TIM_ENCODER_Init

Function name

ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, const LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Configure the encoder interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name

void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters

- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values

- **None:**

LL_TIM_HALLSENSOR_Init

Function name

ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, const LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Configure the Hall sensor interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

Notes

- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.
- OC2REF is selected as trigger output on TRGO.
- LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name

void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Set the fields of the Break and Dead Time configuration data structure to their default values.

Parameters

- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **None:**

LL_TIM_BDTR_Init

Function name

ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, const LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Configure the Break and Dead Time feature of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Break and Dead Time is initialized
  - ERROR: not applicable

Notes

- As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

114.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

114.3.1 TIM

TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI

ICx is mapped on TIx

LL_TIM_ACTIVEINPUT_INDIRECTTI

ICx is mapped on TIy

LL_TIM_ACTIVEINPUT_TRC

ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

LL_TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW

BRK BKIN input is active low

LL_TIM_BKIN_POLARITY_HIGH

BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN

BKIN input from AF controller

BREAK2 AF MODE

LL_TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

Break2 Enable

LL_TIM_BREAK2_DISABLE

Break2 function disabled

LL_TIM_BREAK2_ENABLE

Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK2_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK2_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK2_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK2_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK2_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK2_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK2_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK2_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK2_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK2_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK2_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK2_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK2_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK2_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW

Break input BRK2 is active low

LL_TIM_BREAK2_POLARITY_HIGH

Break input BRK2 is active high

BREAK AF MODE

LL_TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

Break Enable

LL_TIM_BREAK_DISABLE

Break function disabled

LL_TIM_BREAK_ENABLE

Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN input

LL_TIM_BREAK_INPUT_BKIN2

TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW

Break input BRK is active low

LL_TIM_BREAK_POLARITY_HIGH

Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC

CCx DMA request sent when CCx event occurs

LL_TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY

Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_LPTIM_CHANNEL_CH1

LPTIM input/output channel 1

LL_LPTIM_CHANNEL_CH2

LPTIM input/output channel 2

LL_TIM_CHANNEL_CH1

Timer input/output channel 1

LL_TIM_CHANNEL_CH1N

Timer complementary output channel 1

LL_TIM_CHANNEL_CH2

Timer input/output channel 2

LL_TIM_CHANNEL_CH2N

Timer complementary output channel 2

LL_TIM_CHANNEL_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH3N

Timer complementary output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

LL_TIM_CHANNEL_CH4N

Timer complementary output channel 4

LL_TIM_CHANNEL_CH5

Timer output channel 5

LL_TIM_CHANNEL_CH6

Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1

tDTS=tCK_INT

LL_TIM_CLOCKDIVISION_DIV2

tDTS=2*tCK_INT

LL_TIM_CLOCKDIVISION_DIV4

tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL

The timer is clocked by the internal clock provided from the RCC

LL_TIM_CLOCKSOURCE_EXT_MODE1

Counter counts at each rising or falling edge on a selected input

LL_TIM_CLOCKSOURCE_EXT_MODE2

Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP

Timer counter counts up

LL_TIM_COUNTERDIRECTION_DOWN

Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP

Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN

Counter used as downcounter

LL_TIM_COUNTERMODE_CENTER_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.

LL_TIM_COUNTERMODE_CENTER_UP

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up

LL_TIM_COUNTERMODE_CENTER_UP_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1

TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CR2

TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SMCR

TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DIER

TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SR

TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_EGR

TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR1

TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR2

TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCER

TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CNT

TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_PSC

TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ARR

TIMx_ARR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_RCR

TIMx_RCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR1

TIMx_CCR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR2

TIMx_CCR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR3

TIMx_CCR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR4

TIMx_CCR4 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_BDTR

TIMx_BDTR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR5

TIMx_CCR5 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR6

TIMx_CCR6 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR3

TIMx_CCMR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DTR2

TIMx_DTR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ECR

TIMx_ECR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_TISEL

TIMx_TISEL register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF1

TIMx_AF1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF2

TIMx_AF2 register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER

Transfer is done to 1 register starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_2TRANSFERS

Transfer is done to 2 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_3TRANSFERS

Transfer is done to 3 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_4TRANSFERS

Transfer is done to 4 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_5TRANSFERS

Transfer is done to 5 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_6TRANSFERS

Transfer is done to 6 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_7TRANSFERS

Transfer is done to 7 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_8TRANSFERS

Transfer is done to 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_19TRANSFERS

Transfer is done to 19 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_20TRANSFERS

Transfer is done to 20 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_21TRANSFERS

Transfer is done to 21 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_22TRANSFERS

Transfer is done to 22 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_23TRANSFERS

Transfer is done to 23 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_24TRANSFERS

Transfer is done to 24 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_25TRANSFERS

Transfer is done to 25 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_26TRANSFERS

Transfer is done to 26 registers starting from the DMA burst base address

DMA Burst Source

LL_TIM_DMA_UPDATE

Transfer source is update event

LL_TIM_DMA_CC1

Transfer source is CC1 event

LL_TIM_DMA_CC2

Transfer source is CC2 event

LL_TIM_DMA_CC3

Transfer source is CC3 event

LL_TIM_DMA_CC4

Transfer source is CC4 event

LL_TIM_DMA_COM

Transfer source is COM event

LL_TIM_DMA_TRIGGER

Transfer source is trigger event

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1

Quadrature encoder mode 1, x2 mode - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level

LL_TIM_ENCODERMODE_X2_TI2

Quadrature encoder mode 2, x2 mode - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level

LL_TIM_ENCODERMODE_X4_TI12

Quadrature encoder mode 3, x4 mode - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction - x2 mode

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

LL_TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

LL_TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_ETR_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_ETR_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_ETR_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_ETR_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_ETR_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_ETR_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_ETR_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_ETR_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_ETR_FILTER_FDIV8_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_ETR_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_ETR_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_ETR_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED

ETR is non-inverted, active at high level or rising edge

LL_TIM_ETR_POLARITY_INVERTED

ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1

ETR prescaler OFF

LL_TIM_ETR_PRESCALER_DIV2

ETR frequency is divided by 2

LL_TIM_ETR_PRESCALER_DIV4

ETR frequency is divided by 4

LL_TIM_ETR_PRESCALER_DIV8

ETR frequency is divided by 8

first index selection

LL_TIM_INDEX_ALL

Index is always active

LL_TIM_INDEX_FIRST_ONLY

The first Index only resets the counter

Get Flags Defines

LL_TIM_SR_UIF

Update interrupt flag

LL_TIM_SR_CC1IF

Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF

Capture/compare 2 interrupt flag

LL_TIM_SR_CC3IF

Capture/compare 3 interrupt flag

LL_TIM_SR_CC4IF

Capture/compare 4 interrupt flag

LL_TIM_SR_CC5IF

Capture/compare 5 interrupt flag

LL_TIM_SR_CC6IF

Capture/compare 6 interrupt flag

LL_TIM_SR_COMIF

COM interrupt flag

LL_TIM_SR_TIF

Trigger interrupt flag

LL_TIM_SR_BIF

Break interrupt flag

LL_TIM_SR_B2IF

Second break interrupt flag

LL_TIM_SR_CC1OF

Capture/Compare 1 overcapture flag

LL_TIM_SR_CC2OF

Capture/Compare 2 overcapture flag

LL_TIM_SR_CC3OF

Capture/Compare 3 overcapture flag

LL_TIM_SR_CC4OF

Capture/Compare 4 overcapture flag

LL_TIM_SR_SBIF

System Break interrupt flag

LL_TIM_SR_IDXF

Index interrupt flag

LL_TIM_SR_DIRF

Direction Change interrupt flag

LL_TIM_SR_IERRF

Index Error flag

LL_TIM_SR_TERRF

Transition Error flag

GROUPCH5

LL_TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

LL_TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

LL_TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1

No prescaler, capture is done each time an edge is detected on the capture input

LL_TIM_ICPSC_DIV2

Capture is done once every 2 events

LL_TIM_ICPSC_DIV4

Capture is done once every 4 events

LL_TIM_ICPSC_DIV8

Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_IC_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_IC_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_IC_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_IC_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_IC_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_IC_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_IC_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_IC_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_IC_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_IC_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_IC_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_IC_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_IC_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_IC_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_IC_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING

The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

index blanking selection

LL_TIM_INDEX_BLANK_ALWAYS

Index always active

LL_TIM_INDEX_BLANK_TI3

Index disabled when TI3 input is active, as per CC3P bitfield

LL_TIM_INDEX_BLANK_TI4

Index disabled when TI4 input is active, as per CC4P bitfield

index direction selection

LL_TIM_INDEX_UP_DOWN

Index resets the counter whatever the direction

LL_TIM_INDEX_UP

Index resets the counter when up-counting only

LL_TIM_INDEX_DOWN

Index resets the counter when down-counting only

index positioning selection

LL_TIM_INDEX_POSITION_DOWN_DOWN

Index resets the counter when AB = 00

LL_TIM_INDEX_POSITION_DOWN_UP

Index resets the counter when AB = 01

LL_TIM_INDEX_POSITION_UP_DOWN

Index resets the counter when AB = 10

LL_TIM_INDEX_POSITION_UP_UP

Index resets the counter when AB = 11

LL_TIM_INDEX_POSITION_DOWN

Index resets the counter when clock is 0

LL_TIM_INDEX_POSITION_UP

Index resets the counter when clock is 1

IT Defines

LL_TIM_DIER_UIE

Update interrupt enable

LL_TIM_DIER_CC1IE

Capture/compare 1 interrupt enable

LL_TIM_DIER_CC2IE

Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE

Capture/compare 3 interrupt enable

LL_TIM_DIER_CC4IE

Capture/compare 4 interrupt enable

LL_TIM_DIER_COMIE

COM interrupt enable

LL_TIM_DIER_TIE

Trigger interrupt enable

LL_TIM_DIER_BIE

Break interrupt enable

LL_TIM_DIER_IDXIE

Index interrupt enable

LL_TIM_DIER_DIRIE

Direction Change interrupt enable

LL_TIM_DIER_IERRIE

Index Error interrupt enable

LL_TIM_DIER_TERRIE

Transition Error interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF

LOCK OFF - No bit is write protected

LL_TIM_LOCKLEVEL_1

LOCK Level 1

LL_TIM_LOCKLEVEL_2

LOCK Level 2

LL_TIM_LOCKLEVEL_3

LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW

OCx=0 (after a dead-time if OC is implemented) when MOE=0

LL_TIM_OCIDLESTATE_HIGH

OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN

The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level

LL_TIM_OCMODE_ACTIVE

OCyREF is forced high on compare match

LL_TIM_OCMODE_INACTIVE

OCyREF is forced low on compare match

LL_TIM_OCMODE_TOGGLE

OCyREF toggles on compare match

LL_TIM_OCMODE_FORCED_INACTIVE

OCyREF is forced low

LL_TIM_OCMODE_FORCED_ACTIVE

OCyREF is forced high

LL_TIM_OCMODE_PWM1

In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.

LL_TIM_OCMODE_PWM2

In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

LL_TIM_OCMODE_RETRIG_OPM1

Retrigerrable OPM mode 1

LL_TIM_OCMODE_RETRIG_OPM2

Retrigerrable OPM mode 2

LL_TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

LL_TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

LL_TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

LL_TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

LL_TIM_OCMODE_PULSE_ON_COMPARE

Pulse on Compare mode

LL_TIM_OCMODE_DIRECTION_OUTPUT

Direction output mode

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH

OCxactive high

LL_TIM_OCPOLARITY_LOW

OCxactive low

OCREF clear input selection

LL_TIM_OCREF_CLR_INT_OCREF_CLR

OCREF_CLR_INT is connected to the OCREF_CLR input

LL_TIM_OCREF_CLR_INT_ETR

OCREF_CLR_INT is connected to ETRF

Output Configuration State

LL_TIM_OCSTATE_DISABLE

OCx is not active

LL_TIM_OCSTATE_ENABLE

OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE

Counter stops counting at the next update event

LL_TIM_ONEPULSEMODE_REPETITIVE

Counter is not stopped at update event

OSSI

LL_TIM_OSSI_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE

When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Pulse on compare pulse width prescaler

LL_TIM_PWPRSC_X1

Pulse on compare pulse width prescaler 1

LL_TIM_PWPRSC_X2

Pulse on compare pulse width prescaler 2

LL_TIM_PWPRSC_X4

Pulse on compare pulse width prescaler 4

LL_TIM_PWPRSC_X8

Pulse on compare pulse width prescaler 8

LL_TIM_PWPRSC_X16

Pulse on compare pulse width prescaler 16

LL_TIM_PWPRSC_X32

Pulse on compare pulse width prescaler 32

LL_TIM_PWPRSC_X64

Pulse on compare pulse width prescaler 64

LL_TIM_PWPRSC_X128

Pulse on compare pulse width prescaler 128

Slave Mode

LL_TIM_SLAVEMODE_DISABLED

Slave mode disabled

LL_TIM_SLAVEMODE_RESET

Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter

LL_TIM_SLAVEMODE_GATED

Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high

LL_TIM_SLAVEMODE_TRIGGER

Trigger Mode - The counter starts at a rising edge of the trigger TRGI

LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops and is reset) as soon as the trigger becomes low.Both startand stop of the counter are controlled.

SMS Preload Source

LL_TIM_SMSPS_TIMUPDATE

The SMS preload transfer is triggered by the Timer's Update event

LL_TIM_SMSPS_INDEX

The SMS preload transfer is triggered by the Index event

TIM12 External Input Ch1 Remap

LL_TIM_TIM12_TI1_RMP_GPIO

TIM12_TI1 is connected to GPIO

LL_TIM_TIM12_TI1_RMP_HSI_1024

TIM12_TI1 is connected to HSI 1024

LL_TIM_TIM12_TI1_RMP_CSI_128

TIM12_TI1 is connected to CSI 128

TIM15 External Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15_TI1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_TIM2

TIM15_TI1 is connected to TIM2

LL_TIM_TIM15_TI1_RMP_TIM3

TIM15_TI1 is connected to TIM3

LL_TIM_TIM15_TI1_RMP_TIM4

TIM15_TI1 is connected to TIM4

LL_TIM_TIM15_TI1_RMP_LSE

TIM15_TI1 is connected to LSE

LL_TIM_TIM15_TI1_RMP_CSI_128

TIM15_TI1 is connected to CSI 128

LL_TIM_TIM15_TI1_RMP_MCO2

TIM15_TI1 is connected to MCO2

TIM15 External Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15_TI1 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_TIM2

TIM15_TI2 is connected to TIM2

LL_TIM_TIM15_TI2_RMP_TIM3

TIM15_TI2 is connected to TIM3

LL_TIM_TIM15_TI2_RMP_TIM4

TIM15_TI2 is connected to TIM4

TIM16 External Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16_TI1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_LSI

TIM16_TI1 is connected to LSI

LL_TIM_TIM16_TI1_RMP_LSE

TIM16_TI1 is connected to LSE

LL_TIM_TIM16_TI1_RMP_RTC_WKUP

TIM16_TI1 is connected to RTC

TIM17 External Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17_TI1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_HSE_1MHZ

TIM17_TI1 is connected to HSE 1MHZ

LL_TIM_TIM17_TI1_RMP_MCO1

TIM17_TI1 is connected to MCO1

External Trigger Source TIM1

LL_TIM_TIM1_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1

ADC1 analog watchdog 1

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2

ADC1 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3

ADC1 analog watchdog 3

TIM1 External Input Ch1 Remap

LL_TIM_TIM1_TI1_RMP_GPIO

External Trigger Source TIM2

LL_TIM_TIM2_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM2_ETRSOURCE_LSE

ETR input is connected to LSE

LL_TIM_TIM2_ETRSOURCE_SAI1_FSA

ETR input is connected to SAI1_FSA

LL_TIM_TIM2_ETRSOURCE_SAI1_FSB

ETR input is connected to SAI1_FSB

LL_TIM_TIM2_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM2_ETRSOURCE_TIM4_ETR

ETR input is connected to TIM4 ETR

LL_TIM_TIM2_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

LL_TIM_TIM2_ETRSOURCE_ETH_PPS

ETR input is connected to ETH PPS

TIM2 External Input Ch1 Remap

LL_TIM_TIM2_TI1_RMP_GPIO

TIM2_TI1 is connected to GPIO

LL_TIM_TIM2_TI1_RMP_ETH_PPS

TIM2_TI1 is connected to ETH_ PPS

TIM2 External Input Ch2 Remap

LL_TIM_TIM2_TI2_RMP_GPIO

TIM2_TI1 is connected to GPIO

TIM2 External Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO

TIM2_TI4 is connected to GPIO

External Trigger Source TIM3

LL_TIM_TIM3_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM3_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM2 ETR

LL_TIM_TIM3_ETRSOURCE_TIM4_ETR

ETR input is connected to TIM4 ETR

LL_TIM_TIM3_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

LL_TIM_TIM3_ETRSOURCE_ETH_PPS

ETR input is connected to ETH PPS

TIM3 External Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO

TIM3_TI1 is connected to GPIO

LL_TIM_TIM3_TI1_RMP_ETH_PPS

TIM2_TI1 is connected to ETH PPS

TIM3 External Input Ch2 Remap

LL_TIM_TIM3_TI2_RMP_GPIO

TIM3_TI2 is connected to GPIO

External Trigger Source TIM4

LL_TIM_TIM4_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM4_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM4_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM4 ETR

LL_TIM_TIM4_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

External Trigger Source TIM5

LL_TIM_TIM5_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM5_ETRSOURCE_SAI2_FSA

ETR input is connected to SAI2_FSA

LL_TIM_TIM5_ETRSOURCE_SAI2_FSB

ETR input is connected to SAI2_FSB

LL_TIM_TIM5_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM2 ETR

LL_TIM_TIM5_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM5_ETRSOURCE_TIM4_ETR

ETR input is connected to TIM4 ETR

External Trigger Source TIM8

LL_TIM_TIM8_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1

ADC4 analog watchdog 1

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2

ADC4 analog watchdog 2

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3

ADC4 analog watchdog 3

Trigger Output

LL_TIM_TRGO_RESET

UG bit from the TIMx_EGR register is used as trigger output

LL_TIM_TRGO_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output

LL_TIM_TRGO_UPDATE

Update event is used as trigger output

LL_TIM_TRGO_CC1IF

CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF

OC1REF signal is used as trigger output

LL_TIM_TRGO_OC2REF

OC2REF signal is used as trigger output

LL_TIM_TRGO_OC3REF

OC3REF signal is used as trigger output

LL_TIM_TRGO_OC4REF

OC4REF signal is used as trigger output

LL_TIM_TRGO_ENCODERCLK

Encoder clock signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET

UG bit from the TIMx_EGR register is used as trigger output 2

LL_TIM_TRGO2_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output 2

LL_TIM_TRGO2_UPDATE

Update event is used as trigger output 2

LL_TIM_TRGO2_CC1F

CC1 capture or a compare match is used as trigger output 2

LL_TIM_TRGO2_OC1

OC1REF signal is used as trigger output 2

LL_TIM_TRGO2_OC2

OC2REF signal is used as trigger output 2

LL_TIM_TRGO2_OC3

OC3REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4

OC4REF signal is used as trigger output 2

LL_TIM_TRGO2_OC5

OC5REF signal is used as trigger output 2

LL_TIM_TRGO2_OC6

OC6REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4_RISINGFALLING

OC4REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC6_RISINGFALLING

OC6REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_RISING

OC4REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_FALLING

OC4REF rising or OC6REF falling edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_RISING

OC5REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0

Internal Trigger 0 (ITR0) is used as trigger input

LL_TIM_TS_ITR1

Internal Trigger 1 (ITR1) is used as trigger input

LL_TIM_TS_ITR2

Internal Trigger 2 (ITR2) is used as trigger input

LL_TIM_TS_ITR3

Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_ITR4

Internal Trigger 4 (ITR4) is used as trigger input

LL_TIM_TS_ITR5

Internal Trigger 5 (ITR5) is used as trigger input

LL_TIM_TS_ITR6

Internal Trigger 6 (ITR6) is used as trigger input

LL_TIM_TS_ITR7

Internal Trigger 7 (ITR7) is used as trigger input

LL_TIM_TS_ITR8

Internal Trigger 8 (ITR8) is used as trigger input

LL_TIM_TS_ITR9

Internal Trigger 9 (ITR9) is used as trigger input

LL_TIM_TS_ITR10

Internal Trigger 10 (ITR10) is used as trigger input

LL_TIM_TS_ITR11

Internal Trigger 11 (ITR11) is used as trigger input

LL_TIM_TS_ITR12

Internal Trigger 12 (ITR12) is used as trigger input

LL_TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED) is used as trigger input

LL_TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2

Filtered Timer Input 2 (TI12P2) is used as trigger input

LL_TIM_TS_ETRF

Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR

Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request

LL_TIM_UPDATESOURCE_COUNTER

Only counter overflow/underflow generates an update request

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

TIM Exported Macros

__LL_TIM_GETFLAG_UIFCPY

Description:

- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:

- __CNT__: Counter value

Return value:

- UIF: status bit

Notes:

- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_ARR_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR_DITHER (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_DELAY_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY_DITHER (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE_DITHER (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

115 LL UCPD Generic Driver

115.1 UCPD Firmware driver registers structures

115.1.1 LL_UCPD_InitTypeDef

LL_UCPD_InitTypeDef is defined in the stm32h5xx_ll_ucpd.h

Data Fields

- ***uint32_t psc_ucpdclk***
- ***uint32_t transwin***
- ***uint32_t IfrGap***
- ***uint32_t HbitClockDiv***

Field Documentation

- ***uint32_t LL_UCPD_InitTypeDef::psc_ucpdclk***
  Specify the prescaler for the UCPD clock. This parameter can be a value of ***UCPD_LL_EC_PSC***. This feature can be modified afterwards using function **LL_UCPD_SetPSCClk()**.
- ***uint32_t LL_UCPD_InitTypeDef::transwin***
  Specify the number of cycles (minus 1) of the half bit clock (see HBITCLKDIV) to achieve a legal tTransitionWindow (set according to peripheral clock to define an interval of between 12 and 20 us). This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This value can be modified afterwards using function **LL_UCPD_SetTransWin()**.
- ***uint32_t LL_UCPD_InitTypeDef::IfrGap***
  Specify the definition of the clock divider (minus 1) in order to generate tInterframeGap from the peripheral clock. This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This feature can be modified afterwards using function **LL_UCPD_SetIfrGap()**.
- ***uint32_t LL_UCPD_InitTypeDef::HbitClockDiv***
  Specify the number of cycles (minus one) at UCPD peripheral for a half bit clock e.g. program 3 for a bit clock that takes 8 cycles of the peripheral clock : "UCPD1_CLK". This parameter can be a value between Min_Data=0x0 and Max_Data=0x3F. This feature can be modified using function **LL_UCPD_SetHbitClockDiv()**.

115.2 UCPD Firmware driver API description

The following section lists the various functions of the UCPD library.

115.2.1 Detailed description of functions

LL_UCPD_Enable

Function name

__STATIC_INLINE void LL_UCPD_Enable (UCPD_TypeDef * UCPDx)

Function description

Enable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Enable

LL_UCPD_Disable

Function name

__STATIC_INLINE void LL_UCPD_Disable (UCPD_TypeDef * UCPDx)

Function description

Disable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Notes

- When disabling the UCPD, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Disable

LL_UCPD_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabled (UCPD_TypeDef const *const UCPDx)

Function description

Check if UCPD peripheral is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_IsEnabled

LL_UCPD_SetRxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrderSet (UCPD_TypeDef * UCPDx, uint32_t OrderSet)

Function description

Set the receiver ordered set detection enable.

Parameters

- **UCPDx:** UCPD Instance
- **OrderSet:** This parameter can be combination of the following values:
  - LL_UCPD_ORDERSET_SOP
  - LL_UCPD_ORDERSET_SOP1
  - LL_UCPD_ORDERSET_SOP2
  - LL_UCPD_ORDERSET_HARDRST
  - LL_UCPD_ORDERSET_CABLERST
  - LL_UCPD_ORDERSET_SOP1_DEBUG
  - LL_UCPD_ORDERSET_SOP2_DEBUG
  - LL_UCPD_ORDERSET_SOP_EXT1
  - LL_UCPD_ORDERSET_SOP_EXT2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXORDSETEN LL_UCPD_SetRxOrderSet

LL_UCPD_SetPSCClk

Function name

__STATIC_INLINE void LL_UCPD_SetPSCClk (UCPD_TypeDef * UCPDx, uint32_t Psc)

Function description

Set the prescaler for ucpd clock.

Parameters

- **UCPDx:** UCPD Instance
- **Psc:** This parameter can be one of the following values:
  - LL_UCPD_PSC_DIV1
  - LL_UCPD_PSC_DIV2
  - LL_UCPD_PSC_DIV4
  - LL_UCPD_PSC_DIV8
  - LL_UCPD_PSC_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDCLK LL_UCPD_SetPSCClk

LL_UCPD_SetTransWin

Function name

__STATIC_INLINE void LL_UCPD_SetTransWin (UCPD_TypeDef * UCPDx, uint32_t TransWin)

Function description

Set the number of cycles (minus 1) of the half bit clock.

Parameters

- **UCPDx:** UCPD Instance
- **TransWin:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TRANSWIN LL_UCPD_SetTransWin

LL_UCPD_SetIfrGap

Function name

__STATIC_INLINE void LL_UCPD_SetIfrGap (UCPD_TypeDef * UCPDx, uint32_t IfrGap)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **IfrGap:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 IFRGAP LL_UCPD_SetIfrGap

LL_UCPD_SetHbitClockDiv

Function name

__STATIC_INLINE void LL_UCPD_SetHbitClockDiv (UCPD_TypeDef * UCPDx, uint32_t HbitClock)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **HbitClock:** a value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 HBITCLKDIV LL_UCPD_SetHbitClockDiv

LL_UCPD_RxAnalogFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterEnable (UCPD_TypeDef * UCPDx)

Function description

Enable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterEnable

LL_UCPD_RxAnalogFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterDisable (UCPD_TypeDef * UCPDx)

Function description

Disable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterDisable

LL_UCPD_WakeUpEnable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpEnable (UCPD_TypeDef * UCPDx)

Function description

Enable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpEnable

LL_UCPD_WakeUpDisable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpDisable (UCPD_TypeDef * UCPDx)

Function description

Disable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpDisable

LL_UCPD_ForceClockEnable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockEnable (UCPD_TypeDef * UCPDx)

Function description

Force clock enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockEnable

LL_UCPD_ForceClockDisable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockDisable (UCPD_TypeDef * UCPDx)

Function description

Force clock disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockDisable

LL_UCPD_RxFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterEnable (UCPD_TypeDef * UCPDx)

Function description

RxFilter enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterEnable

LL_UCPD_RxFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterDisable (UCPD_TypeDef * UCPDx)

Function description

RxFilter disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterDisable

LL_UCPD_TypeCDetectionCC2Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Enable

LL_UCPD_TypeCDetectionCC2Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Disable

LL_UCPD_TypeCDetectionCC1Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Enable

LL_UCPD_TypeCDetectionCC1Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Disable

LL_UCPD_VconnDischargeEnable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeEnable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeEnable

LL_UCPD_VconnDischargeDisable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeDisable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeDisable

LL_UCPD_SignalFRSTX

Function name

__STATIC_INLINE void LL_UCPD_SignalFRSTX (UCPD_TypeDef * UCPDx)

Function description

Signal Fast Role Swap request.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSTX LL_UCPD_VconnDischargeDisable

LL_UCPD_FRSDetectionEnable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionEnable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionEnable

LL_UCPD_FRSDetectionDisable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionDisable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionDisable

LL_UCPD_SetccEnable

Function name

__STATIC_INLINE void LL_UCPD_SetccEnable (UCPD_TypeDef * UCPDx, uint32_t CCEnable)

Function description

Set cc enable.

Parameters

- **UCPDx:** UCPD Instance
- **CCEnable:** This parameter can be one of the following values:
  - LL_UCPD_CCENABLE_NONE
  - LL_UCPD_CCENABLE_CC1
  - LL_UCPD_CCENABLE_CC2
  - LL_UCPD_CCENABLE_CC1CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1VCONNEN LL_UCPD_SetccEnable

LL_UCPD_SetSNKRole

Function name

__STATIC_INLINE void LL_UCPD_SetSNKRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SNK role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSNKRole

LL_UCPD_SetSRCRole

Function name

__STATIC_INLINE void LL_UCPD_SetSRCRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SRC role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSRCRole

LL_UCPD_GetRole

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetRole (UCPD_TypeDef const *const UCPDx)

Function description

Get UCPD Role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_UCPD_ROLE_SNK
  - LL_UCPD_ROLE_SRC

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_GetRole

LL_UCPD_SetRpResistor

Function name

__STATIC_INLINE void LL_UCPD_SetRpResistor (UCPD_TypeDef * UCPDx, uint32_t Resistor)

Function description

Set Rp resistor.

Parameters

- **UCPDx:** UCPD Instance
- **Resistor:** This parameter can be one of the following values:
  - LL_UCPD_RESISTOR_DEFAULT
  - LL_UCPD_RESISTOR_1_5A
  - LL_UCPD_RESISTOR_3_0A
  - LL_UCPD_RESISTOR_NONE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANASUBMODE LL_UCPD_SetRpResistor

LL_UCPD_SetCCPin

Function name

__STATIC_INLINE void LL_UCPD_SetCCPin (UCPD_TypeDef * UCPDx, uint32_t CCPin)

Function description

Set CC pin.

Parameters

- **UCPDx:** UCPD Instance
- **CCPin:** This parameter can be one of the following values:
  - LL_UCPD_CCPIN_CC1
  - LL_UCPD_CCPIN_CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYCCSEL LL_UCPD_SetCCPin

LL_UCPD_RxEnable

Function name

__STATIC_INLINE void LL_UCPD_RxEnable (UCPD_TypeDef * UCPDx)

Function description

Rx enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxEnable

LL_UCPD_RxDisable

Function name

__STATIC_INLINE void LL_UCPD_RxDisable (UCPD_TypeDef * UCPDx)

Function description

Rx disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxDisable

LL_UCPD_SetRxMode

Function name

__STATIC_INLINE void LL_UCPD_SetRxMode (UCPD_TypeDef * UCPDx, uint32_t RxMode)

Function description

Set Rx mode.

Parameters

- **UCPDx:** UCPD Instance
- **RxMode:** This parameter can be one of the following values:
  - LL_UCPD_RXMODE_NORMAL
  - LL_UCPD_RXMODE_BIST_TEST_DATA

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXMODE LL_UCPD_SetRxMode

LL_UCPD_SendHardReset

Function name

__STATIC_INLINE void LL_UCPD_SendHardReset (UCPD_TypeDef * UCPDx)

Function description

Send Hard Reset.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXHRST LL_UCPD_SendHardReset

LL_UCPD_SendMessage

Function name

__STATIC_INLINE void LL_UCPD_SendMessage (UCPD_TypeDef * UCPDx)

Function description

Send message.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXSEND LL_UCPD_SendMessage

LL_UCPD_SetTxMode

Function name

__STATIC_INLINE void LL_UCPD_SetTxMode (UCPD_TypeDef * UCPDx, uint32_t TxMode)

Function description

Set Tx mode.

Parameters

- **UCPDx:** UCPD Instance
- **TxMode:** This parameter can be one of the following values:
  - LL_UCPD_TXMODE_NORMAL
  - LL_UCPD_TXMODE_CABLE_RESET
  - LL_UCPD_TXMODE_BIST_CARRIER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXMODE LL_UCPD_SetTxMode

LL_UCPD_EnableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Enable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_EnableIT_FRS

LL_UCPD_EnableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_EnableIT_TypeCEventCC2

LL_UCPD_EnableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_EnableIT_TypeCEventCC1

LL_UCPD_EnableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Enable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_EnableIT_RxMsgEnd

LL_UCPD_EnableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Enable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_EnableIT_RxOvr

LL_UCPD_EnableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Enable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_EnableIT_RxHRST

LL_UCPD_EnableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Enable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_EnableIT_RxOrderSet

LL_UCPD_EnableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Enable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_EnableIT_RxNE

LL_UCPD_EnableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Enable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_EnableIT_TxUND

LL_UCPD_EnableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_EnableIT_TxHRSTSENT

LL_UCPD_EnableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_EnableIT_TxHRSTDISC

LL_UCPD_EnableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_EnableIT_TxMSGABT

LL_UCPD_EnableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_EnableIT_TxMSGSENT

LL_UCPD_EnableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_EnableIT_TxMSGDISC

LL_UCPD_EnableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Enable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_EnableIT_TxIS

LL_UCPD_DisableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Disable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_DisableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_DisableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_DisableIT_TypeCEventCC1

LL_UCPD_DisableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Disable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_DisableIT_RxMsgEnd

LL_UCPD_DisableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Disable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_DisableIT_RxOvr

LL_UCPD_DisableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Disable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_DisableIT_RxHRST

LL_UCPD_DisableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Disable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_DisableIT_RxOrderSet

LL_UCPD_DisableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Disable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_DisableIT_RxNE

LL_UCPD_DisableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Disable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_DisableIT_TxUND

LL_UCPD_DisableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_DisableIT_TxHRSTSENT

LL_UCPD_DisableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_DisableIT_TxHRSTDISC

LL_UCPD_DisableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_DisableIT_TxMSGABT

LL_UCPD_DisableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_DisableIT_TxMSGSENT

LL_UCPD_DisableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_DisableIT_TxMSGDISC

LL_UCPD_DisableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Disable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_DisableIT_TxIS

LL_UCPD_IsEnableIT_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_IsEnableIT_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_IsEnableIT_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 TYPECEVT1IE LL_UCPD_IsEnableIT_TypeCEventCC1

LL_UCPD_IsEnableIT_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_IsEnableIT_RxMsgEnd

LL_UCPD_IsEnableIT_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_IsEnableIT_RxOvr

LL_UCPD_IsEnableIT_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_IsEnableIT_RxHRST

LL_UCPD_IsEnableIT_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_IsEnableIT_RxOrderSet

LL_UCPD_IsEnableIT_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_IsEnableIT_RxNE

LL_UCPD_IsEnableIT_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_IsEnableIT_TxUND

LL_UCPD_IsEnableIT_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_IsEnableIT_TxHRSTSENT

LL_UCPD_IsEnableIT_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_IsEnableIT_TxHRSTDISC

LL_UCPD_IsEnableIT_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_IsEnableIT_TxMSGABT

LL_UCPD_IsEnableIT_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_IsEnableIT_TxMSGSENT

LL_UCPD_IsEnableIT_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_IsEnableIT_TxMSGDISC

LL_UCPD_IsEnableIT_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_IsEnableIT_TxIS

LL_UCPD_ClearFlag_FRS

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_FRS (UCPD_TypeDef * UCPDx)

Function description

Clear FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FRSEVTIE LL_UCPD_ClearFlag_FRS

LL_UCPD_ClearFlag_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT2IE LL_UCPD_ClearFlag_TypeCEventCC2

LL_UCPD_ClearFlag_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT1IE LL_UCPD_ClearFlag_TypeCEventCC1

LL_UCPD_ClearFlag_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Clear Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXMSGENDIE LL_UCPD_ClearFlag_RxMsgEnd

LL_UCPD_ClearFlag_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Clear Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXOVRIE LL_UCPD_ClearFlag_RxOvr

LL_UCPD_ClearFlag_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Clear Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXHRSTDETIE LL_UCPD_ClearFlag_RxHRST

LL_UCPD_ClearFlag_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Clear Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXORDDETIE LL_UCPD_ClearFlag_RxOrderSet

LL_UCPD_ClearFlag_TxUND

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxUND (UCPD_TypeDef * UCPDx)

Function description

Clear TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXUNDIE LL_UCPD_ClearFlag_TxUND

LL_UCPD_ClearFlag_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTSENTIE LL_UCPD_ClearFlag_TxHRSTSENT

LL_UCPD_ClearFlag_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTDISCIE LL_UCPD_ClearFlag_TxHRSTDISC

LL_UCPD_ClearFlag_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGABTIE LL_UCPD_ClearFlag_TxMSGABT

LL_UCPD_ClearFlag_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGSENTIE LL_UCPD_ClearFlag_TxMSGSENT

LL_UCPD_ClearFlag_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGDISCIE LL_UCPD_ClearFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRSEVT LL_UCPD_IsActiveFlag_FRS

LL_UCPD_IsActiveFlag_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT2 LL_UCPD_IsActiveFlag_TypeCEventCC2

LL_UCPD_IsActiveFlag_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT1 LL_UCPD_IsActiveFlag_TypeCEventCC1

LL_UCPD_IsActiveFlag_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXMSGEND LL_UCPD_IsActiveFlag_RxMsgEnd

LL_UCPD_IsActiveFlag_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXOVR LL_UCPD_IsActiveFlag_RxOvr

LL_UCPD_IsActiveFlag_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXHRSTDET LL_UCPD_IsActiveFlag_RxHRST

LL_UCPD_IsActiveFlag_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXORDDET LL_UCPD_IsActiveFlag_RxOrderSet

LL_UCPD_IsActiveFlag_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_UCPD_IsActiveFlag_RxNE

LL_UCPD_IsActiveFlag_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXUND LL_UCPD_IsActiveFlag_TxUND

LL_UCPD_IsActiveFlag_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTSENT LL_UCPD_IsActiveFlag_TxHRSTSENT

LL_UCPD_IsActiveFlag_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTDISC LL_UCPD_IsActiveFlag_TxHRSTDISC

LL_UCPD_IsActiveFlag_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGABT LL_UCPD_IsActiveFlag_TxMSGABT

LL_UCPD_IsActiveFlag_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGSENT LL_UCPD_IsActiveFlag_TxMSGSENT

LL_UCPD_IsActiveFlag_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGDISC LL_UCPD_IsActiveFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_IsActiveFlag_TxIS

LL_UCPD_GetTypeCVstateCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC2 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC2

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC2

LL_UCPD_GetTypeCVstateCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC1 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC1

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC1

LL_UCPD_RxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_RxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMAEnable

LL_UCPD_RxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_RxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMADisable

LL_UCPD_TxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_TxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMAEnable

LL_UCPD_TxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_TxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMADisable

LL_UCPD_IsEnabledTxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledTxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Tx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_UCPD_IsEnabledTxDMA

LL_UCPD_IsEnabledRxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledRxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Rx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_UCPD_IsEnabledRxDMA

LL_UCPD_WriteTxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_WriteTxOrderSet (UCPD_TypeDef * UCPDx, uint32_t TxOrderSet)

Function description

write the orderset for Tx message

Parameters

- **UCPDx:** UCPD Instance
- **TxOrderSet:** one of the following value
  - LL_UCPD_ORDERED_SET_SOP
  - LL_UCPD_ORDERED_SET_SOP1
  - LL_UCPD_ORDERED_SET_SOP2
  - LL_UCPD_ORDERED_SET_HARD_RESET
  - LL_UCPD_ORDERED_SET_CABLE_RESET
  - LL_UCPD_ORDERED_SET_SOP1_DEBUG
  - LL_UCPD_ORDERED_SET_SOP2_DEBUG

Return values

- **None:**

Reference Manual to LL API cross reference:

- TX_ORDSET TXORDSET LL_UCPD_WriteTxOrderSet

LL_UCPD_WriteTxPaySize

Function name

__STATIC_INLINE void LL_UCPD_WriteTxPaySize (UCPD_TypeDef * UCPDx, uint32_t TxPaySize)

Function description

write the Tx paysize

Parameters

- **UCPDx:** UCPD Instance
- **TxPaySize:**

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TX_PAYSZ TXPAYSZ LL_UCPD_WriteTxPaySize

LL_UCPD_WriteData

Function name

__STATIC_INLINE void LL_UCPD_WriteData (UCPD_TypeDef * UCPDx, uint8_t Data)

Function description

Write data.

Parameters

- **UCPDx:** UCPD Instance
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TXDR DR LL_UCPD_WriteData

LL_UCPD_ReadRxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

read RX the orderset

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxOrderSet:** one of the following value
  - LL_UCPD_RXORDSET_SOP
  - LL_UCPD_RXORDSET_SOP1
  - LL_UCPD_RXORDSET_SOP2
  - LL_UCPD_RXORDSET_SOP1_DEBUG
  - LL_UCPD_RXORDSET_SOP2_DEBUG
  - LL_UCPD_RXORDSET_CABLE_RESET
  - LL_UCPD_RXORDSET_SOPEXT1
  - LL_UCPD_RXORDSET_SOPEXT2

Reference Manual to LL API cross reference:

- RX_ORDSET RXORDSET LL_UCPD_ReadRxOrderSet

LL_UCPD_ReadRxPaySize

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxPaySize (UCPD_TypeDef const *const UCPDx)

Function description

Read the Rx paysize.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RXPaysize.:**

Reference Manual to LL API cross reference:

- RX_PAYSZ RXPAYSZ LL_UCPD_ReadRxPaySize

LL_UCPD_ReadData

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadData (UCPD_TypeDef const *const UCPDx)

Function description

Read data.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_UCPD_ReadData

LL_UCPD_SetRxOrdExt1

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt1 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext1.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT1 RXSOPX1 LL_UCPD_SetRxOrdExt1

LL_UCPD_SetRxOrdExt2

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt2 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext2.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT2 RXSOPX2 LL_UCPD_SetRxOrdExt2

LL_UCPD_DeInit

Function name

ErrorStatus LL_UCPD_DeInit (UCPD_TypeDef * UCPDx)

Function description

De-initialize the UCPD registers to their default reset values.

Parameters

- **UCPDx:** ucpd Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ucpd registers are de-initialized
  - ERROR: ucpd registers are not de-initialized

LL_UCPD_Init

Function name

ErrorStatus LL_UCPD_Init (UCPD_TypeDef * UCPDx, LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Initialize the ucpd registers according to the specified parameters in UCPD_InitStruct.

Parameters

- **UCPDx:** UCPD Instance
- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure that contains the configuration information for the UCPD peripheral.

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in ucpd configuration registers can only be written when the ucpd is disabled (ucpd_CR1_SPE bit =0), UCPD peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_UCPD_StructInit

Function name

void LL_UCPD_StructInit (LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Set each LL_UCPD_InitTypeDef field to default value.

Parameters

- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

115.3 UCPD Firmware driver defines

The following section lists the various define and macros of the module.

115.3.1 UCPD

UCPD

CC pin enable

LL_UCPD_CCENABLE_NONE

Neither PHY is activated (e.g. disabled state of source)

LL_UCPD_CCENABLE_CC1

Controls apply to only CC1

LL_UCPD_CCENABLE_CC2

Controls apply to only CC1

LL_UCPD_CCENABLE_CC1CC2

Controls apply to both CC1 and CC2 (normal usage for sink/source)

CC pin selection

LL_UCPD_CCPIN_CC1

Use CC1 IO for power delivery communication

LL_UCPD_CCPIN_CC2

Use CC2 IO for power delivery communication

CCx event

LL_UCPD_SNK_CC1_VOPEN

CC1 Sink Open state

LL_UCPD_SNK_CC1_VRP

CC1 Sink vRP default state

LL_UCPD_SNK_CC1_VRP15A

CC1 Sink vRP 1.5A state

LL_UCPD_SNK_CC1_VRP30A

CC1 Sink vRP 3.0A state

LL_UCPD_SNK_CC2_VOPEN

CC2 Sink Open state

LL_UCPD_SNK_CC2_VRP

CC2 Sink vRP default state

LL_UCPD_SNK_CC2_VRP15A

CC2 Sink vRP 1.5A state

LL_UCPD_SNK_CC2_VRP30A

CC2 Sink vRP 3.0A state

LL_UCPD_SRC_CC1_VRA

CC1 Source vRA state

LL_UCPD_SRC_CC1_VRD

CC1 Source vRD state

LL_UCPD_SRC_CC1_OPEN

CC1 Source Open state

LL_UCPD_SRC_CC2_VRA

CC2 Source vRA state

LL_UCPD_SRC_CC2_VRD

CC2 Source vRD state

LL_UCPD_SRC_CC2_OPEN

CC2 Source Open state

ordered set configuration

LL_UCPD_ORDERSET_SOP

SOP Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1

SOP' Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2

SOP'' Ordered set detection enabled

LL_UCPD_ORDERSET_HARDRST

Hard Reset Ordered set detection enabled

LL_UCPD_ORDERSET_CABLERST

Cable Reset Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1_DEBUG

SOP' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2_DEBUG

SOP'' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT1

SOP extension#1 Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT2

SOP extension#2 Ordered set detection enabled

Get Flags Defines

LL_UCPD_SR_TXIS

Transmit interrupt status

LL_UCPD_SR_TXMSGDISC

Transmit message discarded interrupt

LL_UCPD_SR_TXMSGSENT

Transmit message sent interrupt

LL_UCPD_SR_TXMSGABT

Transmit message abort interrupt

LL_UCPD_SR_HRSTDISC

HRST discarded interrupt

LL_UCPD_SR_HRSTSENT

HRST sent interrupt

LL_UCPD_SR_TXUND

Tx data underrun condition interrupt

LL_UCPD_SR_RXNE

Receive data register not empty interrupt

LL_UCPD_SR_RXORDDET

Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_SR_RXHRSTDET

Rx Hard Reset detect interrupt

LL_UCPD_SR_RXOVR

Rx data overflow interrupt

LL_UCPD_SR_RXMSGEND

Rx message received

LL_UCPD_SR_RXERR

Rx error

LL_UCPD_SR_TYPECEVT1

Type C voltage level event on CC1

LL_UCPD_SR_TYPECEVT2

Type C voltage level event on CC2

LL_UCPD_SR_TYPEC_VSTATE_CC1

Status of DC level on CC1 pin

LL_UCPD_SR_TYPEC_VSTATE_CC2

Status of DC level on CC2 pin

LL_UCPD_SR_FRSEVT

Fast Role Swap detection event

IT Defines

LL_UCPD_IMR_TXIS

Enable transmit interrupt status

LL_UCPD_IMR_TXMSGDISC

Enable transmit message discarded interrupt

LL_UCPD_IMR_TXMSGSENT

Enable transmit message sent interrupt

LL_UCPD_IMR_TXMSGABT

Enable transmit message abort interrupt

LL_UCPD_IMR_HRSTDISC

Enable HRST discarded interrupt

LL_UCPD_IMR_HRSTSENT

Enable HRST sent interrupt

LL_UCPD_IMR_TXUND

Enable tx data underrun condition interrupt

LL_UCPD_IMR_RXNE

Enable Receive data register not empty interrupt

LL_UCPD_IMR_RXORDDET

Enable Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_IMR_RXHRSTDET

Enable Rx Hard Reset detect interrupt

LL_UCPD_IMR_RXOVR

Enable Rx data overflow interrupt

LL_UCPD_IMR_RXMSGEND

Enable Rx message received

LL_UCPD_IMR_TYPECEVT1

Enable Type C voltage level event on CC1

LL_UCPD_IMR_TYPECEVT2

Enable Type C voltage level event on CC2

LL_UCPD_IMR_FRSEVT

Enable fast Role Swap detection event

Role Mode

LL_UCPD_ROLE_SNK

Mode SNK Rd

LL_UCPD_ROLE_SRC

Mode SRC Rp

Ordered sets value

LL_UCPD_SYNC1

K-code for Startsynch #1

LL_UCPD_SYNC2

K-code for Startsynch #2

LL_UCPD_SYNC3

K-code for Startsynch #3

LL_UCPD_RST1

K-code for Hard Reset #1

LL_UCPD_RST2

K-code for Hard Reset #2

LL_UCPD_EOP

K-code for EOP End of Packet

LL_UCPD_ORDERED_SET_SOP

SOP Ordered set coding

LL_UCPD_ORDERED_SET_SOP1

SOP' Ordered set coding

LL_UCPD_ORDERED_SET_SOP2

SOP'' Ordered set coding

LL_UCPD_ORDERED_SET_HARD_RESET

Hard Reset Ordered set coding

LL_UCPD_ORDERED_SET_CABLE_RESET

Cable Reset Ordered set coding

LL_UCPD_ORDERED_SET_SOP1_DEBUG

SOP' Debug Ordered set coding

LL_UCPD_ORDERED_SET_SOP2_DEBUG

SOP'' Debug Ordered set coding

prescaler for UCPDCLK

LL_UCPD_PSC_DIV1

Bypass pre-scaling / divide by 1

LL_UCPD_PSC_DIV2

Pre-scale clock by dividing by 2

LL_UCPD_PSC_DIV4

Pre-scale clock by dividing by 4

LL_UCPD_PSC_DIV8

Pre-scale clock by dividing by 8

LL_UCPD_PSC_DIV16

Pre-scale clock by dividing by 16

Resistor value

LL_UCPD_RESISTOR_DEFAULT

Rp default

LL_UCPD_RESISTOR_1_5A

Rp 1.5 A

LL_UCPD_RESISTOR_3_0A

Rp 3.0 A

LL_UCPD_RESISTOR_NONE

No resistor

Receiver mode

LL_UCPD_RXMODE_NORMAL

Normal receive mode

LL_UCPD_RXMODE_BIST_TEST_DATA

BIST receive mode (BIST Test Data Mode)

Rx ordered set code detected

LL_UCPD_RXORDSET_SOP

SOP code detected in receiver

LL_UCPD_RXORDSET_SOP1

SOP' code detected in receiver

LL_UCPD_RXORDSET_SOP2

SOP'' code detected in receiver

LL_UCPD_RXORDSET_SOP1_DEBUG

SOP' Debug code detected in receiver

LL_UCPD_RXORDSET_SOP2_DEBUG

SOP'' Debug code detected in receiver

LL_UCPD_RXORDSET_CABLE_RESET

Cable Reset code detected in receiver

LL_UCPD_RXORDSET_SOPEXT1

SOP extension#1 code detected in receiver

LL_UCPD_RXORDSET_SOPEXT2

SOP extension#2 code detected in receiver

Type of Tx packet

LL_UCPD_TXMODE_NORMAL

Initiate the transfer of a Tx message

LL_UCPD_TXMODE_CABLE_RESET

Trigger a the transfer of a Cable Reset sequence

LL_UCPD_TXMODE_BIST_CARRIER2

Trigger a BIST test sequence send (BIST Carrier Mode 2)

Common Write and read registers Macros

LL_UCPD_WriteReg

Description:

- Write a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_UCPD_ReadReg

Description:

- Read a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be read

Return value:

- Register: value

116 LL USART Generic Driver

116.1 USART Firmware driver registers structures

116.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32h5xx_ll_usart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***
  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***
  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

116.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32h5xx_ll_usart.h

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***

- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***
  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

116.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

116.2.1 Detailed description of functions

LL_USART_Enable

Function name

__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)

Function description

USART Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Enable

LL_USART_Disable

Function name

__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)

Function description

USART Disable (all USART prescalers and outputs are disabled)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name

__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name

__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_DisableFIFO

LL_USART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (const USART_TypeDef * USARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **USARTx:** USART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name

__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)

Function description

USART enabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name

__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)

Function description

USART disabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is disabled, USART is not able to wake up the MCU from Stop mode
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name

__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_SetTransferDirection
- CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (const USART_TypeDef * USARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_GetTransferDirection
- CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name

__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled).

Parameters

- **USARTx:** USART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_SetParity
- CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name

__STATIC_INLINE uint32_t LL_USART_GetParity (const USART_TypeDef * USARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_GetParity
- CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance
- **Method:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (const USART_TypeDef * USARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name

__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)

Function description

Set Word length (i.e.

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_SetDataWidth
- CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_USART_GetDataWidth (const USART_TypeDef * USARTx)

Function description

Return Word length (i.e.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_GetDataWidth
- CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name

__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name

__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)

Function description

Prevent Mute Mode use.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode (const USART_TypeDef * USARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name

__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)

Function description

Set Oversampling to 8-bit or 16-bit mode.

Parameters

- **USARTx:** USART Instance
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name

__STATIC_INLINE uint32_t LL_USART_GetOverSampling (const USART_TypeDef * USARTx)

Function description

Return Oversampling mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name

__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)

Function description

Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters

- **USARTx:** USART Instance
- **LastBitClockPulse:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name

__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (const USART_TypeDef * USARTx)

Function description

Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (const USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (const USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name

__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetPrescaler (const USART_TypeDef * USARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description

Enable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)

Function description

Disable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (const USART_TypeDef * USARTx)

Function description

Indicate if Clock output on SCLK pin is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **USARTx:** USART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (const USART_TypeDef * USARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name

__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_ConfigCharacter
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M0 LL_USART_ConfigCharacter
- CR1 M1 LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **USARTx:** USART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (const USART_TypeDef * USARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **USARTx:** USART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (const USART_TypeDef * USARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (const USART_TypeDef * USARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Enable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Disable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (const USART_TypeDef * USARTx)

Function description

Indicate if Auto Baud-Rate Detection mechanism is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name

__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)

Function description

Set Auto Baud-Rate mode bits.

Parameters

- **USARTx:** USART Instance
- **AutoBaudRateMode:** This parameter can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (const USART_TypeDef * USARTx)

Function description

Return Auto Baud-Rate mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name

__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name

__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (const USART_TypeDef * USARTx)

Function description

Indicate if Receiver Timeout feature is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the USART node.

Parameters

- **USARTx:** USART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the USART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_ConfigNodeAddress
- CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (const USART_TypeDef * USARTx)

Function description

Return 8 bit Address of the USART node as set in ADD field of CR2.

Parameters

- **USARTx:** USART Instance

Return values

- **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (const USART_TypeDef * USARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (const USART_TypeDef * USARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)

Function description

Enable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)

Function description

Disable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (const USART_TypeDef * USARTx)

Function description

Indicate if One bit sampling method is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)

Function description

Enable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)

Function description

Disable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (const USART_TypeDef * USARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name

__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance
- **Type:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_USART_GetWKUPType (const USART_TypeDef * USARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_GetWKUPType

LL_USART_SetBaudRate

Function name

__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)

Function description

Configure USART BRR register for achieving expected Baud Rate value.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values
- Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_USART_GetBaudRate (const USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)

Function description

Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name

__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)

Function description

Set Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance
- **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (const USART_TypeDef * USARTx)

Function description

Get Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name

__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)

Function description

Set Block Length value in reception.

Parameters

- **USARTx:** USART Instance
- **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetBlockLength (const USART_TypeDef * USARTx)

Function description

Get Block Length value in reception.

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name

__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)

Function description

Enable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name

__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)

Function description

Disable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (const USART_TypeDef * USARTx)

Function description

Indicate if IrDA mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name

__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)

Function description

Configure IrDA Power Mode (Normal or Low Power)

Parameters

- **USARTx:** USART Instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_IRDA_POWER_LOW

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (const USART_TypeDef * USARTx)

Function description

Retrieve IrDA Power Mode configuration (Normal or Low Power)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name

__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (const USART_TypeDef * USARTx)

Function description

Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Enable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Disable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard NACK transmission is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name

__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)

Function description

Enable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name

__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)

Function description

Disable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name

__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)

Function description

Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance
- **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (const USART_TypeDef * USARTx)

Function description

Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name

__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (const USART_TypeDef * USARTx)

Function description

Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name

__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)

Function description

Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance
- **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (const USART_TypeDef * USARTx)

Function description

Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (const USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name

__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)

Function description

Enable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name

__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)

Function description

Disable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Synchronous Slave mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Enable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Disable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave will be always selected and NSS input pin will be ignored.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Slave Selection depends on NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name

__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)

Function description

Set LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance
- **LINBDLength:** This parameter can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (const USART_TypeDef * USARTx)

Function description

Return LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name

__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)

Function description

Enable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name

__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)

Function description

Disable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (const USART_TypeDef * USARTx)

Function description

Indicate if LIN mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (const USART_TypeDef * USARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (const USART_TypeDef * USARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name

__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name

__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (const USART_TypeDef * USARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (const USART_TypeDef * USARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (const USART_TypeDef * USARTx)

Function description

Check if the USART Framing Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (const USART_TypeDef * USARTx)

Function description

Check if the USART Noise error detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (const USART_TypeDef * USARTx)

Function description

Check if the USART OverRun Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE line detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS interrupt Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Time Out Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (const USART_TypeDef * USARTx)

Function description

Check if the SPI Slave Underrun error flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (const USART_TypeDef * USARTx)

Function description

Check if the USART Busy Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (const USART_TypeDef * USARTx)

Function description

Check if the USART Send Break Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from stop mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)

Function description

Clear Parity Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)

Function description

Clear Framing Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)

Function description

Clear Noise Error detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Clear Smartcard Transmission Complete Before Guard Time Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name

__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)

Function description

Clear LIN Break Detection Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name

__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)

Function description

Clear Receiver Time Out Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name

__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)

Function description

Clear End Of Block Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)

Function description

Clear SPI Slave Underrun Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)

Function description

Clear Character Match Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_USART_ClearFlag_CM

LL_USART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name

__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name

__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name

__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name

__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name

__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)

Function description

Enable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name

__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)

Function description

Enable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)

Function description

Enable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)

Function description

Enable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Enable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_EnableIT_TCBGT

LL_USART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name

__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name

__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name

__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name

__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name

__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)

Function description

Disable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name

__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)

Function description

Disable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)

Function description

Disable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)

Function description

Disable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_DisableIT_WKUP

LL_USART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Disable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE Interrupt source is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Timeout Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (const USART_TypeDef * USARTx)

Function description

Check if the USART Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (const USART_TypeDef * USARTx)

Function description

Check if USART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (const USART_TypeDef * USARTx)

Function description

Check if USART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * USARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (const USART_TypeDef * USARTx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **USARTx:** USART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_USART_DMA_REG_DATA_TRANSMIT
  - LL_USART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_DMA_GetRegAddr
-
- TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name

__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name

__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)

Function description

Request an Automatic Baud Rate measurement on next received data frame.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name

__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)

Function description

Request Break sending.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)

Function description

Put USART in mute mode and set the RWU flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Transmit data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name

ErrorStatus LL_USART_DeInit (const USART_TypeDef * USARTx)

Function description

De-initialize USART registers (Registers restored to their default values).

Parameters

- **USARTx:** USART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are de-initialized
  - ERROR: USART registers are not de-initialized

LL_USART_Init

Function name

ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, const LL_USART_InitTypeDef * USART_InitStruct)

Function description

Initialize USART registers according to the specified parameters in USART_InitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are initialized according to USART_InitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name

void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)

Function description

Set each LL_USART_InitTypeDef field to default value.

Parameters

- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_USART_ClockInit

Function name

ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, const LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name

void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters

- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

116.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

116.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT

Measurement of the start bit is used to detect the baud rate

LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE

Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx

LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME

0x7F frame detection

LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF

Parity error clear flag

LL_USART_ICR_FECF

Framing error clear flag

LL_USART_ICR_NECF

Noise error detected clear flag

LL_USART_ICR_ORECF

Overrun error clear flag

LL_USART_ICR_IDLECF

Idle line detected clear flag

LL_USART_ICR_TXFECF

TX FIFO Empty clear flag

LL_USART_ICR_TCCF

Transmission complete clear flag

LL_USART_ICR_TCBGTCF

Transmission completed before guard time clear flag

LL_USART_ICR_LBDCF

LIN break detection clear flag

LL_USART_ICR_CTSCF

CTS clear flag

LL_USART_ICR_RTOCF

Receiver timeout clear flag

LL_USART_ICR_EOBCF

End of block clear flag

LL_USART_ICR_UDRCF

SPI Slave Underrun clear flag

LL_USART_ICR_CMCF

Character match clear flag

LL_USART_ICR_WUCF

Wakeup from Stop mode clear flag

Clock Signal

LL_USART_CLOCK_DISABLE

Clock signal not provided

LL_USART_CLOCK_ENABLE

Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_USART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH

DE signal is active high

LL_USART_DE_POLARITY_LOW

DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_USART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_USART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_USART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_USART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_USART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_USART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_USART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_USART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_USART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE

Parity error flag

LL_USART_ISR_FE

Framing error flag

LL_USART_ISR_NE

Noise detected flag

LL_USART_ISR_ORE

Overrun error flag

LL_USART_ISR_IDLE

Idle line detected flag

LL_USART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_USART_ISR_TC

Transmission complete flag

LL_USART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_USART_ISR_LBDF

LIN break detection flag

LL_USART_ISR_CTSIF

CTS interrupt flag

LL_USART_ISR_CTS

CTS flag

LL_USART_ISR_RTOF

Receiver timeout flag

LL_USART_ISR_EOBF

End of block flag

LL_USART_ISR_UDR

SPI Slave underrun error flag

LL_USART_ISR_ABRE

Auto baud rate error flag

LL_USART_ISR_ABRF

Auto baud rate flag

LL_USART_ISR_BUSY

Busy flag

LL_USART_ISR_CMF

Character match flag

LL_USART_ISR_SBKF

Send break flag

LL_USART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_USART_ISR_WUF

Wakeup from Stop mode flag

LL_USART_ISR_TEACK

Transmit enable acknowledge flag

LL_USART_ISR_REACK

Receive enable acknowledge flag

LL_USART_ISR_TXFE

TX FIFO empty flag

LL_USART_ISR_RXFF

RX FIFO full flag

LL_USART_ISR_TCBGT

Transmission complete before guard time completion flag

LL_USART_ISR_RXFT

RX FIFO threshold flag

LL_USART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_USART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_USART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_USART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL

IrDA normal power mode

LL_USART_IRDA_POWER_LOW

IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE

IDLE interrupt enable

LL_USART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_USART_CR1_TCIE

Transmission complete interrupt enable

LL_USART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_USART_CR1_PEIE

Parity error

LL_USART_CR1_CMIE

Character match interrupt enable

LL_USART_CR1_RTOIE

Receiver timeout interrupt enable

LL_USART_CR1_EOBIE

End of Block interrupt enable

LL_USART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_USART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_USART_CR2_LBDIE

LIN break detection interrupt enable

LL_USART_CR3_EIE

Error interrupt enable

LL_USART_CR3_CTSIE

CTS interrupt enable

LL_USART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_USART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_USART_CR3_TCBGTIE

Transmission complete before guard time interrupt enable

LL_USART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT

The clock pulse of the last data bit is not output to the SCLK pin

LL_USART_LASTCLKPULSE_OUTPUT

The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B

10-bit break detection method selected

LL_USART_LINBREAK_DETECT_11B

11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16

Oversampling by 16

LL_USART_OVERSAMPLING_8

Oversampling by 8

Parity Control

LL_USART_PARITY_NONE

Parity control disabled

LL_USART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_USART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE

The first clock transition is the first data capture edge

LL_USART_PHASE_2EDGE

The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW

Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HIGH

Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1

Input clock not divided

LL_USART_PRESCALER_DIV2

Input clock divided by 2

LL_USART_PRESCALER_DIV4

Input clock divided by 4

LL_USART_PRESCALER_DIV6

Input clock divided by 6

LL_USART_PRESCALER_DIV8

Input clock divided by 8

LL_USART_PRESCALER_DIV10

Input clock divided by 10

LL_USART_PRESCALER_DIV12

Input clock divided by 12

LL_USART_PRESCALER_DIV16

Input clock divided by 16

LL_USART_PRESCALER_DIV32

Input clock divided by 32

LL_USART_PRESCALER_DIV64

Input clock divided by 64

LL_USART_PRESCALER_DIV128

Input clock divided by 128

LL_USART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_USART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5

0.5 stop bit

LL_USART_STOPBITS_1

1 stop bit

LL_USART_STOPBITS_1_5

1.5 stop bits

LL_USART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_USART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE

USART wake up from Mute mode on Idle Line

LL_USART_WAKEUP_ADDRESSMARK

USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_USART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_USART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

117 LL UTILS Generic Driver

117.1 UTILS Firmware driver registers structures

117.1.1 LL_UTILS_PLLInitTypeDef

LL_UTILS_PLLInitTypeDef is defined in the stm32h5xx_ll_utils.h

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t FRACN***
- ***uint32_t VCO_Input***
- ***uint32_t VCO_Output***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***
  Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetM()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***
  Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetN()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLP***
  Division for the main system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowedThis feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetP()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::FRACN***
  Fractional part of the multiplication factor for PLL VCO. This parameter can be a value between 0 and 8191This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetFRACN()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::VCO_Input***
  PLL clock Input range. This parameter can be a value of ***RCC_LL_EC_PLLINPUTRANGE***This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetVCOInputRange()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::VCO_Output***
  PLL clock Output range. This parameter can be a value of ***RCC_LL_EC_PLLOUTPUTRANGE***This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetVCOOutputRange()**.

117.1.2 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32h5xx_ll_utils.h

Data Fields

- ***uint32_t SYSCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB3CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::SYSCLKDivider***
  The System clock (SYSCLK) divider. This clock is derived from the System clock. This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.

- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB3CLKDivider***
  The APB3 clock (PCLK3) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB3_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB3Prescaler()**.

117.2 UTILS Firmware driver API description

The following section lists the various functions of the UTILS library.

117.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK is 250 MHz and HCLK is 250 MHz.
- The maximum frequency of the PCLK1, PCLK2 and PCLK3 is 250 MHz.

This section contains the following APIs:

- ***LL_SetSystemCoreClock()***
- ***LL_PLL_ConfigSystemClock_CSI()***
- ***LL_PLL_ConfigSystemClock_HSI()***
- ***LL_PLL_ConfigSystemClock_HSE()***
- ***LL_SetFlashLatency()***

117.2.2 Detailed description of functions

LL_GetUID_Word0

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)

Function description

Get Word0 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[31:0]:** X and Y coordinates on the wafer expressed in BCD format

LL_GetUID_Word1

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)

Function description

Get Word1 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[63:32]:** Wafer number (UID[39:32]) & LOT_NUM[23:0] (UID[63:40])

LL_GetUID_Word2

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)

Function description

Get Word2 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[95:64]:** Lot number (ASCII encoded) - LOT_NUM[55:24]

LL_GetFlashSize

Function name

__STATIC_INLINE uint32_t LL_GetFlashSize (void)

Function description

Get Flash memory size.

Return values

- **FLASH_SIZE[15:0]:** Flash memory size

Notes

- This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_GetPackageType

Function name

__STATIC_INLINE uint32_t LL_GetPackageType (void)

Function description

Get Package type.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_PACKAGETYPE_LQFP64
  - LL_UTILS_PACKAGETYPE_VFQFPN68
  - LL_UTILS_PACKAGETYPE_LQFP100
  - LL_UTILS_PACKAGETYPE_UFBGA176
  - LL_UTILS_PACKAGETYPE_LQFP144
  - LL_UTILS_PACKAGETYPE_LQFP48
  - LL_UTILS_PACKAGETYPE_UFBGA169
  - LL_UTILS_PACKAGETYPE_LQFP176
  - LL_UTILS_PACKAGETYPE_UFQFPN32
  - LL_UTILS_PACKAGETYPE_LQFP100_SMPS
  - LL_UTILS_PACKAGETYPE_UFBGA176_SMPS
  - LL_UTILS_PACKAGETYPE_LQFP144_SMPS
  - LL_UTILS_PACKAGETYPE_LQFP176_SMPS
  - LL_UTILS_PACKAGETYPE_UFBGA169_SMPS
  - LL_UTILS_PACKAGETYPE_WLCSP25
  - LL_UTILS_PACKAGETYPE_UFQFPN48

LL_InitTick

Function name

__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)

Function description

This function configures the Cortex-M SysTick source of the time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name

void LL_Init1msTick (uint32_t HCLKFrequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.
- HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides accurate delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.
- To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name

void LL_SetSystemCoreClock (uint32_t HCLKFrequency)

Function description

This function sets directly SystemCoreClock CMSIS variable.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)

Return values

- **None:**

Notes

- Variable can be calculated also through SystemCoreClockUpdate function.

LL_PLL_ConfigSystemClock_CSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_CSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock at maximum frequency with CSI as clock source of the PLL1.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that all PLLs is disabled.
- Function is based on the following formula: PLL1 output frequency = (((CSI frequency / PLL1M) * PLL1N) / PLL1P)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = CSI frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 192 and 836 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1P: ensure that max frequency at 250 MHz is reached (PLL1VCO_output / PLL1P)

LL_PLL_ConfigSystemClock_HSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock at maximum frequency with HSI as clock source of the PLL1.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL1.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that all PLLs are disabled.
- Function is based on the following formula: PLL1 output frequency = (((HSI frequency / PLL1M) * PLL1N) / PLL1P)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = HSI frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 150 and 836 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1P: ensure that max frequency at 250 MHz is reach (PLL1VCO_output / PLL1P)

LL_PLL_ConfigSystemClock_HSE

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock with HSE as clock source of the PLL1.

Parameters

- **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 50000000
- **HSEBypass:** This parameter can be one of the following values:
  - LL_UTILS_HSEBYPASS_ON
  - LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that the PLL1 is disabled.
- Function is based on the following formula: PLL1 output frequency = (((HSE frequency / PLL1M) * PLL1N) / PLL1P)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = HSE frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 192 and 836 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1P: ensure that max frequency at 250 MHz is reached (PLL1VCO_output / PLL1P)

LL_SetFlashLatency

Function name

ErrorStatus LL_SetFlashLatency (uint32_t HCLK_Frequency)

Function description

Update number of Flash wait states in line with new frequency and current voltage range.

Parameters

- **HCLK_Frequency:** HCLK frequency

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Latency has been modified
  - ERROR: Latency cannot be modified

117.3 UTILS Firmware driver defines

The following section lists the various define and macros of the module.

117.3.1 UTILS

UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass Analog is enabled

LL_UTILS_HSEBYPASS_DIGITAL_ON

HSE Bypass Digital is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_LQFP64

LQFP64 package type

LL_UTILS_PACKAGETYPE_VFQFPN68

VFQFPN68 package type

LL_UTILS_PACKAGETYPE_LQFP100

LQFP100 package type

LL_UTILS_PACKAGETYPE_UFBGA176

UFBGA176+25 package type

LL_UTILS_PACKAGETYPE_LQFP144

LQFP144 package type

LL_UTILS_PACKAGETYPE_LQFP48

LQFP48 package type

LL_UTILS_PACKAGETYPE_UFBGA169

UFBGA169 package type

LL_UTILS_PACKAGETYPE_LQFP176

LQFP176 package type

LL_UTILS_PACKAGETYPE_UFQFPN32

UFQFPN32 package type

LL_UTILS_PACKAGETYPE_LQFP100_SMPS

LQFP100 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA176_SMPS

UFBGA176+25 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP144_SMPS

LQFP144 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP176_SMPS

LQFP176 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA169_SMPS

UFBGA169 with internal SMPS package type

LL_UTILS_PACKAGETYPE_WLCSP25

WLCSP25 package type

LL_UTILS_PACKAGETYPE_UFQFPN48

UFQFPN48 package type

118 LL WWDG Generic Driver

118.1 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

118.1.1 Detailed description of functions

LL_WWDG_Enable

Function name

__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)

Function description

Enable Window Watchdog.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (const WWDG_TypeDef * WWDGx)

Function description

Checks if Window Watchdog is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name

__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)

Function description

Set the Watchdog counter value to provided value (7-bits T[6:0])

Parameters

- **WWDGx:** WWDG Instance
- **Counter:** 0..0x7F (7 bit counter value)

Return values

- **None:**

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetCounter (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Counter Value (7 bits counter value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)

Function description

Set the time base of the prescaler (WDGTB).

Parameters

- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Return values

- **None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

118.2 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

118.2.1 WWDG

WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

LL_WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

LL_WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

LL_WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

LL_WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

Common Write and read registers macros

LL_WWDG_WriteReg

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

119 Correspondence between API registers and API low-layer driver functions

This appendix contains the correspondence table between the register or bit name, as mentioned inside the reference manual of the component, and the name of the LL functions to modify or read this register or bit.

119.1 ADC

Table 25. Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
AWD2CR	AWD2CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
AWD3CR	AWD3CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
CALFACT	CALFACT_D	LL_ADC_GetCalibrationFactor
		LL_ADC_SetCalibrationFactor
	CALFACT_S	LL_ADC_GetCalibrationFactor
		LL_ADC_SetCalibrationFactor
CCR	CKMODE	LL_ADC_GetCommonClock
		LL_ADC_SetCommonClock
	DELAY	LL_ADC_GetMultiTwoSamplingDelay
		LL_ADC_SetMultiTwoSamplingDelay
	DMACFG	LL_ADC_GetMultiDMATransfer
		LL_ADC_SetMultiDMATransfer
	DUAL	LL_ADC_GetMultimode
		LL_ADC_SetMultimode
	MDMA	LL_ADC_GetMultiDMATransfer
		LL_ADC_SetMultiDMATransfer
	PRESC	LL_ADC_GetCommonClock
		LL_ADC_SetCommonClock
	TSEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
	VBATEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
	VREFEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
CDR	RDATA_MST	LL_ADC_DMA_GetRegAddr

Register	Field	Function
CDR	RDATA_MST	LL_ADC_REG_ReadMultiConversionData32
	RDATA_SLV	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadMultiConversionData32
CFGR	ALIGN	LL_ADC_GetDataAlignment
		LL_ADC_SetDataAlignment
	AUTDLY	LL_ADC_GetLowPowerMode
		LL_ADC_SetLowPowerMode
	AWD1CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWD1EN	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWD1SGL	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	CONT	LL_ADC_REG_GetContinuousMode
		LL_ADC_REG_SetContinuousMode
	DISCEN	LL_ADC_REG_GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DISCNUM	LL_ADC_REG_GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DMACFG	LL_ADC_REG_GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	DMAEN	LL_ADC_REG_GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	EXTEN	LL_ADC_REG_GetTriggerEdge
		LL_ADC_REG_GetTriggerSource
		LL_ADC_REG_IsTriggerSourceSWStart
		LL_ADC_REG_SetTriggerEdge
		LL_ADC_REG_SetTriggerSource
	EXTSEL	LL_ADC_REG_GetTriggerSource
		LL_ADC_REG_SetTriggerSource
	JAUTO	LL_ADC_INJ_GetTrigAuto
		LL_ADC_INJ_SetTrigAuto
	JAWD1EN	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	JDISCEN	LL_ADC_INJ_GetSequencerDiscont
		LL_ADC_INJ_SetSequencerDiscont
	JQDIS	LL_ADC_INJ_GetQueueMode
		LL_ADC_INJ_SetQueueMode
	JQM	LL_ADC_INJ_GetQueueMode
		LL_ADC_INJ_SetQueueMode
	OVRMOD	LL_ADC_REG_GetOverrun

Register	Field	Function
CFGR	OVRMOD	LL_ADC_REG_SetOverrun
	RES	LL_ADC_GetResolution
		LL_ADC_SetResolution
CFGR2	BULB	LL_ADC_REG_GetSamplingMode
		LL_ADC_REG_SetSamplingMode
	JOVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	OVSR	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingRatio
	OVSS	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingShift
	ROVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	ROVSM	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	SMPTRIG	LL_ADC_REG_GetSamplingMode
		LL_ADC_REG_SetSamplingMode
	SWTRIG	LL_ADC_REG_StartSamplingPhase
		LL_ADC_REG_StopSamplingPhase
	TROVS	LL_ADC_GetOverSamplingDiscont
		LL_ADC_SetOverSamplingDiscont
CR	ADCAL	LL_ADC_IsCalibrationOnGoing
		LL_ADC_StartCalibration
	ADCALDIF	LL_ADC_StartCalibration
	ADDIS	LL_ADC_Disable
		LL_ADC_IsDisableOngoing
	ADEN	LL_ADC_Enable
		LL_ADC_IsEnabled
	ADSTART	LL_ADC_REG_IsConversionOngoing
		LL_ADC_REG_StartConversion
	ADSTP	LL_ADC_REG_IsStopConversionOngoing
		LL_ADC_REG_StopConversion
	ADVREGEN	LL_ADC_DisableInternalRegulator
		LL_ADC_EnableInternalRegulator
		LL_ADC_IsInternalRegulatorEnabled
	DEEPPWD	LL_ADC_DisableDeepPowerDown
		LL_ADC_EnableDeepPowerDown
		LL_ADC_IsDeepPowerDownEnabled
	JADSTART	LL_ADC_INJ_IsConversionOngoing
		LL_ADC_INJ_StartConversion
	JADSTP	LL_ADC_INJ_IsStopConversionOngoing

Register	Field	Function
CR	JADSTP	LL_ADC_INJ_StopConversion
CSR	ADRDY_MST	LL_ADC_IsActiveFlag_MST_ADRDY
	ADRDY_SLV	LL_ADC_IsActiveFlag_SLV_ADRDY
	AWD1_MST	LL_ADC_IsActiveFlag_MST_AWD1
	AWD1_SLV	LL_ADC_IsActiveFlag_SLV_AWD1
	AWD2_MST	LL_ADC_IsActiveFlag_MST_AWD2
	AWD2_SLV	LL_ADC_IsActiveFlag_SLV_AWD2
	AWD3_MST	LL_ADC_IsActiveFlag_MST_AWD3
	AWD3_SLV	LL_ADC_IsActiveFlag_SLV_AWD3
	EOC_MST	LL_ADC_IsActiveFlag_MST_EOC
	EOC_SLV	LL_ADC_IsActiveFlag_SLV_EOC
	EOSMP_MST	LL_ADC_IsActiveFlag_MST_EOSMP
	EOSMP_SLV	LL_ADC_IsActiveFlag_SLV_EOSMP
	EOS_MST	LL_ADC_IsActiveFlag_MST_EOS
	EOS_SLV	LL_ADC_IsActiveFlag_SLV_EOS
	JEOC_MST	LL_ADC_IsActiveFlag_MST_JEOC
	JEOC_SLV	LL_ADC_IsActiveFlag_SLV_JEOC
	JEOS_MST	LL_ADC_IsActiveFlag_MST_JEOS
	JEOS_SLV	LL_ADC_IsActiveFlag_SLV_JEOS
	JQOVF_MST	LL_ADC_IsActiveFlag_MST_JQOVF
	JQOVF_SLV	LL_ADC_IsActiveFlag_SLV_JQOVF
	OVR_MST	LL_ADC_IsActiveFlag_MST_OVR
	OVR_SLV	LL_ADC_IsActiveFlag_SLV_OVR
DIFSEL	DIFSEL	LL_ADC_GetChannelSingleDiff
		LL_ADC_SetChannelSingleDiff
DR	RDATA	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadConversionData10
		LL_ADC_REG_ReadConversionData12
		LL_ADC_REG_ReadConversionData32
		LL_ADC_REG_ReadConversionData6
		LL_ADC_REG_ReadConversionData8
IER	ADRDYIE	LL_ADC_DisableIT_ADRDY
		LL_ADC_EnableIT_ADRDY
		LL_ADC_IsEnabledIT_ADRDY
	AWD1IE	LL_ADC_DisableIT_AWD1
		LL_ADC_EnableIT_AWD1
		LL_ADC_IsEnabledIT_AWD1
	AWD2IE	LL_ADC_DisableIT_AWD2
		LL_ADC_EnableIT_AWD2
		LL_ADC_IsEnabledIT_AWD2
	AWD3IE	LL_ADC_DisableIT_AWD3

Register	Field	Function
IER	AWD3IE	LL_ADC_EnableIT_AWD3
		LL_ADC_IsEnabledIT_AWD3
	EOCIE	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	EOSIE	LL_ADC_DisableIT_EOS
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	EOSMPIE	LL_ADC_DisableIT_EOSMP
		LL_ADC_EnableIT_EOSMP
		LL_ADC_IsEnabledIT_EOSMP
	JEOCIE	LL_ADC_DisableIT_JEOC
		LL_ADC_EnableIT_JEOC
		LL_ADC_IsEnabledIT_JEOC
	JEOSIE	LL_ADC_DisableIT_JEOS
		LL_ADC_EnableIT_JEOS
		LL_ADC_IsEnabledIT_JEOS
	JQOVFIE	LL_ADC_DisableIT_JQOVF
		LL_ADC_EnableIT_JQOVF
		LL_ADC_IsEnabledIT_JQOVF
	OVRIE	LL_ADC_DisableIT_OVR
		LL_ADC_EnableIT_OVR
		LL_ADC_IsEnabledIT_OVR
ISR	ADRDY	LL_ADC_ClearFlag_ADRDY
		LL_ADC_IsActiveFlag_ADRDY
	AWD1	LL_ADC_ClearFlag_AWD1
		LL_ADC_IsActiveFlag_AWD1
	AWD2	LL_ADC_ClearFlag_AWD2
		LL_ADC_IsActiveFlag_AWD2
	AWD3	LL_ADC_ClearFlag_AWD3
		LL_ADC_IsActiveFlag_AWD3
	EOC	LL_ADC_ClearFlag_EOC
		LL_ADC_IsActiveFlag_EOC
	EOS	LL_ADC_ClearFlag_EOS
		LL_ADC_IsActiveFlag_EOS
	EOSMP	LL_ADC_ClearFlag_EOSMP
		LL_ADC_IsActiveFlag_EOSMP
	JEOC	LL_ADC_ClearFlag_JEOC
		LL_ADC_IsActiveFlag_JEOC
	JEOS	LL_ADC_ClearFlag_JEOS
		LL_ADC_IsActiveFlag_JEOS

Register	Field	Function
ISR	JQOVF	LL_ADC_ClearFlag_JQOVF
		LL_ADC_IsActiveFlag_JQOVF
	OVR	LL_ADC_ClearFlag_OVR
		LL_ADC_IsActiveFlag_OVR
JDR1	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR2	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR3	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JDR4	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData6
		LL_ADC_INJ_ReadConversionData8
JSQR	JEXTEN	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetTriggerEdge
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_IsTriggerSourceSWStart
		LL_ADC_INJ_SetTriggerEdge
		LL_ADC_INJ_SetTriggerSource
	JEXTSEL	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_SetTriggerSource
	JL	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerLength
		LL_ADC_INJ_SetSequencerLength
	JSQ1	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ2	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks

Register	Field	Function
JSQR	JSQ2	LL_ADC_INJ_SetSequencerRanks
	JSQ3	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ4	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
OFR1	OFFSET1	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET1_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET1_EN	LL_ADC_GetOffsetState
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SATEN	LL_ADC_GetOffsetSaturation
		LL_ADC_SetOffsetSaturation
OFR2	OFFSET2	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET2_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET2_EN	LL_ADC_GetOffsetState
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SATEN	LL_ADC_GetOffsetSaturation
		LL_ADC_SetOffsetSaturation
OFR3	OFFSET3	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET3_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET3_EN	LL_ADC_GetOffsetState
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SATEN	LL_ADC_GetOffsetSaturation
		LL_ADC_SetOffsetSaturation
OFR4	OFFSET4	LL_ADC_GetOffsetLevel

Register	Field	Function
OFR4	OFFSET4	LL_ADC_SetOffset
	OFFSET4_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET4_EN	LL_ADC_GetOffsetState
		LL_ADC_SetOffset
		LL_ADC_SetOffsetState
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SATEN	LL_ADC_GetOffsetSaturation
		LL_ADC_SetOffsetSaturation
OR	OP0	LL_ADC_DisableChannel0_GPIO
		LL_ADC_DisableChannelVDDcore
		LL_ADC_EnableChannel0_GPIO
		LL_ADC_EnableChannelVDDcore
SMPR1	SMP0	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP1	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP2	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP3	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP4	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP5	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP6	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP7	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP8	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP9	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMPPLUS	LL_ADC_GetSamplingTimeCommonConfig
		LL_ADC_SetSamplingTimeCommonConfig
SMPR2	SMP10	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP11	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP12	LL_ADC_GetChannelSamplingTime

Register	Field	Function
SMPR2	SMP12	LL_ADC_SetChannelSamplingTime
	SMP13	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP14	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP15	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP16	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP17	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP18	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
SQR1	L	LL_ADC_REG_GetSequencerLength
		LL_ADC_REG_SetSequencerLength
	SQ1	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ2	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ3	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ4	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR2	SQ5	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ6	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ7	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ8	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ9	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR3	SQ10	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ11	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ12	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ13	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks

Register	Field	Function
SQR3	SQ14	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR4	SQ15	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ16	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
TR1	AWDFILT	LL_ADC_GetAWDFilteringConfiguration
		LL_ADC_SetAWDFilteringConfiguration
	HT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR2	HT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR3	HT3	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT3	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds

119.2 BUS

Table 26. Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
AHB1ENR	BKPRAMEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	CORDICEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	CRCEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DCACHE1EN	LL_AHB1_GRP1_DisableClock

Register	Field	Function
AHB1ENR	DCACHE1EN	LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	ETHEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	ETHRXEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	ETHTXEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FLITFEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FMACEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GPDMA1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GPDMA2EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	RAMCFGEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	SRAM1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	TZSC1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
AHB1LPENR	BKPRAMLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	CORDICLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	CRCLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep

Register	Field	Function
AHB1LPENR	DCACHE1LPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	ETHLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	ETHRXLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	ETHTXLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	FLITFLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	FMACLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	GPDMA1LPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	GPDMA2LPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	RAMCFGLPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	SRAM1LPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
	TZSC1LPEN	LL_AHB1_GRP1_DisableClockSleep
		LL_AHB1_GRP1_EnableClockSleep
		LL_AHB1_GRP1_IsEnabledClockSleep
AHB1RSTR	CORDICRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	CRCRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	ETHRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	FMACRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset

Register	Field	Function
AHB1RSTR	GPDMA1RST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPDMA2RST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	RAMCFGRST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	TZSC1RST	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
AHB2ENR	ADCEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	AESEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	DAC1EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	DCMI_PSSIEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOAEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOBEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOCEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIODEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOEEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOFEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOGEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock

Register	Field	Function
AHB2ENR	GPIOHEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOIEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	HASHEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	PKAEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	RNGEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SAESEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SRAM2EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SRAM3EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
AHB2LPENR	ADCLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	AESLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	DAC1LPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	DCMI_PSSILPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOALPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOBLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep

Register	Field	Function
AHB2LPENR	GPIOBLPEN	LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOCLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIODLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOELPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOFLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOGLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOHLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	GPIOILPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	HASHLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	PKALPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	RNGLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	SAESLPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	SRAM2LPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
	SRAM3LPEN	LL_AHB2_GRP1_DisableClockSleep
		LL_AHB2_GRP1_EnableClockSleep
		LL_AHB2_GRP1_IsEnabledClockSleep
AHB2RST	ADCRST	LL_AHB2_GRP1_ForceReset

Register	Field	Function
AHB2RST	ADCRST	LL_AHB2_GRP1_ReleaseReset
	AESRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	DAC1RST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	DCMI_PSSIRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOARST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOBRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOCRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIODRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOERST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOFRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOGRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOHRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOIRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	HASHRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	PKARST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	RNGRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	SAESRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
AHB4ENR	FMCEN	LL_AHB4_GRP1_DisableClock
		LL_AHB4_GRP1_EnableClock
		LL_AHB4_GRP1_IsEnabledClock
	OCTOSPI1EN	LL_AHB4_GRP1_DisableClock
		LL_AHB4_GRP1_EnableClock
		LL_AHB4_GRP1_IsEnabledClock
	OTFDEC1EN	LL_AHB4_GRP1_DisableClock
		LL_AHB4_GRP1_EnableClock

Register	Field	Function
AHB4ENR	OTFDEC1EN	LL_AHB4_GRP1_IsEnabledClock
	SDMMC1EN	LL_AHB4_GRP1_DisableClock
		LL_AHB4_GRP1_EnableClock
		LL_AHB4_GRP1_IsEnabledClock
	SDMMC2EN	LL_AHB4_GRP1_DisableClock
		LL_AHB4_GRP1_EnableClock
		LL_AHB4_GRP1_IsEnabledClock
AHB4LPENR	FMCLPEN	LL_AHB4_GRP1_DisableClockSleep
		LL_AHB4_GRP1_EnableClockSleep
		LL_AHB4_GRP1_IsEnabledClockSleep
	OCTOSPI1LPEN	LL_AHB4_GRP1_DisableClockSleep
		LL_AHB4_GRP1_EnableClockSleep
		LL_AHB4_GRP1_IsEnabledClockSleep
	OTFDEC1LPEN	LL_AHB4_GRP1_DisableClockSleep
		LL_AHB4_GRP1_EnableClockSleep
		LL_AHB4_GRP1_IsEnabledClockSleep
	SDMMC1LPEN	LL_AHB4_GRP1_DisableClockSleep
		LL_AHB4_GRP1_EnableClockSleep
		LL_AHB4_GRP1_IsEnabledClockSleep
	SDMMC2LPEN	LL_AHB4_GRP1_DisableClockSleep
		LL_AHB4_GRP1_EnableClockSleep
		LL_AHB4_GRP1_IsEnabledClockSleep
AHB4RSTR	FMCRST	LL_AHB4_GRP1_ForceReset
		LL_AHB4_GRP1_ReleaseReset
	OCTOSPI1RST	LL_AHB4_GRP1_ForceReset
		LL_AHB4_GRP1_ReleaseReset
	OTFDEC1RST	LL_AHB4_GRP1_ForceReset
		LL_AHB4_GRP1_ReleaseReset
	SDMMC1RST	LL_AHB4_GRP1_ForceReset
		LL_AHB4_GRP1_ReleaseReset
	SDMMC2RST	LL_AHB4_GRP1_ForceReset
		LL_AHB4_GRP1_ReleaseReset
APB1HENR	DTSEN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	FDCANEN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	LPTIM2EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock

Register	Field	Function
APB1HENR	UART12EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	UART9EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	UCPD1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
APB1HLPENR	DTSLPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	FDCAN12LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	FDCAN1LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	LPTIM2LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	UART12LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	UART9LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
	UCPD1LPEN	LL_APB1_GRP2_DisableClockSleep
		LL_APB1_GRP2_EnableClockSleep
		LL_APB1_GRP2_IsEnabledClockSleep
APB1HRSTR	DTSRST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	FDCAN	LL_APB1_GRP2_ReleaseReset
	FDCANRST	LL_APB1_GRP2_ForceReset
	LPTIM2RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	UART12RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	UART9RST	LL_APB1_GRP2_ForceReset
		LL_APB1_GRP2_ReleaseReset
	UCPD1RST	LL_APB1_GRP2_ForceReset

Register	Field	Function
APB1HRSTR	UCPD1RST	LL_APB1_GRP2_ReleaseReset
APB1LENR	CECEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	CRSEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C1EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I3C1EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM12EN	LL_APB1_GRP1_EnableClock
	TIM13EN	LL_APB1_GRP1_EnableClock
	TIM14EN	LL_APB1_GRP1_EnableClock
	TIM2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM4EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM5EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM6EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM7EN	LL_APB1_GRP1_DisableClock

Register	Field	Function
APB1LENR	TIM7EN	LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART4EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART5EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART7EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART8EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART10EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART11EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART6EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	WWDGEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1LLPENR	CECLPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	CRSLPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	I2C1LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep

Register	Field	Function
APB1LLPENR	I2C2LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	I3C1LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	SPI2LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	SPI3LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM2LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM3LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM4LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM5LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM6LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	TIM7LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	UART4LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	UART5LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	UART7LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	UART8LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep

Register	Field	Function
APB1LLPENR	UART8LPEN	LL_APB1_GRP1_IsEnabledClockSleep
	USART10LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	USART11LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	USART2LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	USART3LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	USART6LPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
	WWDGLPEN	LL_APB1_GRP1_DisableClockSleep
		LL_APB1_GRP1_EnableClockSleep
		LL_APB1_GRP1_IsEnabledClockSleep
APB1LRSTR	CECRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	CRSRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I2C1RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I2C2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I3C1RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	SPI2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	SPI3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM4RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM5RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset

Register	Field	Function
APB1LRSTR	TIM6RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM7RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART4RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART5RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART7RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART8RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART10RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART11RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART6RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
APB2ENR	SAI1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SAI2EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI4EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI6EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM15EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM16EN	LL_APB2_GRP1_DisableClock

Register	Field	Function
APB2ENR	TIM16EN	LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM17EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM8EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USART1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USBEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2LPENR	SAI1LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	SAI2LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	SPI1LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	SPI4LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	SPI6LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	TIM15LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	TIM16LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	TIM17LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep

Register	Field	Function
APB2LPENR	TIM1LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	TIM8LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	USART1LPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
	USBLPEN	LL_APB2_GRP1_DisableClockSleep
		LL_APB2_GRP1_EnableClockSleep
		LL_APB2_GRP1_IsEnabledClockSleep
APB2RSTR	SAI1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SAI2RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SPI1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SPI4RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SPI6RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM15RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM16RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM17RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM8RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	USART1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	USBRST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
CFGR2	AHB1DIS	LL_AHB_DisableClock
		LL_AHB_EnableClock
		LL_AHB_IsDisabledClock
	AHB2DIS	LL_AHB_DisableClock
		LL_AHB_EnableClock

Register	Field	Function
CFGR2	AHB2DIS	LL_AHB_IsDisabledClock
	AHB4DIS	LL_AHB_DisableClock
		LL_AHB_EnableClock
		LL_AHB_IsDisabledClock
	APB1DIS	LL_APB_DisableClock
		LL_APB_EnableClock
		LL_APB_IsDisabledClock
	APB2DIS	LL_APB_DisableClock
		LL_APB_EnableClock
		LL_APB_IsDisabledClock
	APB3DIS	LL_APB_DisableClock
		LL_APB_EnableClock
		LL_APB_IsDisabledClock

119.3 COMP

Table 27. Correspondence between COMP registers and COMP low-layer driver functions

Register	Field	Function
CFGR	BLANKING	LL_COMP_GetOutputBlankingSource
		LL_COMP_SetOutputBlankingSource
	EN	LL_COMP_Disable
		LL_COMP_Enable
		LL_COMP_IsEnabled
	LOCK	LL_COMP_IsLocked
		LL_COMP_Lock
	VALUE	LL_COMP_ReadOutputLevel
CFGR1	BRGEN	LL_COMP_ConfigInputs
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	HYST	LL_COMP_SetInputHysteresis
	INMSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	INPSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	POLARITY	LL_COMP_GetOutputPolarity
		LL_COMP_SetOutputPolarity
	PWRMODE	LL_COMP_GetPowerMode
		LL_COMP_SetPowerMode
	SCALEN	LL_COMP_ConfigInputs

Register	Field	Function
CFGR1	SCALEN	LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
CSR	HYST	LL_COMP_GetInputHysteresis

119.4 CORDIC

Table 28. Correspondence between CORDIC registers and CORDIC low-layer driver functions

Register	Field	Function
CSR	ARGSIZE	LL_CORDIC_Config
		LL_CORDIC_GetInSize
		LL_CORDIC_SetInSize
	DMAREN	LL_CORDIC_DisableDMAReq_RD
		LL_CORDIC_EnableDMAReq_RD
		LL_CORDIC_IsEnabledDMAReq_RD
	DMAWEN	LL_CORDIC_DisableDMAReq_WR
		LL_CORDIC_EnableDMAReq_WR
		LL_CORDIC_IsEnabledDMAReq_WR
	FUNC	LL_CORDIC_Config
		LL_CORDIC_GetFunction
		LL_CORDIC_SetFunction
	IEN	LL_CORDIC_DisableIT
		LL_CORDIC_EnableIT
		LL_CORDIC_IsEnabledIT
	NARGS	LL_CORDIC_Config
		LL_CORDIC_GetNbWrite
		LL_CORDIC_SetNbWrite
	NRES	LL_CORDIC_Config
		LL_CORDIC_GetNbRead
		LL_CORDIC_SetNbRead
	PRECISION	LL_CORDIC_Config
		LL_CORDIC_GetPrecision
		LL_CORDIC_SetPrecision
	RESIZE	LL_CORDIC_Config
		LL_CORDIC_GetOutSize
		LL_CORDIC_SetOutSize
	RRDY	LL_CORDIC_IsActiveFlag_RRDY
	SCALE	LL_CORDIC_Config
		LL_CORDIC_GetScale
		LL_CORDIC_SetScale
RDATA	RES	LL_CORDIC_DMA_GetRegAddr
		LL_CORDIC_ReadData

Register	Field	Function
WDATA	ARG	LL_CORDIC_DMA_GetRegAddr
		LL_CORDIC_WriteData

119.5 CORTEX

Table 29. Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	LL_MPU_Disable
		LL_MPU_Disable_NS
		LL_MPU_Enable
		LL_MPU_IsEnabled
		LL_MPU_IsEnabled_NS
MPU_MAIR0	Attribute	LL_MPU_ConfigAttributes
		LL_MPU_ConfigAttributes_NS
MPU_MAIR1	Attribute	LL_MPU_ConfigAttributes
		LL_MPU_ConfigAttributes_NS
MPU_RBAR	ADDR	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_SetRegionBaseAddress
		LL_MPU_SetRegionBaseAddress_NS
	AP	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
	SH	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
	XN	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
MPU_RLAR	ADDR	LL_MPU_ConfigRegion

Register	Field	Function
MPU_RLAR	ADDR	LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_SetRegionLimitAddress
		LL_MPU_SetRegionLimitAddress_NS
	AttrIndx	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
	EN	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
	ENABLE	LL_MPU_DisableRegion
		LL_MPU_DisableRegion_NS
		LL_MPU_EnableRegion
		LL_MPU_EnableRegion_NS
MPU_RNR	ENABLE	LL_MPU_IsEnabled_Region
		LL_MPU_IsEnabled_Region_NS
	REGION	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_DisableRegion
		LL_MPU_DisableRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_GetRegionBaseAddress
		LL_MPU_GetRegionBaseAddress_NS
		LL_MPU_GetRegionLimitAddress
		LL_MPU_GetRegionLimitAddress_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
		LL_MPU_SetRegionBaseAddress
		LL_MPU_SetRegionBaseAddress_NS
		LL_MPU_SetRegionLimitAddress
		LL_MPU_SetRegionLimitAddress_NS
SCB_CPUID	ARCHITECTURE	LL_CPUID_GetArchitecture
	IMPLEMENTER	LL_CPUID_GetImplementer
	PARTNO	LL_CPUID_GetParNo
	REVISION	LL_CPUID_GetRevision
	VARIANT	LL_CPUID_GetVariant
SCB_SCR	SEVEONPEND	LL_LPM_DisableEventOnPend
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	LL_LPM_EnableDeepSleep

Register	Field	Function
SCB_SCR	SLEEPDEEP	LL_LPM_EnableSleep
	SLEEPONEXIT	LL_LPM_DisableSleepOnExit
		LL_LPM_EnableSleepOnExit
SCB_SHCSR	BUSFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	MEMFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	SECUREFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	USGFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
STK_CTRL	CLKSOURCE	LL_SYSTICK_GetClkSource
		LL_SYSTICK_SetClkSource
	COUNTFLAG	LL_SYSTICK_IsActiveCounterFlag
	TICKINT	LL_SYSTICK_DisableIT
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

119.6 CRC

Table 30. Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	POLYSIZE	LL_CRC_GetPolynomialSize
		LL_CRC_SetPolynomialSize
	RESET	LL_CRC_ResetCRCCalculationUnit
	REV_IN	LL_CRC_GetInputDataReverseMode
		LL_CRC_SetInputDataReverseMode
	REV_OUT	LL_CRC_GetOutputDataReverseMode
		LL_CRC_SetOutputDataReverseMode
DR	DR	LL_CRC_FeedData16
		LL_CRC_FeedData32
		LL_CRC_FeedData8
		LL_CRC_ReadData16
		LL_CRC_ReadData32
		LL_CRC_ReadData7
		LL_CRC_ReadData8
IDR	IDR	LL_CRC_Read_IDR
		LL_CRC_Write_IDR
INIT	INIT	LL_CRC_GetInitialData
		LL_CRC_SetInitialData
POL	POL	LL_CRC_GetPolynomialCoef

Register	Field	Function
POL	POL	LL_CRC_SetPolynomialCoef

119.7 CRS

Table 31. Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFGR	FELIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetFreqErrorLimit
		LL_CRS_SetFreqErrorLimit
	RELOAD	LL_CRS_ConfigSynchronization
		LL_CRS_GetReloadCounter
		LL_CRS_SetReloadCounter
	SYNCDIV	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncDivider
		LL_CRS_SetSyncDivider
	SYNCPOL	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncPolarity
		LL_CRS_SetSyncPolarity
	SYNCSRC	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncSignalSource
		LL_CRS_SetSyncSignalSource
CR	AUTOTRIMEN	LL_CRS_DisableAutoTrimming
		LL_CRS_EnableAutoTrimming
		LL_CRS_IsEnabledAutoTrimming
	CEN	LL_CRS_DisableFreqErrorCounter
		LL_CRS_EnableFreqErrorCounter
		LL_CRS_IsEnabledFreqErrorCounter
	ERRIE	LL_CRS_DisableIT_ERR
		LL_CRS_EnableIT_ERR
		LL_CRS_IsEnabledIT_ERR
	ESYNCIE	LL_CRS_DisableIT_ESYNC
		LL_CRS_EnableIT_ESYNC
		LL_CRS_IsEnabledIT_ESYNC
	SWSYNC	LL_CRS_GenerateEvent_SWSYNC
	SYNCOKIE	LL_CRS_DisableIT_SYNCOK
		LL_CRS_EnableIT_SYNCOK
		LL_CRS_IsEnabledIT_SYNCOK
	SYNCWARNIE	LL_CRS_DisableIT_SYNCWARN
		LL_CRS_EnableIT_SYNCWARN
		LL_CRS_IsEnabledIT_SYNCWARN
	TRIM	LL_CRS_ConfigSynchronization

Register	Field	Function
CR	TRIM	LL_CRS_GetHSI48SmoothTrimming
		LL_CRS_SetHSI48SmoothTrimming
ICR	ERRC	LL_CRS_ClearFlag_ERR
	ESYNCC	LL_CRS_ClearFlag_ESYNC
	SYNCOKC	LL_CRS_ClearFlag_SYNCOK
	SYNCWARNC	LL_CRS_ClearFlag_SYNCWARN
ISR	ERRF	LL_CRS_IsActiveFlag_ERR
	ESYNCF	LL_CRS_IsActiveFlag_ESYNC
	FECAP	LL_CRS_GetFreqErrorCapture
	FEDIR	LL_CRS_GetFreqErrorDirection
	SYNCERR	LL_CRS_IsActiveFlag_SYNCERR
	SYNCMISS	LL_CRS_IsActiveFlag_SYNCMISS
	SYNCOKF	LL_CRS_IsActiveFlag_SYNCOK
	SYNCWARNF	LL_CRS_IsActiveFlag_SYNCWARN
	TRIMOVF	LL_CRS_IsActiveFlag_TRIMOVF

119.8 DAC

Table 32. Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
CCR	OTRIM1	LL_DAC_GetTrimmingValue
		LL_DAC_SetTrimmingValue
	OTRIM2	LL_DAC_GetTrimmingValue
		LL_DAC_SetTrimmingValue
CR	CEN1	LL_DAC_GetMode
		LL_DAC_SetMode
	CEN2	LL_DAC_GetMode
		LL_DAC_SetMode
	DMAEN1	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAEN2	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAUDRIE1	LL_DAC_DisableIT_DMAUDR1
		LL_DAC_EnableIT_DMAUDR1
		LL_DAC_IsEnabledIT_DMAUDR1
	DMAUDRIE2	LL_DAC_DisableIT_DMAUDR2
		LL_DAC_EnableIT_DMAUDR2
		LL_DAC_IsEnabledIT_DMAUDR2
	EN1	LL_DAC_Disable

Register	Field	Function
CR	EN1	LL_DAC_Enable
		LL_DAC_IsEnabled
	EN2	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	MAMP1	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MODE1	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer
		LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	MODE2	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer
		LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	TEN1	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TSEL1	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	TSEL2	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	WAVE1	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration
	WAVE2	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration

Register	Field	Function
DHR12L1	DACC1DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	LL_DAC_ConvertDualData12LeftAligned
	DACC2DHR	LL_DAC_ConvertDualData12LeftAligned
DHR12R1	DACC1DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	LL_DAC_ConvertDualData12RightAligned
	DACC2DHR	LL_DAC_ConvertDualData12RightAligned
DHR8R1	DACC1DHR	LL_DAC_ConvertData8RightAligned
		LL_DAC_DMA_GetRegAddr
DHR8R2	DACC2DHR	LL_DAC_ConvertData8RightAligned
		LL_DAC_DMA_GetRegAddr
DHR8RD	DACC1DHR	LL_DAC_ConvertDualData8RightAligned
	DACC2DHR	LL_DAC_ConvertDualData8RightAligned
DOR1	DACC1DOR	LL_DAC_RetrieveOutputData
DOR2	DACC2DOR	LL_DAC_RetrieveOutputData
MCR	DMADOUBLE1	LL_DAC_DisableDMADoubleDataMode
		LL_DAC_EnableDMADoubleDataMode
		LL_DAC_IsDMADoubleDataModeEnabled
	DMADOUBLE2	LL_DAC_DisableDMADoubleDataMode
		LL_DAC_EnableDMADoubleDataMode
		LL_DAC_IsDMADoubleDataModeEnabled
	HFSEL	LL_DAC_GetHighFrequencyMode
		LL_DAC_SetHighFrequencyMode
	SINFORMAT1	LL_DAC_GetSignedFormat
		LL_DAC_SetSignedFormat
	SINFORMAT2	LL_DAC_GetSignedFormat
		LL_DAC_SetSignedFormat
SHHR	THOLD1	LL_DAC_GetSampleAndHoldHoldTime
		LL_DAC_SetSampleAndHoldHoldTime
	THOLD2	LL_DAC_GetSampleAndHoldHoldTime
		LL_DAC_SetSampleAndHoldHoldTime
SHRR	TREFRESH1	LL_DAC_GetSampleAndHoldRefreshTime
		LL_DAC_SetSampleAndHoldRefreshTime
	TREFRESH2	LL_DAC_GetSampleAndHoldRefreshTime
		LL_DAC_SetSampleAndHoldRefreshTime
SHSR1	TSAMPLE1	LL_DAC_GetSampleAndHoldSampleTime

Register	Field	Function
SHSR1	TSAMPLE1	LL_DAC_SetSampleAndHoldSampleTime
SHSR2	TSAMPLE2	LL_DAC_GetSampleAndHoldSampleTime
		LL_DAC_SetSampleAndHoldSampleTime
SR	BWST1	LL_DAC_IsActiveFlag_BWST1
	BWST2	LL_DAC_IsActiveFlag_BWST2
	CAL_FLAG1	LL_DAC_IsActiveFlag_CAL1
	CAL_FLAG2	LL_DAC_IsActiveFlag_CAL2
	DAC1RDY	LL_DAC_IsActiveFlag_DAC1RDY
		LL_DAC_IsReady
	DAC2RDY	LL_DAC_IsActiveFlag_DAC2RDY
		LL_DAC_IsReady
	DMAUDR1	LL_DAC_ClearFlag_DMAUDR1
		LL_DAC_IsActiveFlag_DMAUDR1
	DMAUDR2	LL_DAC_ClearFlag_DMAUDR2
		LL_DAC_IsActiveFlag_DMAUDR2
	DORSTAT1	LL_DAC_IsActiveFlag_DORSTAT1
	DORSTAT2	LL_DAC_IsActiveFlag_DORSTAT2
SWTRIGR	SWTRIG1	LL_DAC_TrigSWConversion
	SWTRIG2	LL_DAC_TrigSWConversion

119.9 DCACHE

Table 33. Correspondence between DCACHE registers and DCACHE low-layer driver functions

Register	Field	Function
CR	(RHITMRST/RMISSMRST/WHITMRST/WMISSMRST)	LL_DCACHE_ResetMonitors
	(RMISSMEN/RHITMEN/WMISSMEN/WHITMEN)	LL_DCACHE_DisableMonitors
		LL_DCACHE_EnableMonitors
		LL_DCACHE_IsEnabledMonitors
	CACHECMD	LL_DCACHE_GetCommand
		LL_DCACHE_SetCommand
		LL_DCACHE_StartCommand
	CACHEINV	LL_DCACHE_Invalidate
	CMDREADDRR	LL_DCACHE_GetEndAddress
		LL_DCACHE_SetEndAddress
	CMDRSADDRR	LL_DCACHE_GetStartAddress
		LL_DCACHE_SetStartAddress
	EN	LL_DCACHE_Disable
		LL_DCACHE_Enable
		LL_DCACHE_IsEnabled
	HBURST	LL_DCACHE_GetReadBurstType
		LL_DCACHE_SetReadBurstType

Register	Field	Function
FCR	CBSYENDF	LL_DCACHE_ClearFlag_BSYEND
	CCMDENDF	LL_DCACHE_ClearFlag_CMDEND
	CERRF	LL_DCACHE_ClearFlag_ERR
IER	BSYENDIE	LL_DCACHE_DisableIT_BSYEND
		LL_DCACHE_EnableIT_BSYEND
		LL_DCACHE_IsEnabledIT_BSYEND
	CMDENDIE	LL_DCACHE_DisableIT_CMDEND
		LL_DCACHE_EnableIT_CMDEND
		LL_DCACHE_IsEnabledIT_CMDEND
	ERRIE	LL_DCACHE_DisableIT_ERR
		LL_DCACHE_EnableIT_ERR
		LL_DCACHE_IsEnabledIT_ERR
SR	BSYENDF	LL_DCACHE_IsActiveFlag_BSYEND
	BUSYCMDF	LL_DCACHE_IsActiveFlag_BUSYCMD
	BUSYF	LL_DCACHE_IsActiveFlag_BUSY
	CMDENDF	LL_DCACHE_IsActiveFlag_CMDEND
	ERRF	LL_DCACHE_IsActiveFlag_ERR

119.10 DMA

Table 34. Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CBR1	BNDT	LL_DMA_ConfigBlkCounters
		LL_DMA_GetBlkDataLength
		LL_DMA_SetBlkDataLength
	BRC	LL_DMA_ConfigBlkCounters
		LL_DMA_GetBlkRptCount
		LL_DMA_SetBlkRptCount
	BRDDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetBlkRptDestAddrUpdate
		LL_DMA_SetBlkRptDestAddrUpdate
	BRSDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetBlkRptSrcAddrUpdate
		LL_DMA_SetBlkRptSrcAddrUpdate
	DDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetDestAddrUpdate
		LL_DMA_SetDestAddrUpdate
	SDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetSrcAddrUpdate
		LL_DMA_SetSrcAddrUpdate
CBR2	BRDAO	LL_DMA_ConfigBlkRptAddrUpdateValue

Register	Field	Function
CBR2	BRDAO	LL_DMA_GetBlkRptDestAddrUpdateValue
		LL_DMA_SetBlkRptDestAddrUpdateValue
	BRSAO	LL_DMA_ConfigBlkRptAddrUpdateValue
		LL_DMA_GetBlkRptSrcAddrUpdateValue
		LL_DMA_SetBlkRptSrcAddrUpdateValue
CCR	DTEIE	LL_DMA_DisableIT_DTE
		LL_DMA_EnableIT_DTE
		LL_DMA_IsEnabledIT_DTE
	EN	LL_DMA_DisableChannel
		LL_DMA_EnableChannel
		LL_DMA_IsEnabledChannel
	HTIE	LL_DMA_DisableIT_HT
		LL_DMA_EnableIT_HT
		LL_DMA_IsEnabledIT_HT
	LAP	LL_DMA_ConfigControl
		LL_DMA_GetLinkAllocatedPort
		LL_DMA_SetLinkAllocatedPort
	LSM	LL_DMA_ConfigControl
		LL_DMA_GetLinkStepMode
		LL_DMA_SetLinkStepMode
	PRIO	LL_DMA_ConfigControl
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	RESET	LL_DMA_ResetChannel
	SUSP	LL_DMA_IsSuspendedChannel
		LL_DMA_ResumeChannel
		LL_DMA_SuspendChannel
	SUSPIE	LL_DMA_DisableIT_SUSP
		LL_DMA_EnableIT_SUSP
		LL_DMA_IsEnabledIT_SUSP
	TCIE	LL_DMA_DisableIT_TC
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TOIE	LL_DMA_DisableIT_TO
		LL_DMA_EnableIT_TO
		LL_DMA_IsEnabledIT_TO
	ULEIE	LL_DMA_DisableIT_ULE
		LL_DMA_EnableIT_ULE
		LL_DMA_IsEnabledIT_ULE
	USEIE	LL_DMA_DisableIT_USE
		LL_DMA_EnableIT_USE

Register	Field	Function
CCR	USEIE	LL_DMA_IsEnabledIT_USE
CDAR	DA	LL_DMA_ConfigAddresses
		LL_DMA_GetDestAddress
		LL_DMA_SetDestAddress
	DAO	LL_DMA_GetDestAddrUpdateValue
CFCR	DTEF	LL_DMA_ClearFlag_DTE
	HTF	LL_DMA_ClearFlag_HT
	SUSPF	LL_DMA_ClearFlag_SUSP
	TCF	LL_DMA_ClearFlag_TC
	TOF	LL_DMA_ClearFlag_TO
	ULEF	LL_DMA_ClearFlag_ULE
	USEF	LL_DMA_ClearFlag_USE
CLBAR	LBA	LL_DMA_GetLinkedListBaseAddr
		LL_DMA_SetLinkedListBaseAddr
CLLR	LA	LL_DMA_GetLinkedListAddrOffset
		LL_DMA_SetLinkedListAddrOffset
	UB1	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCBR1Update
		LL_DMA_EnableCBR1Update
		LL_DMA_IsEnabledCBR1Update
	UB2	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCBR2Update
		LL_DMA_EnableCBR2Update
		LL_DMA_IsEnabledCBR2Update
	UDA	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCDARUpdate
		LL_DMA_EnableCDARUpdate
		LL_DMA_IsEnabledCDARUpdate
	ULL	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCLLRUpdate
		LL_DMA_EnableCLLRUpdate
		LL_DMA_IsEnabledCLLRUpdate
	USA	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCSARUpdate
		LL_DMA_EnableCSARUpdate
		LL_DMA_IsEnabledCSARUpdate
	UT1	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCTR1Update
		LL_DMA_EnableCTR1Update
		LL_DMA_IsEnabledCTR1Update
	UT2	LL_DMA_ConfigLinkUpdate

Register	Field	Function
CLLR	UT2	LL_DMA_DisableCTR2Update
		LL_DMA_EnableCTR2Update
		LL_DMA_IsEnabledCTR2Update
	UT3	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCTR3Update
		LL_DMA_EnableCTR3Update
		LL_DMA_IsEnabledCTR3Update
CSAR	SA	LL_DMA_ConfigAddresses
		LL_DMA_GetSrcAddress
		LL_DMA_SetSrcAddress
CSR	DTEF	LL_DMA_IsActiveFlag_DTE
	FIFOL	LL_DMA_GetFIFOLevel
	HTF	LL_DMA_IsActiveFlag_HT
	IDLEF	LL_DMA_IsActiveFlag_IDLE
	SUSPF	LL_DMA_IsActiveFlag_SUSP
	TCF	LL_DMA_IsActiveFlag_TC
	TOF	LL_DMA_IsActiveFlag_TO
	ULEF	LL_DMA_IsActiveFlag_ULE
	USEF	LL_DMA_IsActiveFlag_USE
CTR1	DAP	LL_DMA_ConfigTransfer
		LL_DMA_GetDestAllocatedPort
		LL_DMA_SetDestAllocatedPort
	DBL_1	LL_DMA_GetDestBurstLength
		LL_DMA_SetDestBurstLength
	DBX	LL_DMA_ConfigTransfer
		LL_DMA_GetDestByteExchange
		LL_DMA_SetDestByteExchange
	DDW_LOG2	LL_DMA_GetDestDataWidth
		LL_DMA_SetDestDataWidth
	DHX	LL_DMA_ConfigTransfer
		LL_DMA_GetDestHWordExchange
		LL_DMA_SetDestHWordExchange
	DINC	LL_DMA_ConfigTransfer
		LL_DMA_GetDestIncMode
		LL_DMA_SetDestIncMode
	DSEC	LL_DMA_ConfigChannelSecure
		LL_DMA_DisableChannelDestSecure
		LL_DMA_EnableChannelDestSecure
		LL_DMA_IsEnabledChannelDestSecure
	PAM	LL_DMA_ConfigTransfer
		LL_DMA_GetDataAlignment

Register	Field	Function
CTR1	PAM	LL_DMA_SetDataAlignment
	SAP	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcAllocatedPort
		LL_DMA_SetSrcAllocatedPort
	SBL_1	LL_DMA_GetSrcBurstLength
		LL_DMA_SetDestBurstLength
		LL_DMA_SetSrcBurstLength
	SBX	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcByteExchange
		LL_DMA_SetSrcByteExchange
	SDW_LOG2	LL_DMA_GetSrcDataWidth
		LL_DMA_SetSrcDataWidth
	SINC	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcIncMode
		LL_DMA_SetSrcIncMode
	SSEC	LL_DMA_ConfigChannelSecure
		LL_DMA_DisableChannelSrcSecure
		LL_DMA_EnableChannelSrcSecure
		LL_DMA_IsEnabledChannelSrcSecure
CTR2	BREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetBlkHWRequest
		LL_DMA_SetBlkHWRequest
	DREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	PFREQ	LL_DMA_SetTransferMode
	REQSEL	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	SWREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	TCEM	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTransferEventMode
		LL_DMA_SetTransferEventMode
	TRIGM	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTriggerMode
		LL_DMA_SetTriggerMode
	TRIGPOL	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTriggerPolarity
		LL_DMA_SetTriggerPolarity
	TRIGSEL	LL_DMA_GetHWTrigger

Register	Field	Function
CTR2	TRIGSEL	LL_DMA_GetTransferMode
		LL_DMA_SetHWTrigger
CTR3	DAO	LL_DMA_ConfigAddrUpdateValue
		LL_DMA_SetDestAddrUpdateValue
	SAO	LL_DMA_ConfigAddrUpdateValue
		LL_DMA_GetSrcAddrUpdateValue
		LL_DMA_SetSrcAddrUpdateValue
MISR	MISx	LL_DMA_IsActiveFlag_MIS
PRIVCFGR	PRIVx	LL_DMA_DisableChannelPrivilege
		LL_DMA_EnableChannelPrivilege
		LL_DMA_IsEnabledChannelPrivilege
RCFGLOCKR	LOCKx	LL_DMA_EnableChannelLockAttribute
SECCFGR	LOCKx	LL_DMA_IsEnabledChannelLockAttribute
	SEC	LL_DMA_ConfigChannelSecure
	SECx	LL_DMA_DisableChannelSecure
		LL_DMA_EnableChannelSecure
		LL_DMA_IsEnabledChannelSecure
SMISR	MISx	LL_DMA_IsActiveFlag_SMIS

119.11 EXTI

Table 35. Correspondence between EXTI registers and EXTI low-layer driver functions

Register	Field	Function
EMR1	EMx	LL_EXTI_DisableEvent_0_31
		LL_EXTI_EnableEvent_0_31
		LL_EXTI_IsEnabledEvent_0_31
EMR2	EMx	LL_EXTI_DisableEvent_32_63
		LL_EXTI_EnableEvent_32_63
		LL_EXTI_IsEnabledEvent_32_63
EXTI_EXTICR1	EXTI0	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI1	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI2	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI3	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR2	EXTI4	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI5	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource

Register	Field	Function
EXTI_EXTICR2	EXTI6	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI7	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR3	EXTI10	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI11	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI8	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI9	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR4	EXTI12	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI13	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI14	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI15	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
FPR1	FPIFx	LL_EXTI_ClearFallingFlag_0_31
		LL_EXTI_IsActiveFallingFlag_0_31
		LL_EXTI_ReadFallingFlag_0_31
FPR2	FPIFx	LL_EXTI_ClearFallingFlag_32_63
		LL_EXTI_IsActiveFallingFlag_32_63
		LL_EXTI_ReadFallingFlag_32_63
FTSR1	FTx	LL_EXTI_DisableFallingTrig_0_31
		LL_EXTI_EnableFallingTrig_0_31
		LL_EXTI_IsEnabledFallingTrig_0_31
FTSR2	FTx	LL_EXTI_DisableFallingTrig_32_63
		LL_EXTI_EnableFallingTrig_32_63
		LL_EXTI_IsEnabledFallingTrig_32_63
IMR1	IMx	LL_EXTI_DisableIT_0_31
		LL_EXTI_EnableIT_0_31
		LL_EXTI_IsEnabledIT_0_31
IMR2	IMx	LL_EXTI_DisableIT_32_63
		LL_EXTI_EnableIT_32_63
		LL_EXTI_IsEnabledIT_32_63
LOCKR	LOCK	LL_EXTI_EnableLockAttribute
		LL_EXTI_IsEnabledLockAttribute
PRIVCFGR1	PRIVx	LL_EXTI_DisablePrivilege_0_31

Register	Field	Function
PRIVCFGR1	PRIVx	LL_EXTI_EnablePrivilege_0_31
		LL_EXTI_IsEnabledPrivilege_0_31
PRIVCFGR2	PRIVx	LL_EXTI_EnablePrivilege_32_63
		LL_EXTI_IsEnabledPrivilege_32_63
RPR1	RPIFx	LL_EXTI_ClearRisingFlag_0_31
		LL_EXTI_IsActiveRisingFlag_0_31
		LL_EXTI_ReadRisingFlag_0_31
RPR2	RPIFx	LL_EXTI_ClearRisingFlag_32_63
		LL_EXTI_IsActiveRisingFlag_32_63
		LL_EXTI_ReadRisingFlag_32_63
RTSR1	RTx	LL_EXTI_DisableRisingTrig_0_31
		LL_EXTI_EnableRisingTrig_0_31
		LL_EXTI_IsEnabledRisingTrig_0_31
RTSR2	RTx	LL_EXTI_DisableRisingTrig_32_63
		LL_EXTI_EnableRisingTrig_32_63
		LL_EXTI_IsEnabledRisingTrig_32_63
SECCFGR1	SECx	LL_EXTI_DisableSecure_0_31
		LL_EXTI_EnableSecure_0_31
		LL_EXTI_IsEnabledSecure_0_31
SECCFGR2	SECx	LL_EXTI_DisableSecure_32_63
		LL_EXTI_EnableSecure_32_63
		LL_EXTI_IsEnabledSecure_32_63
SWIER1	SWIx	LL_EXTI_GenerateSWI_0_31
SWIER2	SWIx	LL_EXTI_GenerateSWI_32_63

119.12 FMAC

Table 36. Correspondence between FMAC registers and FMAC low-layer driver functions

Register	Field	Function
CR	CLIPEN	LL_FMAC_DisableClipping
		LL_FMAC_EnableClipping
		LL_FMAC_IsEnabledClipping
	DMAREN	LL_FMAC_DisableDMAReq_READ
		LL_FMAC_EnableDMAReq_READ
		LL_FMAC_IsEnabledDMAReq_READ
	DMAWEN	LL_FMAC_DisableDMAReq_WRITE
		LL_FMAC_EnableDMAReq_WRITE
		LL_FMAC_IsEnabledDMAReq_WRITE
	OVFLIEN	LL_FMAC_DisableIT_OVFL
		LL_FMAC_EnableIT_OVFL
		LL_FMAC_IsEnabledIT_OVFL

Register	Field	Function
CR	RESET	LL_FMAC_EnableReset
		LL_FMAC_IsEnabledReset
	RIEN	LL_FMAC_DisableIT_RD
		LL_FMAC_EnableIT_RD
		LL_FMAC_IsEnabledIT_RD
	SATIEN	LL_FMAC_DisableIT_SAT
		LL_FMAC_EnableIT_SAT
		LL_FMAC_IsEnabledIT_SAT
	UNFLIEN	LL_FMAC_DisableIT_UNFL
		LL_FMAC_EnableIT_UNFL
		LL_FMAC_IsEnabledIT_UNFL
	WIEN	LL_FMAC_DisableIT_WR
		LL_FMAC_EnableIT_WR
		LL_FMAC_IsEnabledIT_WR
PARAM	FUNC	LL_FMAC_ConfigFunc
		LL_FMAC_GetFunction
		LL_FMAC_SetFunction
	P	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamP
		LL_FMAC_SetParamP
	Q	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamQ
		LL_FMAC_SetParamQ
	R	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamR
		LL_FMAC_SetParamR
	START	LL_FMAC_ConfigFunc
		LL_FMAC_DisableStart
		LL_FMAC_EnableStart
		LL_FMAC_IsEnabledStart
RDATA	RDATA	LL_FMAC_ReadData
SR	OVFL	LL_FMAC_IsActiveFlag_OVFL
	SAT	LL_FMAC_IsActiveFlag_SAT
	UNFL	LL_FMAC_IsActiveFlag_UNFL
	X1FULL	LL_FMAC_IsActiveFlag_X1FULL
	YEMPTY	LL_FMAC_IsActiveFlag_YEMPTY
WDATA	WDATA	LL_FMAC_WriteData
X1BUFCFG	FULL_WM	LL_FMAC_ConfigX1
		LL_FMAC_GetX1FullWatermark
		LL_FMAC_SetX1FullWatermark
	X1_BASE	LL_FMAC_ConfigX1

Register	Field	Function
X1BUFCFG	X1_BASE	LL_FMAC_GetX1Base
		LL_FMAC_SetX1Base
	X1_BUF_SIZE	LL_FMAC_ConfigX1
		LL_FMAC_GetX1BufferSize
		LL_FMAC_SetX1BufferSize
X2BUFCFG	X2_BASE	LL_FMAC_ConfigX2
		LL_FMAC_GetX2Base
		LL_FMAC_SetX2Base
	X2_BUF_SIZE	LL_FMAC_ConfigX2
		LL_FMAC_GetX2BufferSize
		LL_FMAC_SetX2BufferSize
YBUFCFG	EMPTY_WM	LL_FMAC_ConfigY
		LL_FMAC_GetYEmptyWatermark
		LL_FMAC_SetYEmptyWatermark
	Y_BASE	LL_FMAC_ConfigY
		LL_FMAC_GetYBase
		LL_FMAC_SetYBase
	Y_BUF_SIZE	LL_FMAC_ConfigY
		LL_FMAC_GetYBufferSize
		LL_FMAC_SetYBufferSize

119.13 GPIO

Table 37. Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
AFRH	AFSELy	LL_GPIO_GetAFPin_8_15
		LL_GPIO_SetAFPin_8_15
AFRL	AFSELy	LL_GPIO_GetAFPin_0_7
		LL_GPIO_SetAFPin_0_7
BRR	BRy	LL_GPIO_ResetOutputPin
BSRR	BSy	LL_GPIO_SetOutputPin
HSLVR	HSLVy	LL_GPIO_DisableHighSPeedLowVoltage
		LL_GPIO_EnableHighSPeedLowVoltage
		LL_GPIO_IsEnabledHighSPeedLowVoltage
IDR	IDy	LL_GPIO_IsInputPinSet
		LL_GPIO_ReadInputPort
LCKR	LCKK	LL_GPIO_IsAnyPinLocked
		LL_GPIO_LockPin
	LCKy	LL_GPIO_IsPinLocked
MODER	MODEy	LL_GPIO_GetPinMode
		LL_GPIO_SetPinMode

Register	Field	Function
ODR	ODy	LL_GPIO_IsOutputPinSet
		LL_GPIO_ReadOutputPort
		LL_GPIO_TogglePin
		LL_GPIO_WriteOutputPort
OSPEEDR	OSPEEDy	LL_GPIO_GetPinSpeed
		LL_GPIO_SetPinSpeed
OTYPER	OTy	LL_GPIO_GetPinOutputType
		LL_GPIO_SetPinOutputType
PUPDR	PUPDy	LL_GPIO_GetPinPull
		LL_GPIO_SetPinPull
SECCFGR	SECy	LL_GPIO_DisablePinSecure
		LL_GPIO_EnablePinSecure
		LL_GPIO_IsEnabledPinSecure

119.14 I2C

Table 38. Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
CR1	ADDRACLR	LL_I2C_DisableAutoClearFlag_ADDR
		LL_I2C_EnableAutoClearFlag_ADDR
		LL_I2C_IsEnabledAutoClearFlag_ADDR
	ADDRIE	LL_I2C_DisableIT_ADDR
		LL_I2C_EnableIT_ADDR
		LL_I2C_IsEnabledIT_ADDR
	ALERTEN	LL_I2C_DisableSMBusAlert
		LL_I2C_EnableSMBusAlert
		LL_I2C_IsEnabledSMBusAlert
	ANFOFF	LL_I2C_ConfigFilters
		LL_I2C_DisableAnalogFilter
		LL_I2C_EnableAnalogFilter
		LL_I2C_IsEnabledAnalogFilter
	DNF	LL_I2C_ConfigFilters
		LL_I2C_GetDigitalFilter
		LL_I2C_SetDigitalFilter
	ERRIE	LL_I2C_DisableIT_ERR
		LL_I2C_EnableIT_ERR
		LL_I2C_IsEnabledIT_ERR
	FMP	LL_I2C_DisableFastModePlus
		LL_I2C_EnableFastModePlus
		LL_I2C_IsEnabledFastModePlus
	GCEN	LL_I2C_DisableGeneralCall

Register	Field	Function
CR1	GCEN	LL_I2C_EnableGeneralCall
		LL_I2C_IsEnabledGeneralCall
	NACKIE	LL_I2C_DisableIT_NACK
		LL_I2C_EnableIT_NACK
		LL_I2C_IsEnabledIT_NACK
	NOSTRETCH	LL_I2C_DisableClockStretching
		LL_I2C_EnableClockStretching
		LL_I2C_IsEnabledClockStretching
	PE	LL_I2C_Disable
		LL_I2C_Enable
		LL_I2C_IsEnabled
	PECEN	LL_I2C_DisableSMBusPEC
		LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	RXDMAEN	LL_I2C_DisableDMAReq_RX
		LL_I2C_EnableDMAReq_RX
		LL_I2C_IsEnabledDMAReq_RX
	RXIE	LL_I2C_DisableIT_RX
		LL_I2C_EnableIT_RX
		LL_I2C_IsEnabledIT_RX
	SBC	LL_I2C_DisableSlaveByteControl
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	LL_I2C_GetMode
		LL_I2C_SetMode
	SMBHEN	LL_I2C_GetMode
		LL_I2C_SetMode
	STOPFACLR	LL_I2C_DisableAutoClearFlag_STOP
		LL_I2C_EnableAutoClearFlag_STOP
		LL_I2C_IsEnabledAutoClearFlag_STOP
	STOPIE	LL_I2C_DisableIT_STOP
		LL_I2C_EnableIT_STOP
		LL_I2C_IsEnabledIT_STOP
	TCIE	LL_I2C_DisableIT_TC
		LL_I2C_EnableIT_TC
		LL_I2C_IsEnabledIT_TC
	TXDMAEN	LL_I2C_DisableDMAReq_TX
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_TX
	TXIE	LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_TX

Register	Field	Function
CR1	TXIE	LL_I2C_IsEnabledIT_TX
	WUPEN	LL_I2C_DisableWakeUpFromStop
		LL_I2C_EnableWakeUpFromStop
		LL_I2C_IsEnabledWakeUpFromStop
CR2	ADD10	LL_I2C_GetMasterAddressingMode
		LL_I2C_HandleTransfer
		LL_I2C_SetMasterAddressingMode
	AUTOEND	LL_I2C_DisableAutoEndMode
		LL_I2C_EnableAutoEndMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
	HEAD10R	LL_I2C_DisableAuto10BitRead
		LL_I2C_EnableAuto10BitRead
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAuto10BitRead
	NACK	LL_I2C_AcknowledgeNextData
	NBYTES	LL_I2C_GetTransferSize
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferSize
	PECBYTE	LL_I2C_EnableSMBusPECCompare
		LL_I2C_IsEnabledSMBusPECCompare
	RD_WRN	LL_I2C_GetTransferRequest
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferRequest
	RELOAD	LL_I2C_DisableReloadMode
		LL_I2C_EnableReloadMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledReloadMode
	SADD	LL_I2C_GetSlaveAddr
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	LL_I2C_GenerateStartCondition
		LL_I2C_HandleTransfer
	STOP	LL_I2C_GenerateStopCondition
		LL_I2C_HandleTransfer
ICR	ADDRCF	LL_I2C_ClearFlag_ADDR
	ALERTCF	LL_I2C_ClearSMBusFlag_ALERT
	ARLOCF	LL_I2C_ClearFlag_ARLO
	BERRCF	LL_I2C_ClearFlag_BERR
	NACKCF	LL_I2C_ClearFlag_NACK
	OVRCF	LL_I2C_ClearFlag_OVR

Register	Field	Function
ICR	PECCF	LL_I2C_ClearSMBusFlag_PECERR
	STOPCF	LL_I2C_ClearFlag_STOP
	TIMOUTCF	LL_I2C_ClearSMBusFlag_TIMEOUT
ISR	ADDCODE	LL_I2C_GetAddressMatchCode
	ADDR	LL_I2C_IsActiveFlag_ADDR
	ALERT	LL_I2C_IsActiveSMBusFlag_ALERT
	ARLO	LL_I2C_IsActiveFlag_ARLO
	BERR	LL_I2C_IsActiveFlag_BERR
	BUSY	LL_I2C_IsActiveFlag_BUSY
	DIR	LL_I2C_GetTransferDirection
	NACKF	LL_I2C_IsActiveFlag_NACK
	OVR	LL_I2C_IsActiveFlag_OVR
	PECERR	LL_I2C_IsActiveSMBusFlag_PECERR
	RXNE	LL_I2C_IsActiveFlag_RXNE
	STOPF	LL_I2C_IsActiveFlag_STOP
	TC	LL_I2C_IsActiveFlag_TC
	TCR	LL_I2C_IsActiveFlag_TCR
	TIMEOUT	LL_I2C_IsActiveSMBusFlag_TIMEOUT
	TXE	LL_I2C_ClearFlag_TXE
		LL_I2C_IsActiveFlag_TXE
	TXIS	LL_I2C_IsActiveFlag_TXIS
OAR1	OA1	LL_I2C_SetOwnAddress1
	OA1EN	LL_I2C_DisableOwnAddress1
		LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
	OA1MODE	LL_I2C_SetOwnAddress1
OAR2	OA2	LL_I2C_SetOwnAddress2
	OA2EN	LL_I2C_DisableOwnAddress2
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
	OA2MSK	LL_I2C_SetOwnAddress2
PECR	PEC	LL_I2C_GetSMBusPEC
RXDR	RXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_ReceiveData8
TIMEOUTR	TEXTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
	TIDLE	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutAMode
		LL_I2C_SetSMBusTimeoutAMode
	TIMEOUTA	LL_I2C_ConfigSMBusTimeout

Register	Field	Function
TIMEOUTR	TIMEOUTA	LL_I2C_GetSMBusTimeoutA
		LL_I2C_SetSMBusTimeoutA
	TIMEOUTB	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutB
		LL_I2C_SetSMBusTimeoutB
	TIMOUTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
TIMINGR	PRESC	LL_I2C_GetTimingPrescaler
	SCLDEL	LL_I2C_GetDataSetupTime
	SCLH	LL_I2C_GetClockHighPeriod
	SCLL	LL_I2C_GetClockLowPeriod
	SDADEL	LL_I2C_GetDataHoldTime
	TIMINGR	LL_I2C_SetTiming
TXDR	TXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_TransmitData8

119.15 I2S

Table 39. Correspondence between I2S registers and I2S low-layer driver functions

Register	Field	Function
CFG1	FTHLV	LL_I2S_GetFIFOThreshold
		LL_I2S_SetFIFOThreshold
	RXDMAEN	LL_I2S_DisableDMAReq_RX
		LL_I2S_EnableDMAReq_RX
		LL_I2S_IsEnabledDMAReq_RX
	TXDMAEN	LL_I2S_DisableDMAReq_TX
		LL_I2S_EnableDMAReq_TX
		LL_I2S_IsEnabledDMAReq_TX
CFG2	AFCNTR	LL_I2S_DisableGPIOControl
		LL_I2S_EnableGPIOControl
		LL_I2S_IsEnabledGPIOControl
	IOSWP	LL_I2S_DisableIOSwap
		LL_I2S_EnableIOSwap
		LL_I2S_IsEnabledIOSwap
	LSBFRST	LL_I2S_GetTransferBitOrder
		LL_I2S_SetTransferBitOrder
CR1	CSTART	LL_I2S_IsActiveTransfer
		LL_I2S_StartTransfer
	IOLOCK	LL_I2S_IsEnabledIOLock
		Section 100.2.1 LL_SPI_EnableIOLock

Register	Field	Function
CR1	SPE	LL_I2S_Disable
		LL_I2S_Enable
I2SCFGR	CHLEN	LL_I2S_GetDataFormat
		LL_I2S_SetDataFormat
	CKPOL	LL_I2S_GetClockPolarity
		LL_I2S_SetClockPolarity
	DATFMT	LL_I2S_GetDataFormat
		LL_I2S_SetDataFormat
	DATLEN	LL_I2S_GetDataFormat
		LL_I2S_SetDataFormat
	FIXCH	LL_I2S_GetChannelLengthType
		LL_I2S_SetChannelLengthType
	I2SCFG	LL_I2S_GetTransferMode
		LL_I2S_SetTransferMode
	I2SDIV	LL_I2S_GetPrescalerLinear
		LL_I2S_SetPrescalerLinear
	I2SMOD	LL_I2S_Disable
		LL_I2S_Enable
	I2SSTD	LL_I2S_GetStandard
		LL_I2S_SetStandard
	MCKOE	LL_I2S_DisableMasterClock
		LL_I2S_EnableMasterClock
		LL_I2S_IsEnabledMasterClock
	ODD	LL_I2S_GetPrescalerParity
		LL_I2S_SetPrescalerParity
	PCMSYNC	LL_I2S_GetStandard
		LL_I2S_SetStandard
	WSINV	LL_I2S_DisableWordSelectInversion
		LL_I2S_EnableWordSelectInversion
		LL_I2S_IsEnabledWordSelectInversion
IER	OVRIE	LL_I2S_DisableIT_OVR
		LL_I2S_EnableIT_OVR
		LL_I2S_IsEnabledIT_OVR
	RXPIE	LL_I2S_DisableIT_RXP
		LL_I2S_EnableIT_RXP
		LL_I2S_IsEnabledIT_RXP
	TIFREIE	LL_I2S_DisableIT_FRE
		LL_I2S_EnableIT_FRE
		LL_I2S_IsEnabledIT_FRE
	TXPIE	LL_I2S_DisableIT_TXP
		LL_I2S_EnableIT_TXP

Register	Field	Function
IER	TXPIE	LL_I2S_IsEnabledIT_TXP
	UDRIE	LL_I2S_DisableIT_UDR
		LL_I2S_EnableIT_UDR
		LL_I2S_IsEnabledIT_UDR
IFCR	OVRC	LL_I2S_ClearFlag_OVR
	TIFREC	LL_I2S_ClearFlag_FRE
	UDRC	LL_I2S_ClearFlag_UDR
RXDR	.	LL_I2S_ReceiveData16
		LL_I2S_ReceiveData32
SR	OVR	LL_I2S_IsActiveFlag_OVR
	RXP	LL_I2S_IsActiveFlag_RXP
	TIFRE	LL_I2S_IsActiveFlag_FRE
	TXP	LL_I2S_IsActiveFlag_TXP
	UDR	LL_I2S_IsActiveFlag_UDR
TXDR	.	LL_I2S_TransmitData16
		LL_I2S_TransmitData32

119.16 I3C

Table 40. Correspondence between I3C registers and I3C low-layer driver functions

Register	Field	Function
BCR	BCR0	LL_I3C_GetDataSpeedLimitation
		LL_I3C_SetDataSpeedLimitation
	BCR2	LL_I3C_GetDeviceIBIPayload
		LL_I3C_SetDeviceIBIPayload
	BCR6	LL_I3C_GetDeviceCapabilityOnBus
		LL_I3C_SetDeviceCapabilityOnBus
CEVR	CASUPDF	LL_I3C_ClearFlag_ASUPD
	CCRF	LL_I3C_ClearFlag_CR
	CCRUPDF	LL_I3C_ClearFlag_CRUPD
	CDAUPDF	LL_I3C_ClearFlag_DAUPD
	CDEFF	LL_I3C_ClearFlag_DEF
	CERRF	LL_I3C_ClearFlag_ERR
	CFCF	LL_I3C_ClearFlag_FC
	CGETF	LL_I3C_ClearFlag_GET
	CGRPF	LL_I3C_ClearFlag_GRP
	CHJF	LL_I3C_ClearFlag_HJ
	CIBIENDF	LL_I3C_ClearFlag_IBIEND
	CIBIF	LL_I3C_ClearFlag_IBI
	CINTUPDF	LL_I3C_ClearFlag_INTUPD
	CMRLUPDF	LL_I3C_ClearFlag_MRLUPD

Register	Field	Function
CEVR	CMWLUPDF	LL_I3C_ClearFlag_MWLUPD
	CRSTF	LL_I3C_ClearFlag_RST
	CRXTGTENDF	LL_I3C_ClearFlag_RXTGTEND
	CSTAF	LL_I3C_ClearFlag_STA
	CWKPF	LL_I3C_ClearFlag_WKP
CFGR	CDMAEN	LL_I3C_DisableDMAReq_Control
		LL_I3C_EnableDMAReq_Control
		LL_I3C_IsEnabledDMAReq_Control
	CFLUSH	LL_I3C_RequestControlFIFOFlush
	CRINIT	LL_I3C_GetMode
		LL_I3C_SetMode
	EN	LL_I3C_Disable
		LL_I3C_Enable
		LL_I3C_IsEnabled
	EXITPTRN	LL_I3C_DisableExitPattern
		LL_I3C_EnableExitPattern
		LL_I3C_IsEnabledExitPattern
	HJACK	LL_I3C_DisableHJAck
		LL_I3C_EnableHJAck
		LL_I3C_IsEnabledHJAck
	HKSDAEN	LL_I3C_DisableHighKeeperSDA
		LL_I3C_EnableHighKeeperSDA
		LL_I3C_IsEnabledHighKeeperSDA
	NOARBH	LL_I3C_DisableArbitrationHeader
		LL_I3C_EnableArbitrationHeader
		LL_I3C_IsEnabledArbitrationHeader
	RSTPTRN	LL_I3C_DisableResetPattern
		LL_I3C_EnableResetPattern
		LL_I3C_IsEnabledResetPattern
	RXDMAEN	LL_I3C_DisableDMAReq_RX
		LL_I3C_EnableDMAReq_RX
		LL_I3C_IsEnabledDMAReq_RX
	RXFLUSH	LL_I3C_RequestRxFIFOFlush
	RXTHRES	LL_I3C_GetRxFIFOThreshold
		LL_I3C_SetRxFIFOThreshold
	SDMAEN	LL_I3C_DisableDMAReq_Status
		LL_I3C_EnableDMAReq_Status
		LL_I3C_IsEnabledDMAReq_Status
	SFLUSH	LL_I3C_RequestStatusFIFOFlush
	SMODE	LL_I3C_DisableStatusFIFO
		LL_I3C_EnableStatusFIFO

Register	Field	Function
CFGR	SMODE	LL_I3C_IsEnabledStatusFIFO
	TMODE	LL_I3C_DisableControlFIFO
		LL_I3C_EnableControlFIFO
		LL_I3C_IsEnabledControlFIFO
	TSFSET	LL_I3C_RequestTransfer
	TXDMAEN	LL_I3C_DisableDMAReq_TX
		LL_I3C_EnableDMAReq_TX
		LL_I3C_IsEnabledDMAReq_TX
	TXFLUSH	LL_I3C_RequestTxFIFOFlush
	TXTHRES	LL_I3C_GetTxFIFOThreshold
		LL_I3C_SetTxFIFOThreshold
CR	ADD	LL_I3C_ControllerHandleMessage
	CCC	LL_I3C_ControllerHandleCCC
	CR	LL_I3C_DMA_GetRegAddr
	DCNT	LL_I3C_ControllerHandleCCC
		LL_I3C_ControllerHandleMessage
		LL_I3C_TargetHandleMessage
	MEND	LL_I3C_ControllerHandleCCC
		LL_I3C_ControllerHandleMessage
	MTYPE	LL_I3C_ControllerHandleCCC
		LL_I3C_ControllerHandleMessage
		LL_I3C_TargetHandleMessage
	RNW	LL_I3C_ControllerHandleMessage
CRCAPR	CAPDHOFF	LL_I3C_GetControllerHandoffDelayed
		LL_I3C_SetControllerHandoffDelayed
	CAPGRP	LL_I3C_GetGrpAddrHandoffSupport
		LL_I3C_SetGrpAddrHandoffSupport
DCR	DC	LL_I3C_SetDeviceCharacteristics
	DCR	LL_I3C_GetDeviceCharacteristics
DEVR0	AS	LL_I3C_GetActivityState
	CREN	LL_I3C_DisableControllerRoleReq
		LL_I3C_EnableControllerRoleReq
		LL_I3C_IsEnabledControllerRoleReq
	DA	LL_I3C_GetOwnDynamicAddress
		LL_I3C_SetOwnDynamicAddress
	DAVAL	LL_I3C_DisableOwnDynAddress
		LL_I3C_EnableOwnDynAddress
		LL_I3C_IsEnabledOwnDynAddress
	HJEN	LL_I3C_DisableHotJoin
		LL_I3C_EnableHotJoin
		LL_I3C_IsEnabledHotJoin

Register	Field	Function
DEVR0	IBIEN	LL_I3C_DisableIBI
		LL_I3C_EnableIBI
		LL_I3C_IsEnabledIBI
	RSTACT	LL_I3C_GetResetAction
	RSTVAL	LL_I3C_IsEnabledReset
DEVRX	CRACK	LL_I3C_ConfigDeviceCapabilities
		LL_I3C_DisableTargetCRAck
		LL_I3C_EnableTargetCRAck
		LL_I3C_IsEnabledTargetCRAck
	DA	LL_I3C_ConfigDeviceCapabilities
		LL_I3C_GetTargetDynamicAddress
		LL_I3C_SetTargetDynamicAddress
	DIS	LL_I3C_IsAllowedPayloadUpdate
	IBIACK	LL_I3C_ConfigDeviceCapabilities
		LL_I3C_DisableTargetIBIAck
		LL_I3C_EnableTargetIBIAck
		LL_I3C_IsEnabledTargetIBIAck
	IBIDEN	LL_I3C_ConfigDeviceCapabilities
		LL_I3C_DisableIBIAddData
		LL_I3C_EnableIBIAddData
		LL_I3C_IsEnabledIBIAddData
	SUSP	LL_I3C_DisableFrameSuspend
		LL_I3C_EnableFrameSuspend
		LL_I3C_IsFrameMustBeSuspended
EPIDR	IDTSEL	LL_I3C_GetIDTypeSelector
	MIPIID	LL_I3C_GetMIPIInstanceID
		LL_I3C_SetMIPIInstanceID
	MIPIMID	LL_I3C_GetMIPIManufacturerID
EVR	ASUPDF	LL_I3C_IsActiveFlag_ASUPD
	BCUPDF	LL_I3C_IsActiveFlag_CRUPD
	CFEF	LL_I3C_IsActiveFlag_CFE
	CFNFF	LL_I3C_IsActiveFlag_CFNF
	CRF	LL_I3C_IsActiveFlag_CR
	DAUPDF	LL_I3C_IsActiveFlag_DAUPD
	DEFF	LL_I3C_IsActiveFlag_DEF
	ERRF	LL_I3C_IsActiveFlag_ERR
	FCF	LL_I3C_IsActiveFlag_FC
	GETF	LL_I3C_IsActiveFlag_GET
	GRPF	LL_I3C_IsActiveFlag_GRP
	HJF	LL_I3C_IsActiveFlag_HJ
	IBIENDF	LL_I3C_IsActiveFlag_IBIEND

Register	Field	Function
EVR	IBIF	LL_I3C_IsActiveFlag_IBI
	INTUPDF	LL_I3C_IsActiveFlag_INTUPD
	MRLUPDF	LL_I3C_IsActiveFlag_MRLUPD
	MWLUPDF	LL_I3C_IsActiveFlag_MWLUPD
	RSTF	LL_I3C_IsActiveFlag_RST
	RXFNEF	LL_I3C_IsActiveFlag_RXFNE
	RXLASTF	LL_I3C_IsActiveFlag_RXLAST
	RXTGTENDF	LL_I3C_IsActiveFlag_RXTGTEND
	SFNEF	LL_I3C_IsActiveFlag_SFNE
	STAF	LL_I3C_IsActiveFlag_STA
	TXFEF	LL_I3C_IsActiveFlag_TXFE
	TXFNFF	LL_I3C_IsActiveFlag_TXFNF
	TXLASTF	LL_I3C_IsActiveFlag_TXLAST
	WKPF	LL_I3C_IsActiveFlag_WKP
GETCAPR	CAPPEND	LL_I3C_GetPendingReadMDB
		LL_I3C_SetPendingReadMDB
GETMXDSR	FMT	LL_I3C_GetMaxDataSpeedFormat
		LL_I3C_SetMaxDataSpeedFormat
	HOFFAS	LL_I3C_GetHandoffActivityState
		LL_I3C_SetHandoffActivityState
	RDTURN	LL_I3C_GetMiddleByteTurnAround
		LL_I3C_SetMiddleByteTurnAround
	TSCO	LL_I3C_GetDataTurnAroundTime
		LL_I3C_SetDataTurnAroundTime
IBIDR	IBIDR	LL_I3C_GetIBIPayload
		LL_I3C_SetIBIPayload
IER	ASUPDIE	LL_I3C_DisableIT_ASUPD
		LL_I3C_EnableIT_ASUPD
		LL_I3C_IsEnabledIT_ASUPD
	CFNFIE	LL_I3C_DisableIT_CFNF
		LL_I3C_EnableIT_CFNF
		LL_I3C_IsEnabledIT_CFNF
	CRIE	LL_I3C_DisableIT_CR
		LL_I3C_EnableIT_CR
		LL_I3C_IsEnabledIT_CR
	CRUPDIE	LL_I3C_DisableIT_CRUPD
		LL_I3C_EnableIT_CRUPD
		LL_I3C_IsEnabledIT_CRUPD
	DAUPDIE	LL_I3C_DisableIT_DAUPD
		LL_I3C_EnableIT_DAUPD
		LL_I3C_IsEnabledIT_DAUPD

Register	Field	Function
IER	DEFIE	LL_I3C_DisableIT_DEF
		LL_I3C_EnableIT_DEF
		LL_I3C_IsEnabledIT_DEF
	ERRIE	LL_I3C_DisableIT_ERR
		LL_I3C_EnableIT_ERR
		LL_I3C_IsEnabledIT_ERR
	FCIE	LL_I3C_DisableIT_FC
		LL_I3C_EnableIT_FC
		LL_I3C_IsEnabledIT_FC
	GETIE	LL_I3C_DisableIT_GET
		LL_I3C_EnableIT_GET
		LL_I3C_IsEnabledIT_GET
	GRPIE	LL_I3C_DisableIT_GRP
		LL_I3C_EnableIT_GRP
		LL_I3C_IsEnabledIT_GRP
	HJIE	LL_I3C_DisableIT_HJ
		LL_I3C_EnableIT_HJ
		LL_I3C_IsEnabledIT_HJ
	IBIENDIE	LL_I3C_DisableIT_IBIEND
		LL_I3C_EnableIT_IBIEND
		LL_I3C_IsEnabledIT_IBIEND
	IBIIE	LL_I3C_DisableIT_IBI
		LL_I3C_EnableIT_IBI
		LL_I3C_IsEnabledIT_IBI
	INTUPDIE	LL_I3C_DisableIT_INTUPD
		LL_I3C_EnableIT_INTUPD
		LL_I3C_IsEnabledIT_INTUPD
	MRLUPDIE	LL_I3C_DisableIT_MRLUPD
		LL_I3C_EnableIT_MRLUPD
		LL_I3C_IsEnabledIT_MRLUPD
	MWLUPDIE	LL_I3C_DisableIT_MWLUPD
		LL_I3C_EnableIT_MWLUPD
		LL_I3C_IsEnabledIT_MWLUPD
	RSTIE	LL_I3C_DisableIT_RST
		LL_I3C_EnableIT_RST
		LL_I3C_IsEnabledIT_RST
	RXFNEIE	LL_I3C_DisableIT_RXFNE
		LL_I3C_EnableIT_RXFNE
		LL_I3C_IsEnabledIT_RXFNE
	RXTGTENDIE	LL_I3C_DisableIT_RXTGTEND
		LL_I3C_EnableIT_RXTGTEND

Register	Field	Function
IER	RXTGTENDIE	LL_I3C_IsEnabledIT_RXTGTEND
	SFNEIE	LL_I3C_DisableIT_SFNE
		LL_I3C_EnableIT_SFNE
		LL_I3C_IsEnabledIT_SFNE
	STAIE	LL_I3C_DisableIT_STA
		LL_I3C_EnableIT_STA
		LL_I3C_IsEnabledIT_STA
	TXFNFIE	LL_I3C_DisableIT_TXFNF
		LL_I3C_EnableIT_TXFNF
		LL_I3C_IsEnabledIT_TXFNF
	WKPIE	LL_I3C_DisableIT_WKP
		LL_I3C_EnableIT_WKP
		LL_I3C_IsEnabledIT_WKP
MAXRLR	IBIP	LL_I3C_ConfigNbIBIAddData
		LL_I3C_GetConfigNbIBIAddData
	MRL	LL_I3C_GetMaxReadLength
		LL_I3C_SetMaxReadLength
MAXWLR	MWL	LL_I3C_GetMaxWriteLength
		LL_I3C_SetMaxWriteLength
RDR	RDB0	LL_I3C_ReceiveData8
	RXRB0	LL_I3C_DMA_GetRegAddr
RDWR	RDWR	LL_I3C_DMA_GetRegAddr
		LL_I3C_ReceiveData32
RMR	IBIRDCNT	LL_I3C_GetNbIBIAddData
	RADD	LL_I3C_GetIBITargetAddr
	RCODE	LL_I3C_GetReceiveCommandCode
SER	ANACK	LL_I3C_IsActiveFlag_ANACK
	CODERR	LL_I3C_GetMessageErrorCode
	COVR	LL_I3C_IsActiveFlag_COVR
	DERR	LL_I3C_IsActiveFlag_DERR
	DNACK	LL_I3C_IsActiveFlag_DNACK
	DOVR	LL_I3C_IsActiveFlag_DOVR
	PERR	LL_I3C_IsActiveFlag_PERR
	STALL	LL_I3C_IsActiveFlag_STALL
SR	ABT	LL_I3C_IsTargetAbortPrivateRead
	DIR	LL_I3C_GetMessageDirection
	MID	LL_I3C_GetMessageIdentifier
	SR	LL_I3C_DMA_GetRegAddr
	XDCNT	LL_I3C_GetXferDataCount
TDR	TDB0	LL_I3C_DMA_GetRegAddr
		LL_I3C_TransmitData8

Register	Field	Function
TDWR	TDWR	LL_I3C_DMA_GetRegAddr
		LL_I3C_TransmitData32
TGTTDR	PRELOAD	LL_I3C_ConfigTxPreload
		LL_I3C_IsActiveTxPreload
	TDCNT	LL_I3C_ConfigTxPreload
		LL_I3C_GetTxPreloadDataCount
TIMINGR0	SCLH_I2C	LL_I3C_GetPeriodClockHighI2C
		LL_I3C_SetPeriodClockHighI2C
	SCLH_I3C	LL_I3C_GetPeriodClockHighI3C
		LL_I3C_SetPeriodClockHighI3C
	SCLL_OD	LL_I3C_GetPeriodClockLowOD
		LL_I3C_SetPeriodClockLowOD
	SCLL_PP	LL_I3C_GetPeriodClockLowPP
		LL_I3C_SetPeriodClockLowPP
	TIMINGR0	LL_I3C_ConfigClockWaveForm
		LL_I3C_GetClockWaveForm
TIMINGR1	ASNCR	LL_I3C_GetControllerActivityState
		LL_I3C_GetCtrlBusCharacteristic
		LL_I3C_SetControllerActivityState
		LL_I3C_SetCtrlBusCharacteristic
	AVAL	LL_I3C_GetAvalTiming
		LL_I3C_SetAvalTiming
	FREE	LL_I3C_GetCtrlBusCharacteristic
		LL_I3C_GetFreeTiming
		LL_I3C_SetCtrlBusCharacteristic
		LL_I3C_SetFreeTiming
	IDLE	LL_I3C_GetCtrlBusCharacteristic
		LL_I3C_GetTgtBusCharacteristic
		LL_I3C_SetCtrlBusCharacteristic
		LL_I3C_SetTgtBusCharacteristic
	SDA_HD	LL_I3C_GetCtrlBusCharacteristic
		LL_I3C_GetDataHoldTime
		LL_I3C_SetCtrlBusCharacteristic
		LL_I3C_SetDataHoldTime
TIMINGR2	STALL	LL_I3C_GetStallTime
		LL_I3C_SetStallTime
	STALLA	LL_I3C_DisableStallACK
		LL_I3C_EnableStallACK
		LL_I3C_IsEnabledStallACK
	STALLC	LL_I3C_DisableStallParityCCC
		LL_I3C_EnableStallParityCCC

Register	Field	Function
TIMINGR2	STALLC	LL_I3C_IsEnabledStallParityCCC
	STALLD	LL_I3C_DisableStallParityData
		LL_I3C_EnableStallParityData
		LL_I3C_IsEnabledStallParityData
	STALLT	LL_I3C_DisableStallTbit
		LL_I3C_EnableStallTbit
		LL_I3C_IsEnabledStallTbit

119.17 ICACHE

Table 41. Correspondence between ICACHE registers and ICACHE low-layer driver functions

Register	Field	Function
CR	CACHEINV	LL_ICACHE_Invalidate
	EN	LL_ICACHE_Disable
		LL_ICACHE_Enable
		LL_ICACHE_IsEnabled
	HITMEN	LL_ICACHE_DisableMonitors
		LL_ICACHE_EnableMonitors
		LL_ICACHE_IsEnabledMonitors
	HITMRST	LL_ICACHE_ResetMonitors
	MISSMEN	LL_ICACHE_DisableMonitors
		LL_ICACHE_EnableMonitors
		LL_ICACHE_IsEnabledMonitors
	MISSMRST	LL_ICACHE_ResetMonitors
	WAYSEL	LL_ICACHE_GetMode
		LL_ICACHE_SetMode
CRRx	BASEADDR	LL_ICACHE_GetRegionBaseAddress
		LL_ICACHE_SetRegionBaseAddress
	HBURST	LL_ICACHE_GetRegionOutputBurstType
		LL_ICACHE_SetRegionOutputBurstType
	MSTSEL	LL_ICACHE_GetRegionMasterPort
		LL_ICACHE_SetRegionMasterPort
	REMAPADDR	LL_ICACHE_GetRegionRemapAddress
		LL_ICACHE_SetRegionRemapAddress
	REN	LL_ICACHE_DisableRegion
		LL_ICACHE_EnableRegion
		LL_ICACHE_IsEnabledRegion
	RSIZE	LL_ICACHE_GetRegionSize
		LL_ICACHE_SetRegionSize
FCR	CBSYENDF	LL_ICACHE_ClearFlag_BSYEND
	ERRF	LL_ICACHE_ClearFlag_ERR

Register	Field	Function
HMONR	HITMON	LL_ICACHE_GetHitMonitor
IER	BSYENDIE	LL_ICACHE_DisableIT_BSYEND
		LL_ICACHE_EnableIT_BSYEND
		LL_ICACHE_IsEnabledIT_BSYEND
	ERRIE	LL_ICACHE_DisableIT_ERR
		LL_ICACHE_EnableIT_ERR
		LL_ICACHE_IsEnabledIT_ERR
MMONR	MISSMON	LL_ICACHE_GetMissMonitor
SR	BSYEND	LL_ICACHE_IsActiveFlag_BSYEND
	BUSYF	LL_ICACHE_IsActiveFlag_BUSY
	ERRF	LL_ICACHE_IsActiveFlag_ERR

119.18 IWDG

Table 42. Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
EWCR	EWIC	LL_IWDG_ClearFlag_EWIF
	EWIE	LL_IWDG_DisableIT_EWI
		LL_IWDG_EnableIT_EWI
		LL_IWDG_IsEnabledIT_EWI
	EWIT	LL_IWDG_GetEwiTime
		LL_IWDG_SetEwiTime
KR	KEY	LL_IWDG_DisableWriteAccess
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess
		LL_IWDG_ReloadCounter
PR	PR	LL_IWDG_GetPrescaler
		LL_IWDG_SetPrescaler
RLR	RL	LL_IWDG_GetReloadCounter
		LL_IWDG_SetReloadCounter
SR	EVU	LL_IWDG_IsActiveFlag_EWU
	EWIF	LL_IWDG_IsActiveFlag_EWIF
	EWU	LL_IWDG_IsReady
	ONF	LL_IWDG_IsActiveFlag_ONF
	PVU	LL_IWDG_IsActiveFlag_PVU
		LL_IWDG_IsReady
	RVU	LL_IWDG_IsActiveFlag_RVU
		LL_IWDG_IsReady
	WVU	LL_IWDG_IsActiveFlag_WVU
		LL_IWDG_IsReady
WINR	WIN	LL_IWDG_GetWindow

Register	Field	Function
WINR	WIN	LL_IWDG_SetWindow

119.19 LPTIM

Table 43. Correspondence between LPTIM registers and LPTIM low-layer driver functions

Register	Field	Function
ARR	ARR	LL_LPTIM_GetAutoReload
		LL_LPTIM_SetAutoReload
CCMR1	CC1E	LL_LPTIM_CC_DisableChannel
		LL_LPTIM_CC_EnableChannel
		LL_LPTIM_CC_IsEnabledChannel
	CC1P	LL_LPTIM_IC_GetPolarity
		LL_LPTIM_IC_SetPolarity
		LL_LPTIM_OC_GetPolarity
		LL_LPTIM_OC_SetPolarity
	CC1SEL	LL_LPTIM_CC_GetChannelMode
		LL_LPTIM_CC_SetChannelMode
	CC2E	LL_LPTIM_CC_DisableChannel
		LL_LPTIM_CC_EnableChannel
		LL_LPTIM_CC_IsEnabledChannel
	CC2P	LL_LPTIM_IC_GetPolarity
		LL_LPTIM_IC_SetPolarity
		LL_LPTIM_OC_GetPolarity
		LL_LPTIM_OC_SetPolarity
	CC2SEL	LL_LPTIM_CC_GetChannelMode
		LL_LPTIM_CC_SetChannelMode
	IC1F	LL_LPTIM_IC_GetFilter
		LL_LPTIM_IC_SetFilter
	IC1PSC	LL_LPTIM_IC_GetPrescaler
		LL_LPTIM_IC_SetPrescaler
	IC2F	LL_LPTIM_IC_GetFilter
		LL_LPTIM_IC_SetFilter
	IC2PSC	LL_LPTIM_IC_GetPrescaler
		LL_LPTIM_IC_SetPrescaler
CCR1	CCR1	LL_LPTIM_IC_GetCaptureCH1
		LL_LPTIM_OC_GetCompareCH1
		LL_LPTIM_OC_SetCompareCH1
CCR2	CCR2	LL_LPTIM_IC_GetCaptureCH2
		LL_LPTIM_OC_GetCompareCH2
		LL_LPTIM_OC_SetCompareCH2
CFGR	CKFLT	LL_LPTIM_ConfigClock

Register	Field	Function
CFGR	CKFLT	LL_LPTIM_GetClockFilter
	CKPOL	LL_LPTIM_ConfigClock
		LL_LPTIM_GetClockPolarity
		LL_LPTIM_GetEncoderMode
		LL_LPTIM_SetEncoderMode
	CKSEL	LL_LPTIM_GetClockSource
		LL_LPTIM_SetClockSource
	COUNTMODE	LL_LPTIM_GetCounterMode
		LL_LPTIM_SetCounterMode
	ENC	LL_LPTIM_DisableEncoderMode
		LL_LPTIM_EnableEncoderMode
		LL_LPTIM_IsEnabledEncoderMode
	PRELOAD	LL_LPTIM_GetUpdateMode
		LL_LPTIM_SetUpdateMode
	PRESC	LL_LPTIM_GetPrescaler
		LL_LPTIM_SetPrescaler
	TIMOUT	LL_LPTIM_DisableTimeout
		LL_LPTIM_EnableTimeout
		LL_LPTIM_IsEnabledTimeout
	TRGFLT	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerFilter
	TRIGEN	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerPolarity
		LL_LPTIM_TrigSw
	TRIGSEL	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerSource
	WAVE	LL_LPTIM_GetWaveform
		LL_LPTIM_SetWaveform
CFGR2	IC1SEL	LL_LPTIM_SetRemap
	IC2SEL	LL_LPTIM_SetRemap
	IN1SEL	LL_LPTIM_SetInput1Src
	IN2SEL	LL_LPTIM_SetInput2Src
CNT	CNT	LL_LPTIM_GetCounter
CR	CNTSTRT	LL_LPTIM_StartCounter
	COUNTRST	LL_LPTIM_ResetCounter
	ENABLE	LL_LPTIM_Disable
		LL_LPTIM_Enable
		LL_LPTIM_IsEnabled
	RSTARE	LL_LPTIM_DisableResetAfterRead
		LL_LPTIM_EnableResetAfterRead
		LL_LPTIM_IsEnabledResetAfterRead

Register	Field	Function
CR	SNGSTRT	LL_LPTIM_StartCounter
DIER	ARRMIE	LL_LPTIM_DisableIT_ARRM
		LL_LPTIM_EnableIT_ARRM
		LL_LPTIM_IsEnabledIT_ARRM
	ARROKIE	LL_LPTIM_DisableIT_ARROK
		LL_LPTIM_EnableIT_ARROK
		LL_LPTIM_IsEnabledIT_ARROK
	CC1IE	LL_LPTIM_DisableIT_CC1
		LL_LPTIM_EnableIT_CC1
		LL_LPTIM_IsEnabledIT_CC1
	CC1OIE	LL_LPTIM_DisableIT_CC1O
		LL_LPTIM_EnableIT_CC1O
		LL_LPTIM_IsEnabledIT_CC1O
	CC2IE	LL_LPTIM_DisableIT_CC2
		LL_LPTIM_EnableIT_CC2
		LL_LPTIM_IsEnabledIT_CC2
	CC2OIE	LL_LPTIM_DisableIT_CC2O
		LL_LPTIM_EnableIT_CC2O
		LL_LPTIM_IsEnabledIT_CC2O
	DOWNIE	LL_LPTIM_DisableIT_DOWN
		LL_LPTIM_EnableIT_DOWN
		LL_LPTIM_IsEnabledIT_DOWN
	EXTTRIGIE	LL_LPTIM_DisableIT_EXTTRIG
		LL_LPTIM_EnableIT_EXTTRIG
		LL_LPTIM_IsEnabledIT_EXTTRIG
	REPOKIE	LL_LPTIM_DisableIT_REPOK
		LL_LPTIM_EnableIT_REPOK
		LL_LPTIM_IsEnabledIT_REPOK
	UEIE	LL_LPTIM_DisableIT_UE
		LL_LPTIM_EnableIT_UE
		LL_LPTIM_IsEnabledIT_UE
	UPIE	LL_LPTIM_DisableIT_UP
		LL_LPTIM_EnableIT_UP
		LL_LPTIM_IsEnabledIT_UP
ICR	ARRMCF	LL_LPTIM_ClearFlag_ARRM
	ARROKCF	LL_LPTIM_ClearFlag_ARROK
	CC1CF	LL_LPTIM_ClearFlag_CC1
	CC1OCF	LL_LPTIM_ClearFlag_CC1O
	CC2CF	LL_LPTIM_ClearFlag_CC2
	CC2OCF	LL_LPTIM_ClearFlag_CC2O
	CMP1OKCF	LL_LPTIM_ClearFlag_CMP1OK

Register	Field	Function
ICR	CMP2OKCF	LL_LPTIM_ClearFlag_CMP2OK
	DIEROKCF	LL_LPTIM_ClearFlag_DIEROK
	DOWNCF	LL_LPTIM_ClearFlag_DOWN
	EXTTRIGCF	LL_LPTIM_ClearFlag_EXTTRIG
	REPOKCF	LL_LPTIM_ClearFlag_REPOK
	UECF	LL_LPTIM_ClearFlag_UE
	UPCF	LL_LPTIM_ClearFlag_UP
IER	CMP1OKIE	LL_LPTIM_EnableIT_CMP1OK
		LL_LPTIM_IsEnabledIT_CMP1OK
	CMP2OKIE	LL_LPTIM_DisableIT_CMP2OK
		LL_LPTIM_EnableIT_CMP2OK
		LL_LPTIM_IsEnabledIT_CMP2OK
	CMPO1KIE	LL_LPTIM_DisableIT_CMP1OK
ISR	ARRM	LL_LPTIM_IsActiveFlag_ARRM
	ARROK	LL_LPTIM_IsActiveFlag_ARROK
	CC1IF	LL_LPTIM_IsActiveFlag_CC1
	CC1OF	LL_LPTIM_IsActiveFlag_CC1O
	CC2IF	LL_LPTIM_IsActiveFlag_CC2
	CC2OF	LL_LPTIM_IsActiveFlag_CC2O
	CMP1OK	LL_LPTIM_IsActiveFlag_CMP1OK
	CMP2OK	LL_LPTIM_IsActiveFlag_CMP2OK
	DIEROK	LL_LPTIM_IsActiveFlag_DIEROK
	DOWN	LL_LPTIM_IsActiveFlag_DOWN
	EXTTRIG	LL_LPTIM_IsActiveFlag_EXTTRIG
	REPOK	LL_LPTIM_IsActiveFlag_REPOK
	UE	LL_LPTIM_IsActiveFlag_UE
	UP	LL_LPTIM_IsActiveFlag_UP
RCR	REP	LL_LPTIM_GetRepetition
		LL_LPTIM_SetRepetition

119.20 LPUART

Table 44. Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_LPUART_GetBaudRate
		LL_LPUART_SetBaudRate
CR1	CMIE	LL_LPUART_DisableIT_CM
		LL_LPUART_EnableIT_CM
		LL_LPUART_IsEnabledIT_CM
	DEAT	LL_LPUART_GetDEAssertionTime
		LL_LPUART_SetDEAssertionTime

Register	Field	Function
CR1	DEDT	LL_LPUART_GetDEDeassertionTime
		LL_LPUART_SetDEDeassertionTime
	FIFOEN	LL_LPUART_DisableFIFO
		LL_LPUART_EnableFIFO
		LL_LPUART_IsEnabledFIFO
	IDLEIE	LL_LPUART_DisableIT_IDLE
		LL_LPUART_EnableIT_IDLE
		LL_LPUART_IsEnabledIT_IDLE
	M	LL_LPUART_ConfigCharacter
		LL_LPUART_GetDataWidth
		LL_LPUART_SetDataWidth
	MME	LL_LPUART_DisableMuteMode
		LL_LPUART_EnableMuteMode
		LL_LPUART_IsEnabledMuteMode
	PCE	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	PEIE	LL_LPUART_DisableIT_PE
		LL_LPUART_EnableIT_PE
		LL_LPUART_IsEnabledIT_PE
	PS	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	RE	LL_LPUART_DisableDirectionRx
		LL_LPUART_EnableDirectionRx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	RXFFIE	LL_LPUART_DisableIT_RXFF
		LL_LPUART_EnableIT_RXFF
		LL_LPUART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_LPUART_DisableIT_RXNE_RXFNE
		LL_LPUART_EnableIT_RXNE_RXFNE
		LL_LPUART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_LPUART_DisableIT_TC
		LL_LPUART_EnableIT_TC
		LL_LPUART_IsEnabledIT_TC
	TE	LL_LPUART_DisableDirectionTx
		LL_LPUART_EnableDirectionTx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	TXEIE_TXFNFIE	LL_LPUART_DisableIT_TXE_TXFNF

Register	Field	Function
CR1	TXEIE_TXFNFIE	LL_LPUART_EnableIT_TXE_TXFNF
		LL_LPUART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_LPUART_DisableIT_TXFE
		LL_LPUART_EnableIT_TXFE
		LL_LPUART_IsEnabledIT_TXFE
	UE	LL_LPUART_Disable
		LL_LPUART_Enable
		LL_LPUART_IsEnabled
	UESM	LL_LPUART_DisableInStopMode
		LL_LPUART_EnableInStopMode
		LL_LPUART_IsEnabledInStopMode
	WAKE	LL_LPUART_GetWakeUpMethod
		LL_LPUART_SetWakeUpMethod
CR2	ADD	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddress
	ADDM7	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddressLen
	DATAINV	LL_LPUART_GetBinaryDataLogic
		LL_LPUART_SetBinaryDataLogic
	MSBFIRST	LL_LPUART_GetTransferBitOrder
		LL_LPUART_SetTransferBitOrder
	RXINV	LL_LPUART_GetRXPinLevel
		LL_LPUART_SetRXPinLevel
	STOP	LL_LPUART_ConfigCharacter
		LL_LPUART_GetStopBitsLength
		LL_LPUART_SetStopBitsLength
	SWAP	LL_LPUART_GetTXRXSwap
		LL_LPUART_SetTXRXSwap
	TXINV	LL_LPUART_GetTXPinLevel
		LL_LPUART_SetTXPinLevel
CR3	CTSE	LL_LPUART_DisableCTSHWFlowCtrl
		LL_LPUART_EnableCTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	CTSIE	LL_LPUART_DisableIT_CTS
		LL_LPUART_EnableIT_CTS
		LL_LPUART_IsEnabledIT_CTS
	DDRE	LL_LPUART_DisableDMADeactOnRxErr
		LL_LPUART_EnableDMADeactOnRxErr
		LL_LPUART_IsEnabledDMADeactOnRxErr
	DEM	LL_LPUART_DisableDEMode

Register	Field	Function
CR3	DEM	LL_LPUART_EnableDEMode
		LL_LPUART_IsEnabledDEMode
	DEP	LL_LPUART_GetDESignalPolarity
		LL_LPUART_SetDESignalPolarity
	DMAR	LL_LPUART_DisableDMAReq_RX
		LL_LPUART_EnableDMAReq_RX
		LL_LPUART_IsEnabledDMAReq_RX
	DMAT	LL_LPUART_DisableDMAReq_TX
		LL_LPUART_EnableDMAReq_TX
		LL_LPUART_IsEnabledDMAReq_TX
	EIE	LL_LPUART_DisableIT_ERROR
		LL_LPUART_EnableIT_ERROR
		LL_LPUART_IsEnabledIT_ERROR
	HDSEL	LL_LPUART_DisableHalfDuplex
		LL_LPUART_EnableHalfDuplex
		LL_LPUART_IsEnabledHalfDuplex
	OVRDIS	LL_LPUART_DisableOverrunDetect
		LL_LPUART_EnableOverrunDetect
		LL_LPUART_IsEnabledOverrunDetect
	RTSE	LL_LPUART_DisableRTSHWFlowCtrl
		LL_LPUART_EnableRTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	RXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetRXFIFOThreshold
		LL_LPUART_SetRXFIFOThreshold
	RXFTIE	LL_LPUART_DisableIT_RXFT
		LL_LPUART_EnableIT_RXFT
		LL_LPUART_IsEnabledIT_RXFT
	TXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetTXFIFOThreshold
		LL_LPUART_SetTXFIFOThreshold
	TXFTIE	LL_LPUART_DisableIT_TXFT
		LL_LPUART_EnableIT_TXFT
		LL_LPUART_IsEnabledIT_TXFT
	WUFIE	LL_LPUART_DisableIT_WKUP
		LL_LPUART_EnableIT_WKUP
		LL_LPUART_IsEnabledIT_WKUP
	WUS	LL_LPUART_GetWKUPType
		LL_LPUART_SetWKUPType
ICR	CMCF	LL_LPUART_ClearFlag_CM

Register	Field	Function
ICR	CTSCF	LL_LPUART_ClearFlag_nCTS
	FECF	LL_LPUART_ClearFlag_FE
	IDLECF	LL_LPUART_ClearFlag_IDLE
	NECF	LL_LPUART_ClearFlag_NE
	ORECF	LL_LPUART_ClearFlag_ORE
	PECF	LL_LPUART_ClearFlag_PE
	TCCF	LL_LPUART_ClearFlag_TC
	WUCF	LL_LPUART_ClearFlag_WKUP
ISR	BUSY	LL_LPUART_IsActiveFlag_BUSY
	CMF	LL_LPUART_IsActiveFlag_CM
	CTS	LL_LPUART_IsActiveFlag_CTS
	CTSIF	LL_LPUART_IsActiveFlag_nCTS
	FE	LL_LPUART_IsActiveFlag_FE
	IDLE	LL_LPUART_IsActiveFlag_IDLE
	NE	LL_LPUART_IsActiveFlag_NE
	ORE	LL_LPUART_IsActiveFlag_ORE
	PE	LL_LPUART_IsActiveFlag_PE
	REACK	LL_LPUART_IsActiveFlag_REACK
	RWU	LL_LPUART_IsActiveFlag_RWU
	RXFF	LL_LPUART_IsActiveFlag_RXFF
	RXFT	LL_LPUART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_LPUART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_LPUART_IsActiveFlag_SBK
	TC	LL_LPUART_IsActiveFlag_TC
	TEACK	LL_LPUART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_LPUART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_LPUART_IsActiveFlag_TXFE
	TXFT	LL_LPUART_IsActiveFlag_TXFT
	WUF	LL_LPUART_IsActiveFlag_WKUP
PRESC	PRESCALER	LL_LPUART_GetPrescaler
		LL_LPUART_SetPrescaler
RDR	RDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_ReceiveData8
		LL_LPUART_ReceiveData9
RQR	MMRQ	LL_LPUART_RequestEnterMuteMode
	RXFRQ	LL_LPUART_RequestRxDataFlush
	SBKRQ	LL_LPUART_RequestBreakSending
TDR	TDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_TransmitData8
		LL_LPUART_TransmitData9

119.21 PKA

Table 45. **Correspondence between PKA registers and PKA low-layer driver functions**

Register	Field	Function
CLRFR	ADDRERRFC	LL_PKA_ClearFlag_ADDERR
	OPERRFC	LL_PKA_ClearFlag_OPERR
	PROCENDFC	LL_PKA_ClearFlag_PROCEND
	RAMERRFC	LL_PKA_ClearFlag_RAMERR
CR	ADDRERRIE	LL_PKA_DisableIT_ADDERR
		LL_PKA_EnableIT_ADDRERR
		LL_PKA_IsEnabledIT_ADDRERR
	EN	LL_PKA_Disable
		LL_PKA_Enable
		LL_PKA_IsEnabled
	MODE	LL_PKA_Config
		LL_PKA_GetMode
		LL_PKA_SetMode
	OPERRIE	LL_PKA_EnableIT_OPERR
		LL_PKA_IsEnabledIT_OPERR
	PROCENDIE	LL_PKA_DisableIT_PROCEND
		LL_PKA_EnableIT_PROCEND
		LL_PKA_IsEnabledIT_PROCEND
	RAMERRIE	LL_PKA_DisableIT_RAMERR
		LL_PKA_EnableIT_RAMERR
		LL_PKA_IsEnabledIT_RAMERR
	START	LL_PKA_Start
SR	ADDRERRF	LL_PKA_IsActiveFlag_ADDRERR
	BUSY	LL_PKA_IsActiveFlag_BUSY
	OPERRF	LL_PKA_IsActiveFlag_OPERR
	PROCENDF	LL_PKA_IsActiveFlag_PROCEND
	RAMERRF	LL_PKA_IsActiveFlag_RAMERR

119.22 PWR

Table 46. **Correspondence between PWR registers and PWR low-layer driver functions**

Register	Field	Function
BDCR	BREN	LL_PWR_DisableBkUpRegulator
		LL_PWR_EnableBkUpRegulator
		LL_PWR_IsEnabledBkUpRegulator
	MONEN	LL_PWR_DisableMonitoring
		LL_PWR_EnableMonitoring
		LL_PWR_IsEnabledMonitoring

Register	Field	Function
BDCR	VBE	LL_PWR_DisableBatteryCharging
		LL_PWR_EnableBatteryCharging
		LL_PWR_IsEnabledBatteryCharging
	VBRS	LL_PWR_GetBattChargResistor
		LL_PWR_SetBattChargResistor
BDSR	BRRDY	LL_PWR_IsActiveFlag_BRR
	TEMPH	LL_PWR_IsActiveFlag_TEMPH
	TEMPL	LL_PWR_IsActiveFlag_TEMPL
	VBATH	LL_PWR_IsActiveFlag_VBATH
	VBATL	LL_PWR_IsActiveFlag_VBATL
CR1	ALS	LL_PWR_GetAVDLevel
	SRAM1SO	LL_PWR_IsEnabledAHBRAM1ShutOff
DBPCR	DBP	LL_PWR_DisableBkUpAccess
		LL_PWR_EnableBkUpAccess
		LL_PWR_IsEnabledBkUpAccess
IORETR	IORETEN	LL_PWR_DisableIORetention
		LL_PWR_EnableIORetention
		LL_PWR_IsEnabledIORetention
	JTAGIORETEN	LL_PWR_IsEnabledJTAGIORetention
JTAGIORETR	JTAGIORETEN	LL_PWR_DisableJTAGIORetention
		LL_PWR_EnableJTAGIORetention
PMCR	AVD_READY	LL_PWR_DisableAnalogVoltageReady
		LL_PWR_EnableAnalogVoltageReady
		LL_PWR_IsEnabledAnalogVoltageReady
	BOOSTE	LL_PWR_DisableAnalogBooster
		LL_PWR_EnableAnalogBooster
		LL_PWR_IsEnabledAnalogBooster
	CSSF	LL_PWR_ClearFlag_SB
		LL_PWR_ClearFlag_STOP
	ETHERNETSO	LL_PWR_DisableETHERNETRAMShutOff
		LL_PWR_EnableETHERNETRAMShutOff
		LL_PWR_IsEnabledETHERNETRAMShutOff
	FLPS	LL_PWR_DisableFlashPowerDown
		LL_PWR_EnableFlashPowerDown
		LL_PWR_IsEnabledFlashPowerDown
	LPMS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
	SRAM1SO	LL_PWR_DisableAHBRAM1ShutOff
		LL_PWR_EnableAHBRAM1ShutOff
	SRAM2_16SO	LL_PWR_DisableAHBRAM2_16K_ShutOff
		LL_PWR_EnableAHBRAM2_16K_ShutOff

Register	Field	Function
PMCR	SRAM2_16SO	LL_PWR_IsEnabledAHBRAM2_16K_ShutOff
	SRAM2_48SO	LL_PWR_DisableAHBRAM2_48K_ShutOff
		LL_PWR_EnableAHBRAM2_48K_ShutOff
		LL_PWR_IsEnabledAHBRAM2_48K_ShutOff
	SRAM3SO	LL_PWR_DisableAHBRAM3ShutOff
		LL_PWR_EnableAHBRAM3ShutOff
		LL_PWR_IsEnabledAHBRAM3ShutOff
	SVOS	LL_PWR_GetStopModeRegulVoltageScaling
		LL_PWR_SetStopModeRegulVoltageScaling
PMSR	SBF	LL_PWR_IsActiveFlag_SB
	STOPF	LL_PWR_IsActiveFlag_STOP
PRIVCFGR	NSPRIV	LL_PWR_DisableNSecurePrivilege
		LL_PWR_EnableNSecurePrivilege
		LL_PWR_IsEnabledNSecurePrivilege
	SPRIV	LL_PWR_DisableSecurePrivilege
		LL_PWR_EnableSecurePrivilege
		LL_PWR_IsEnabledSecurePrivilege
SCCR	BYPASS	LL_PWR_ConfigSupply
		LL_PWR_GetSupply
SECCFGR	APCSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	LPMSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	RETSEC	LL_PWR_ConfigSecure
	VBSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	VDMSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP1SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP2SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP3SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP4SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP5SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP6SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP7SEC	LL_PWR_ConfigSecure

Register	Field	Function
SECCFGR	WUP7SEC	LL_PWR_GetConfigSecure
	WUP8SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
UCPDR	UCPD_DBDIS	LL_PWR_DisableUCPDDeadBattery
		LL_PWR_EnableUCPDDeadBattery
		LL_PWR_IsEnabledUCPDDeadBattery
	UCPD_STDBY	LL_PWR_DisableUCPDStandbyMode
		LL_PWR_EnableUCPDStandbyMode
		LL_PWR_IsEnabledUCPDStandbyMode
USBSCR	USB33DEN	LL_PWR_DisableUSBVoltageDetector
		LL_PWR_EnableUSBVoltageDetector
		LL_PWR_IsEnabledUSBVoltageDetector
	USB33SV	LL_PWR_DisableVDDUSB
		LL_PWR_EnableVDDUSB
		LL_PWR_IsEnabledVDDUSB
VMCR	ALS	LL_PWR_SetAVDLevel
	AVDEN	LL_PWR_DisableAVD
		LL_PWR_EnableAVD
		LL_PWR_IsEnabledAVD
	PLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
	PVDEN	LL_PWR_DisablePVD
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
VMSR	AVDO	LL_PWR_IsActiveFlag_AVDO
	PVDO	LL_PWR_IsActiveFlag_PVDO
	USB33RDY	LL_PWR_IsActiveFlag_VDDUSB
	VDDIO2RDY	LL_PWR_IsActiveFlag_VDDIO2
VOSCR	VOS	LL_PWR_GetRegulVoltageScaling
		LL_PWR_SetRegulVoltageScaling
VOSSR	ACTVOSRDY	LL_PWR_IsActiveFlag_ACTVOS
	ACTVOS[1:0]	LL_PWR_GetCurrentVOS
	VOSRDY	LL_PWR_IsActiveFlag_VOS
WUCR	WKUPP1	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP2	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP3	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP4	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow

Register	Field	Function
WUCR	WKUPP5	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP6	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP7	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WKUPP8	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUPENx	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
	WUPP1	LL_PWR_SetWakeUpPinPolarityLow
	WUPP2	LL_PWR_SetWakeUpPinPolarityLow
	WUPP3	LL_PWR_SetWakeUpPinPolarityLow
	WUPP4	LL_PWR_SetWakeUpPinPolarityLow
	WUPP5	LL_PWR_SetWakeUpPinPolarityLow
	WUPP6	LL_PWR_SetWakeUpPinPolarityLow
	WUPP7	LL_PWR_SetWakeUpPinPolarityLow
	WUPP8	LL_PWR_SetWakeUpPinPolarityLow
	WUPPUPD1	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD2	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD3	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD4	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD5	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD6	LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp

Register	Field	Function
WUCR	WUPPUPD7	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPUPD8	LL_PWR_GetWakeUpPinPull
		LL_PWR_SetWakeUpPinPullDown
		LL_PWR_SetWakeUpPinPullNone
		LL_PWR_SetWakeUpPinPullUp
	WUPPx	LL_PWR_IsEnabledWakeUpPin
WUSCR	CWUF	LL_PWR_ClearFlag_WU
	CWUF1	LL_PWR_ClearFlag_WU1
	CWUF2	LL_PWR_ClearFlag_WU2
	CWUF3	LL_PWR_ClearFlag_WU3
	CWUF4	LL_PWR_ClearFlag_WU4
	CWUF5	LL_PWR_ClearFlag_WU5
	CWUF6	LL_PWR_ClearFlag_WU6
	CWUF7	LL_PWR_ClearFlag_WU7
	CWUF8	LL_PWR_ClearFlag_WU8
WUSR	WUF1	LL_PWR_IsActiveFlag_WU1
	WUF2	LL_PWR_IsActiveFlag_WU2
	WUF3	LL_PWR_IsActiveFlag_WU3
	WUF4	LL_PWR_IsActiveFlag_WU4
	WUF5	LL_PWR_IsActiveFlag_WU5
	WUF6	LL_PWR_IsActiveFlag_WU6
	WUF7	LL_PWR_IsActiveFlag_WU7
	WUF8	LL_PWR_IsActiveFlag_WU8

119.23 RCC

Table 47. Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
BDCR	BDRST	LL_RCC_ForceBackupDomainReset
		LL_RCC_ReleaseBackupDomainReset
	LSCOEN	LL_RCC_LSCO_Disable
		LL_RCC_LSCO_Enable
	LSCOSEL	LL_RCC_LSCO_GetSource
		LL_RCC_LSCO_SetSource
	LSEBYP	LL_RCC_LSE_DisableBypass
		LL_RCC_LSE_EnableBypass
	LSECSSD	LL_RCC_LSE_IsCSSDetected
	LSECSSON	LL_RCC_LSE_DisableCSS

Register	Field	Function
BDCR	LSECSSON	LL_RCC_LSE_EnableCSS
	LSEDRV	LL_RCC_LSE_GetDriveCapability
		LL_RCC_LSE_SetDriveCapability
	LSEEXT	LL_RCC_LSE_GetExternalClockType
		LL_RCC_LSE_SetExternalClockType
	LSEON	LL_RCC_LSE_Disable
		LL_RCC_LSE_Enable
	LSERDY	LL_RCC_LSE_IsReady
	LSION	LL_RCC_LSI_Disable
		LL_RCC_LSI_Enable
	LSIRDY	LL_RCC_LSI_IsReady
	RTCEN	LL_RCC_DisableRTC
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCSEL	LL_RCC_GetRTCClockSource
		LL_RCC_SetRTCClockSource
CCIPR1	*	LL_RCC_GetClockSource
		LL_RCC_SetClockSource
	TIMICSEL	LL_RCC_TIMIC_Disable
		LL_RCC_TIMIC_Enable
	UART4SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	UART5SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	UART7SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	UART8SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	UART9SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	USART10SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART1SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART2SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART3SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART6SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
CCIPR2	*	LL_RCC_GetClockSource

Register	Field	Function
CCIPR2	*	LL_RCC_SetClockSource
	LPTIM1SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM2SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM3SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM4SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM5SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM6SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	SAI1SEL	LL_RCC_GetSAIClockSource
		LL_RCC_SetSAIClockSource
	SAI2SEL	LL_RCC_GetSAIClockSource
		LL_RCC_SetSAIClockSource
	UART12SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	USART11SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
CCIPR3	*	LL_RCC_GetClockSource
		LL_RCC_SetClockSource
	LPUART1SEL	LL_RCC_GetLPUARTClockSource
		LL_RCC_SetLPUARTClockSource
	SPI1SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI2SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI3SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI4SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI5SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI6SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
CCIPR4	*	LL_RCC_GetClockSource
		LL_RCC_SetClockSource
	I2C1SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource

Register	Field	Function
CCIPR4	I2C2SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	I2C3SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	I2C4SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	I3C1SEL	LL_RCC_GetI3CClockSource
		LL_RCC_SetI3CClockSource
	I3C2SEL	LL_RCC_GetI3CClockSource
		LL_RCC_SetI3CClockSource
	OCTOSPISEL	LL_RCC_GetOCTOSPIClockSource
		LL_RCC_SetOCTOSPIClockSource
	SDMMC1SEL	LL_RCC_SetSDMMCClockSource
	SDMMC2SEL	LL_RCC_SetSDMMCClockSource
	SYSTICKSEL	LL_RCC_SetSystickClockSource
	USBSEL	LL_RCC_GetUSBClockSource
		LL_RCC_SetUSBClockSource
CCIPR5	*	LL_RCC_GetClockSource
		LL_RCC_SetClockSource
	ADCDACSEL	LL_RCC_GetADCDACClockSource
		LL_RCC_SetADCDACClockSource
	CECSEL	LL_RCC_GetCECClockSource
		LL_RCC_SetCECClockSource
	CKPERSEL	LL_RCC_GetCLKPClockSource
		LL_RCC_SetCLKPClockSource
	DACSEL	LL_RCC_GetDACLPClockSource
		LL_RCC_SetDACLPClockSource
	FDCANSEL	LL_RCC_GetFDCANClockSource
		LL_RCC_SetFDCANClockSource
	RNGSEL	LL_RCC_GetRNGClockSource
		LL_RCC_SetRNGClockSource
CFGR1	MCO1	LL_RCC_ConfigMCO
	MCO1PRE	LL_RCC_ConfigMCO
	MCO2	LL_RCC_ConfigMCO
	MCO2PRE	LL_RCC_ConfigMCO
	RTCPRE	LL_RCC_GetRTC_HSEPrescaler
		LL_RCC_SetRTC_HSEPrescaler
	STOPKERWUCK	LL_RCC_GetKerWakeUpClkSource
		LL_RCC_SetKerWakeUpClkSource
	STOPWUCK	LL_RCC_GetClkAfterWakeFromStop
		LL_RCC_SetClkAfterWakeFromStop

Register	Field	Function
CFGR1	SW	LL_RCC_SetSysClkSource
	SWS	LL_RCC_GetSysClkSource
	TIMPRE	LL_RCC_GetTIMPrescaler
		LL_RCC_SetTIMPrescaler
CFGR2	HPRE	LL_RCC_GetAHBPrescaler
		LL_RCC_SetAHBPrescaler
	PPRE1	LL_RCC_GetAPB1Prescaler
		LL_RCC_SetAPB1Prescaler
	PPRE2	LL_RCC_GetAPB2Prescaler
		LL_RCC_SetAPB2Prescaler
CFGR3	PPRE3	LL_RCC_GetAPB2Prescaler
		LL_RCC_SetAPB3Prescaler
CICR	CSIRDYC	LL_RCC_ClearFlag_CSIRDY
	HSECSSC	LL_RCC_ClearFlag_HSECSS
	HSERDYC	LL_RCC_ClearFlag_HSERDY
	HSI48RDYC	LL_RCC_ClearFlag_HSI48RDY
	HSIRDYC	LL_RCC_ClearFlag_HSIRDY
	LSERDYC	LL_RCC_ClearFlag_LSERDY
	LSIRDYC	LL_RCC_ClearFlag_LSIRDY
	PLL1RDYC	LL_RCC_ClearFlag_PLL1RDY
	PLL2RDYC	LL_RCC_ClearFlag_PLL2RDY
	PLL3RDYC	LL_RCC_ClearFlag_PLL3RDY
CIER	CSIRDYIE	LL_RCC_DisableIT_CSIRDY
		LL_RCC_EnableIT_CSIRDY
		LL_RCC_IsEnabledIT_CSIRDY
	HSERDYIE	LL_RCC_DisableIT_HSERDY
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSI48RDYIE	LL_RCC_DisableIT_HSI48RDY
		LL_RCC_EnableIT_HSI48RDY
		LL_RCC_IsEnabledIT_HSI48RDY
	HSIRDYIE	LL_RCC_DisableIT_HSIRDY
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSERDYIE	LL_RCC_DisableIT_LSERDY
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYIE	LL_RCC_DisableIT_LSIRDY
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	PLL1RDYIE	LL_RCC_DisableIT_PLL1RDY

Register	Field	Function
CIER	PLL1RDYIE	LL_RCC_EnableIT_PLL1RDY
		LL_RCC_IsEnabledIT_PLL1RDY
	PLL2RDYIE	LL_RCC_DisableIT_PLL2RDY
		LL_RCC_EnableIT_PLL2RDY
		LL_RCC_IsEnabledIT_PLL2RDY
	PLL3RDYIE	LL_RCC_DisableIT_PLL3RDY
		LL_RCC_EnableIT_PLL3RDY
		LL_RCC_IsEnabledIT_PLL3RDY
CIFR	CSIRDYF	LL_RCC_IsActiveFlag_CSIRDY
	HSECSSF	LL_RCC_IsActiveFlag_HSECSS
	HSERDYF	LL_RCC_IsActiveFlag_HSERDY
	HSI48RDYF	LL_RCC_IsActiveFlag_HSI48RDY
	HSIRDYF	LL_RCC_IsActiveFlag_HSIRDY
	LSERDYF	LL_RCC_IsActiveFlag_LSERDY
	LSIRDYF	LL_RCC_IsActiveFlag_LSIRDY
	PLL1RDYF	LL_RCC_IsActiveFlag_PLL1RDY
	PLL2RDYF	LL_RCC_IsActiveFlag_PLL2RDY
	PLL3RDYF	LL_RCC_IsActiveFlag_PLL3RDY
CR	CSIKERON	LL_RCC_CSI_DisableInStopMode
		LL_RCC_CSI_EnableInStopMode
		LL_RCC_CSI_IsEnabledInStopMode
	CSION	LL_RCC_CSI_Disable
		LL_RCC_CSI_Enable
	CSIRDY	LL_RCC_CSI_IsReady
	HSEBYP	LL_RCC_HSE_DisableBypass
		LL_RCC_HSE_EnableBypass
	HSECSSON	LL_RCC_HSE_EnableCSS
	HSEEXT	LL_RCC_HSE_GetExternalClockType
		LL_RCC_HSE_SetExternalClockType
	HSEON	LL_RCC_HSE_Disable
		LL_RCC_HSE_Enable
	HSERDY	LL_RCC_HSE_IsReady
	HSI48ON	LL_RCC_HSI48_Disable
		LL_RCC_HSI48_Enable
	HSI48RDY	LL_RCC_HSI48_IsReady
	HSIDIV	LL_RCC_HSI_GetDivider
		LL_RCC_HSI_SetDivider
	HSIDIVF	LL_RCC_HSI_IsDividerReady
	HSIKERON	LL_RCC_HSI_DisableInStopMode
		LL_RCC_HSI_EnableInStopMode
		LL_RCC_HSI_IsEnabledInStopMode

Register	Field	Function
CR	HSION	LL_RCC_HSI_Disable
		LL_RCC_HSI_Enable
	HSIRDY	LL_RCC_HSI_IsReady
	PLL1ON	LL_RCC_PLL1_Enable
	PLL1RDY	LL_RCC_PLL1_IsReady
	PLL2ON	LL_RCC_PLL2_Disable
		LL_RCC_PLL2_Enable
	PLL2RDY	LL_RCC_PLL2_IsReady
	PLL3ON	LL_RCC_PLL3_Disable
		LL_RCC_PLL3_Enable
	PLL3RDY	LL_RCC_PLL3_IsReady
	PLLON	LL_RCC_PLL1_Disable
CRRCR	HSI48CAL	LL_RCC_HSI48_GetCalibration
CSICFGR	CSICAL	LL_RCC_CSI_GetCalibration
	CSITRIM	LL_RCC_CSI_GetCalibTrimming
		LL_RCC_CSI_SetCalibTrimming
HSICFGR	HSICAL	LL_RCC_HSI_GetCalibration
	HSITRIM	LL_RCC_HSI_SetCalibTrimming
ICSC3R	HSITRIM	LL_RCC_HSI_GetCalibTrimming
PLL1CFGR	PLL1FRACEN	LL_RCC_PLL1FRACN_Disable
		LL_RCC_PLL1FRACN_Enable
		LL_RCC_PLL1FRACN_IsEnabled
	PLL1M	LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetM
		LL_RCC_PLL1_SetM
	PLL1N	LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetN
		LL_RCC_PLL1_SetN
	PLL1P	LL_RCC_PLL1_GetP
		LL_RCC_PLL1_SetP
	PLL1PEN	LL_RCC_PLL1P_Disable
		LL_RCC_PLL1P_Enable
	PLL1Q	LL_RCC_PLL1_GetQ
	PLL1QEN	LL_RCC_PLL1Q_Disable
		LL_RCC_PLL1Q_Enable
	PLL1R	LL_RCC_PLL1_ConfigDomain_SYS
	PLL1REN	LL_RCC_PLL1R_Disable
		LL_RCC_PLL1R_Enable
	PLL1RGE	LL_RCC_PLL1_SetVCOInputRange
	PLL1SRC	LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetSource

Register	Field	Function
PLL1CFGR	PLL1SRC	LL_RCC_PLL1_SetSource
PLL1DIVR	PLL1R	LL_RCC_PLL1_GetR
		LL_RCC_PLL1_SetR
PLL1FRACR	PLL1FRACN	LL_RCC_PLL1_GetFRACN
		LL_RCC_PLL1_SetFRACN
PLL2CFGR	PLL2FRACEN	LL_RCC_PLL2FRACN_Disable
		LL_RCC_PLL2FRACN_Enable
		LL_RCC_PLL2FRACN_IsEnabled
	PLL2M	LL_RCC_PLL2_GetM
		LL_RCC_PLL2_SetM
	PLL2N	LL_RCC_PLL2_GetN
		LL_RCC_PLL2_SetN
	PLL2P	LL_RCC_PLL2_GetP
		LL_RCC_PLL2_SetP
	PLL2PEN	LL_RCC_PLL2P_Disable
		LL_RCC_PLL2P_Enable
		LL_RCC_PLL2P_IsEnabled
	PLL2Q	LL_RCC_PLL2_GetQ
		LL_RCC_PLL2_SetR
	PLL2QEN	LL_RCC_PLL2Q_Enable
		LL_RCC_PLL2Q_IsEnabled
		LL_RCC_PLL2_Disable
	PLL2REN	LL_RCC_PLL2R_Disable
		LL_RCC_PLL2R_Enable
		LL_RCC_PLL2R_IsEnabled
	PLL2RGE	LL_RCC_PLL2_SetVCOInputRange
	PLL2SRC	LL_RCC_PLL2_GetSource
		LL_RCC_PLL2_SetSource
	PLL2VCOSEL	LL_RCC_PLL2_SetVCOOutputRange
PLL2DIVR	PLL2R	LL_RCC_PLL2_GetR
PLL2FRACR	PLL2FRACN	LL_RCC_PLL2_GetFRACN
		LL_RCC_PLL2_SetFRACN
PLL3CFGR	PLL3FRACEN	LL_RCC_PLL3FRACN_Disable
		LL_RCC_PLL3FRACN_Enable
		LL_RCC_PLL3FRACN_IsEnabled
	PLL3M	LL_RCC_PLL3_GetM
		LL_RCC_PLL3_SetM
	PLL3N	LL_RCC_PLL3_GetN
		LL_RCC_PLL3_SetN
	PLL3P	LL_RCC_PLL3_GetP
		LL_RCC_PLL3_SetP

Register	Field	Function
PLL3CFGR	PLL3PEN	LL_RCC_PLL3P_Disable
		LL_RCC_PLL3P_Enable
		LL_RCC_PLL3P_IsEnabled
	PLL3Q	LL_RCC_PLL3_GetQ
		LL_RCC_PLL3_SetR
	PLL3QEN	LL_RCC_PLL3Q_Disable
		LL_RCC_PLL3Q_Enable
		LL_RCC_PLL3Q_IsEnabled
	PLL3REN	LL_RCC_PLL3R_Disable
		LL_RCC_PLL3R_Enable
		LL_RCC_PLL3R_IsEnabled
	PLL3RGE	LL_RCC_PLL3_SetVCOInputRange
	PLL3SRC	LL_RCC_PLL3_GetSource
		LL_RCC_PLL3_SetSource
	PLL3VCOSEL	LL_RCC_PLL3_SetVCOOutputRange
PLL3DIVR	PLL3R	LL_RCC_PLL3_GetR
PLL3FRACR	PLL3FRACN	LL_RCC_PLL3_GetFRACN
		LL_RCC_PLL3_SetFRACN
PLLCFGR	DIVP1EN	LL_RCC_PLL1P_IsEnabled
	DIVQ1EN	LL_RCC_PLL1Q_IsEnabled
	DIVR1EN	LL_RCC_PLL1R_IsEnabled
	PLL1Q	LL_RCC_PLL1_SetQ
	PLL1VCOSEL	LL_RCC_PLL1_SetVCOOutputRange
	PLL2Q	LL_RCC_PLL2_SetQ
	PLL3Q	LL_RCC_PLL3_SetQ
PRIVCFGR	NSPRIV	LL_RCC_DisableNSecPrivilegedMode
		LL_RCC_EnableNSecPrivilegedMode
		LL_RCC_IsEnabledNSecPrivilegedMode
	SPRIV	LL_RCC_DisableSecPrivilegedMode
		LL_RCC_EnableSecPrivilegedMode
		LL_RCC_IsEnabledSecPrivilegedMode
RSR	BORRSTF	LL_RCC_IsActiveFlag_BORRST
	IWDGRSTF	LL_RCC_IsActiveFlag_IWDGRST
	LPWRRSTF	LL_RCC_IsActiveFlag_LPWRRST
	PINRSTF	LL_RCC_IsActiveFlag_PINRST
	RMVF	LL_RCC_ClearResetFlags
	SFTRSTF	LL_RCC_IsActiveFlag_SFTRST
	WWDGRSTF	LL_RCC_IsActiveFlag_WWDGRST
SECCFGR	CKPERSELSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	CSISEC	LL_RCC_ConfigSecure

Register	Field	Function
SECCFGR	CSISEC	LL_RCC_GetConfigSecure
	HSESEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	HSI48SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	HSISEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	LSESEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	LSISEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL1SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL2SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL3SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PRESCSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	RMVFSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	SYSCLKSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure

119.24 RNG

Table 48. Correspondence between RNG registers and RNG low-layer driver functions

Register	Field	Function
CR	CED	LL_RNG_DisableClkErrorDetect
		LL_RNG_EnableClkErrorDetect
		LL_RNG_IsEnabledClkErrorDetect
	CLKDIV	LL_RNG_GetClockDivider
		LL_RNG_SetClockDivider
	CONDRST	LL_RNG_DisableCondReset
		LL_RNG_EnableCondReset
		LL_RNG_IsEnabledCondReset
	CONFIGLOCK	LL_RNG_ConfigLock
		LL_RNG_IsConfigLocked
	IE	LL_RNG_DisableIT
		LL_RNG_EnableIT
		LL_RNG_IsEnabledIT

Register	Field	Function
CR	NISTC	LL_RNG_DisableNistCompliance
		LL_RNG_EnableNistCompliance
		LL_RNG_IsEnabledNistCompliance
	RNGEN	LL_RNG_Disable
		LL_RNG_Enable
		LL_RNG_IsEnabled
	RNG_CONFIG1	LL_RNG_GetConfig1
		LL_RNG_SetConfig1
	RNG_CONFIG2	LL_RNG_GetConfig2
		LL_RNG_SetConfig2
	RNG_CONFIG3	LL_RNG_GetConfig3
		LL_RNG_SetConfig3
DR	RNDATA	LL_RNG_ReadRandData32
HTCR	HTCFG	LL_RNG_GetHealthConfig
		LL_RNG_SetHealthConfig
SR	CECS	LL_RNG_IsActiveFlag_CECS
	CEIS	LL_RNG_ClearFlag_CEIS
		LL_RNG_IsActiveFlag_CEIS
	DRDY	LL_RNG_IsActiveFlag_DRDY
	SECS	LL_RNG_IsActiveFlag_SECS
	SEIS	LL_RNG_ClearFlag_SEIS
		LL_RNG_IsActiveFlag_SEIS

119.25 RTC

Table 49. Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
RCT_ALRMASSR	SS	LL_RTC_ALMA_GetSubSecond
		LL_RTC_ALMA_SetSubSecond
RTC_ALRABINR	SSCLR	LL_RTC_ALMA_GetBinAutoClr
		LL_RTC_ALMA_SetBinAutoClr
RTC_ALRBBINR	SSCLR	LL_RTC_ALMB_GetBinAutoClr
		LL_RTC_ALMB_SetBinAutoClr
RTC_ALRMAR	DT	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_SetDay
	DU	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_GetWeekDay
		LL_RTC_ALMA_SetDay
		LL_RTC_ALMA_SetWeekDay
	HT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour

Register	Field	Function
RTC_ALRMAR	HT	LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	HU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	MNT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MNU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MSK1	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK2	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK3	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK4	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	PM	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetTimeFormat
		LL_RTC_ALMA_SetTimeFormat
	ST	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	SU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	WDSEL	LL_RTC_ALMA_DisableWeekday
		LL_RTC_ALMA_EnableWeekday
RTC_ALRMASSR	MASKSS	LL_RTC_ALMA_GetSubSecondMask
		LL_RTC_ALMA_SetSubSecondMask
RTC_ALRMBR	DT	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_SetDay
	DU	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_GetWeekDay

Register	Field	Function
RTC_ALRMBR	DU	LL_RTC_ALMB_SetDay
		LL_RTC_ALMB_SetWeekDay
	HT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	HU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	MNT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MNU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MSK1	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK2	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK3	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK4	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	PM	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetTimeFormat
		LL_RTC_ALMB_SetTimeFormat
	ST	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	SU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	WDSEL	LL_RTC_ALMB_DisableWeekday
		LL_RTC_ALMB_EnableWeekday
RTC_ALRMBSSR	MASKSS	LL_RTC_ALMB_GetSubSecondMask
		LL_RTC_ALMB_SetSubSecondMask

Register	Field	Function
RTC_ALRMBSSR	SS	LL_RTC_ALMB_GetSubSecond
		LL_RTC_ALMB_SetSubSecond
RTC_CALR	CALM	LL_RTC_CAL_GetMinus
		LL_RTC_CAL_SetMinus
	CALP	LL_RTC_CAL_IsPulseInserted
		LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	LPCAL	LL_RTC_CAL_LowPower_Disable
		LL_RTC_CAL_LowPower_Enable
		LL_RTC_CAL_LowPower_IsEnabled
RTC_CR	ADD1H	LL_RTC_TIME_IncHour
	ALRAE	LL_RTC_ALMA_Disable
		LL_RTC_ALMA_Enable
	ALRAIE	LL_RTC_DisableIT_ALRA
		LL_RTC_EnableIT_ALRA
		LL_RTC_IsEnabledIT_ALRA
	ALRBE	LL_RTC_ALMB_Disable
		LL_RTC_ALMB_Enable
	ALRBIE	LL_RTC_DisableIT_ALRB
		LL_RTC_EnableIT_ALRB
		LL_RTC_IsEnabledIT_ALRB
	BKP	LL_RTC_TIME_DisableDayLightStore
		LL_RTC_TIME_EnableDayLightStore
		LL_RTC_TIME_IsDayLightStoreEnabled
	BYPSHAD	LL_RTC_DisableShadowRegBypass
		LL_RTC_EnableShadowRegBypass
		LL_RTC_IsShadowRegBypassEnabled
	COE	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	COSEL	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	FMT	LL_RTC_GetHourFormat
		LL_RTC_SetHourFormat
	ITSE	LL_RTC_TS_DisableInternalEvent
		LL_RTC_TS_EnableInternalEvent
	OSEL	LL_RTC_GetAlarmOutEvent
		LL_RTC_SetAlarmOutEvent
	OUT2EN	LL_RTC_DisableOutput2

Register	Field	Function
RTC_CR	OUT2EN	LL_RTC_EnableOutput2
		LL_RTC_IsOutput2Enabled
	POL	LL_RTC_GetOutputPolarity
		LL_RTC_SetOutputPolarity
	REFCKON	LL_RTC_DisableRefClock
		LL_RTC_EnableRefClock
	SSRUIE	LL_RTC_DisableIT_SSRU
		LL_RTC_EnableIT_SSRU
		LL_RTC_IsEnabledIT_SSRU
	SUB1H	LL_RTC_TIME_DecHour
	TAMPALRM_PU	LL_RTC_EnableAlarmPullUp
		LL_RTC_IsAlarmPullUpEnabled
	TAMPALRM_TYPE	LL_RTC_GetAlarmOutputType
		LL_RTC_SetAlarmOutputType
	TAMPOE	LL_RTC_DisableTamperOutput
		LL_RTC_EnableTamperOutput
		LL_RTC_IsTamperOutputEnabled
	TAMPTS	LL_RTC_TS_DisableOnTamper
		LL_RTC_TS_EnableOnTamper
	TSE	LL_RTC_TS_Disable
		LL_RTC_TS_Enable
	TSEDGE	LL_RTC_TS_GetActiveEdge
		LL_RTC_TS_SetActiveEdge
	TSIE	LL_RTC_DisableIT_TS
		LL_RTC_EnableIT_TS
		LL_RTC_IsEnabledIT_TS
	WUCKSEL	LL_RTC_WAKEUP_GetClock
		LL_RTC_WAKEUP_SetClock
	WUTE	LL_RTC_WAKEUP_Disable
		LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	LL_RTC_DisableIT_WUT
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
RTC_DR	DT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay

Register	Field	Function
RTC_DR	DU	LL_RTC_DATE_SetDay
	MT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	MU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	WDU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetWeekDay
		LL_RTC_DATE_SetWeekDay
	YT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
	YU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
RTC_ICSR	BCDU	LL_RTC_GetBinMixBCDU
		LL_RTC_SetBinMixBCDU
	BIN	LL_RTC_GetBinaryMode
		LL_RTC_SetBinaryMode
	INIT	LL_RTC_DisableInitMode
		LL_RTC_EnableInitMode
	INITF	LL_RTC_IsActiveFlag_INIT
	INITS	LL_RTC_IsActiveFlag_INITS
	RECALPF	LL_RTC_IsActiveFlag_RECALP
	RSF	LL_RTC_ClearFlag_RS
		LL_RTC_IsActiveFlag_RS
	SHPF	LL_RTC_IsActiveFlag_SHP
	WUTWF	LL_RTC_IsActiveFlag_WUTW
RTC_MISR	ALRAMF	LL_RTC_IsActiveFlag_ALRAM
	ALRBMF	LL_RTC_IsActiveFlag_ALRBM
	ITSMF	LL_RTC_IsActiveFlag_ITSM
	SSRUMF	LL_RTC_IsActiveFlag_SSRUM
	TSMF	LL_RTC_IsActiveFlag_TSM
	TSOVMF	LL_RTC_IsActiveFlag_TSOVM
	WUTMF	LL_RTC_IsActiveFlag_WUTM

Register	Field	Function
RTC_PRER	PREDIV_A	LL_RTC_GetAsynchPrescaler
		LL_RTC_SetAsynchPrescaler
	PREDIV_S	LL_RTC_GetSynchPrescaler
		LL_RTC_SetSynchPrescaler
RTC_PRIVCFGR	ALRAPRIV	LL_RTC_SetRtcPrivilege
	ALRBPRIV	LL_RTC_SetRtcPrivilege
	CALPRIV	LL_RTC_SetRtcPrivilege
	INITPRIV	LL_RTC_SetRtcPrivilege
	PRIV	LL_RTC_SetRtcPrivilege
	TSPRIV	LL_RTC_SetRtcPrivilege
	WUTPRIV	LL_RTC_SetRtcPrivilege
RTC_SCR	CALRAF	LL_RTC_ClearFlag_ALRA
	CALRBF	LL_RTC_ClearFlag_ALRB
	CITSF	LL_RTC_ClearFlag_ITS
	CSSRUF	LL_RTC_ClearFlag_SSRU
	CTSF	LL_RTC_ClearFlag_TS
	CTSOVF	LL_RTC_ClearFlag_TSOV
	CWUTF	LL_RTC_ClearFlag_WUT
RTC_SECCFGR	ALRASEC	LL_RTC_SetRtcSecure
	ALRBSEC	LL_RTC_SetRtcSecure
	CALSEC	LL_RTC_SetRtcSecure
	INISEC	LL_RTC_SetRtcSecure
	INITSEC	LL_RTC_SetRtcSecure
	SEC	LL_RTC_SetRtcSecure
	TSSEC	LL_RTC_SetRtcSecure
	WUTSEC	LL_RTC_SetRtcSecure
RTC_SHIFTR	ADD1S	LL_RTC_TIME_Synchronize
	SUBFS	LL_RTC_TIME_Synchronize
RTC_SR	ALRAF	LL_RTC_IsActiveFlag_ALRA
	ALRBF	LL_RTC_IsActiveFlag_ALRB
	ITSF	LL_RTC_IsActiveFlag_ITS
	SSRUF	LL_RTC_IsActiveFlag_SSRU
	TSF	LL_RTC_IsActiveFlag_TS
	TSOVF	LL_RTC_IsActiveFlag_TSOV
	WUTF	LL_RTC_IsActiveFlag_WUT
RTC_SSR	SS	LL_RTC_TIME_GetSubSecond
RTC_TR	HT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config

Register	Field	Function
RTC_TR	HU	LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat
	ST	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
	SU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
RTC_TSDR	DT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	DU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	MT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	MU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	SS	LL_RTC_TS_GetSubSecond
	WDU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetWeekDay
RTC_TSTR	HT	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	HU	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	MNT	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime
	MNU	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime

Register	Field	Function
RTC_TSTR	PM	LL_RTC_TS_GetTimeFormat
	ST	LL_RTC_TS_GetSecond
		LL_RTC_TS_GetTime
	SU	LL_RTC_TS_GetSecond
		LL_RTC_TS_GetTime
RTC_WPR	KEY	LL_RTC_DisableWriteProtection
		LL_RTC_EnableWriteProtection
RTC_WUTR	WUT	LL_RTC_WAKEUP_GetAutoReload
		LL_RTC_WAKEUP_SetAutoReload
TAMP_ATCR1	ATCKSEL	LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
		LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler
	ATOSHARE	LL_RTC_TAMPER_ATAMP_DisableOutputSharing
		LL_RTC_TAMPER_ATAMP_EnableOutputSharing
	ATPER	LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod
		LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod
	FLTEN	LL_RTC_TAMPER_ATAMP_DisableFilter
		LL_RTC_TAMPER_ATAMP_EnableFilter
	TAMP1AM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
	TAMP2AM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
	TAMPxAM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
TAMP_ATCR2	ATOSELx	LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection
		LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection
TAMP_ATOR	INITS	LL_RTC_IsActiveFlag_ATAMP_INITS
	SEEDF	LL_RTC_IsActiveFlag_ATAMP_SEEDF
TAMP_ATSEEDR	SEED	LL_RTC_TAMPER_ATAMP_WriteSeed
TAMP_BKPxR	BKP	LL_RTC_BKP_GetRegister
		LL_RTC_BKP_SetRegister
TAMP_COUNT1R	COUNT	LL_RTC_GetMonotonicCounter
		LL_RTC_IncrementMonotonicCounter
TAMP_CR1	ITAMP1E	LL_RTC_TAMPER_ITAMP_Disable
		LL_RTC_TAMPER_ITAMP_Enable
	ITAMP2E	LL_RTC_TAMPER_ITAMP_Disable
		LL_RTC_TAMPER_ITAMP_Enable
	ITAMPxE	LL_RTC_TAMPER_ITAMP_Disable
	ITAMPxE..	LL_RTC_TAMPER_ITAMP_Enable
	TAMP1E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP2E...	LL_RTC_TAMPER_Disable

Register	Field	Function
TAMP_CR1	TAMP2E...	LL_RTC_TAMPER_Enable
TAMP_CR2	TAMP1MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP1NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP1TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP2MF...	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP2NOERASE ...	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP2TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMPxTRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
TAMP_FLTCR	TAMPFLT	LL_RTC_TAMPER_GetFilterCount
		LL_RTC_TAMPER_SetFilterCount
	TAMPFREQ	LL_RTC_TAMPER_GetSamplingFreq
		LL_RTC_TAMPER_SetSamplingFreq
	TAMPPRCH	LL_RTC_TAMPER_GetPrecharge
		LL_RTC_TAMPER_SetPrecharge
	TAMPPUDIS	LL_RTC_TAMPER_DisablePullUp
		LL_RTC_TAMPER_EnablePullUp
TAMP_IER	ITAMP11IE	LL_RTC_DisableIT_ITAMP11
		LL_RTC_EnableIT_ITAMP11
		LL_RTC_IsEnabledIT_ITAMP11
	ITAMP12IE	LL_RTC_DisableIT_ITAMP12
		LL_RTC_EnableIT_ITAMP12
		LL_RTC_IsEnabledIT_ITAMP12
	ITAMP13IE	LL_RTC_DisableIT_ITAMP13
		LL_RTC_EnableIT_ITAMP13
		LL_RTC_IsEnabledIT_ITAMP13
	ITAMP15IE	LL_RTC_DisableIT_ITAMP15
		LL_RTC_EnableIT_ITAMP15
		LL_RTC_IsEnabledIT_ITAMP15
	ITAMP1IE	LL_RTC_DisableIT_ITAMP1
		LL_RTC_EnableIT_ITAMP1
		LL_RTC_IsEnabledIT_ITAMP1
	ITAMP2IE	LL_RTC_DisableIT_ITAMP2
		LL_RTC_EnableIT_ITAMP2
		LL_RTC_IsEnabledIT_ITAMP2

Register	Field	Function
TAMP_IER	ITAMP3IE	LL_RTC_DisableIT_ITAMP3
		LL_RTC_EnableIT_ITAMP3
		LL_RTC_IsEnabledIT_ITAMP3
	ITAMP4IE	LL_RTC_DisableIT_ITAMP4
		LL_RTC_EnableIT_ITAMP4
		LL_RTC_IsEnabledIT_ITAMP4
	ITAMP5IE	LL_RTC_DisableIT_ITAMP5
		LL_RTC_EnableIT_ITAMP5
		LL_RTC_IsEnabledIT_ITAMP5
	ITAMP6IE	LL_RTC_DisableIT_ITAMP6
		LL_RTC_EnableIT_ITAMP6
		LL_RTC_IsEnabledIT_ITAMP6
	ITAMP7IE	LL_RTC_DisableIT_ITAMP7
		LL_RTC_EnableIT_ITAMP7
		LL_RTC_IsEnabledIT_ITAMP7
	ITAMP8IE	LL_RTC_DisableIT_ITAMP8
		LL_RTC_EnableIT_ITAMP8
		LL_RTC_IsEnabledIT_ITAMP8
	ITAMP9IE	LL_RTC_DisableIT_ITAMP9
		LL_RTC_EnableIT_ITAMP9
		LL_RTC_IsEnabledIT_ITAMP9
	TAMP1IE	LL_RTC_DisableIT_TAMP1
		LL_RTC_EnableIT_TAMP1
		LL_RTC_IsEnabledIT_TAMP1
	TAMP2IE	LL_RTC_DisableIT_TAMP2
		LL_RTC_EnableIT_TAMP2
		LL_RTC_IsEnabledIT_TAMP2
	TAMP3IE	LL_RTC_DisableIT_TAMP3
		LL_RTC_EnableIT_TAMP3
		LL_RTC_IsEnabledIT_TAMP3
	TAMP4IE	LL_RTC_DisableIT_TAMP4
		LL_RTC_EnableIT_TAMP4
		LL_RTC_IsEnabledIT_TAMP4
	TAMP5IE	LL_RTC_DisableIT_TAMP5
		LL_RTC_EnableIT_TAMP5
		LL_RTC_IsEnabledIT_TAMP5
	TAMP6IE	LL_RTC_DisableIT_TAMP6
		LL_RTC_EnableIT_TAMP6
		LL_RTC_IsEnabledIT_TAMP6
	TAMP7IE	LL_RTC_DisableIT_TAMP7
		LL_RTC_EnableIT_TAMP7

Register	Field	Function
TAMP_IER	TAMP7IE	LL_RTC_IsEnabledIT_TAMP7
	TAMP8IE	LL_RTC_DisableIT_TAMP8
		LL_RTC_EnableIT_TAMP8
		LL_RTC_IsEnabledIT_TAMP8
TAMP_MISR	ITAMP11MF	LL_RTC_IsActiveFlag_ITAMP11M
	ITAMP12MF	LL_RTC_IsActiveFlag_ITAMP12M
	ITAMP13MF	LL_RTC_IsActiveFlag_ITAMP13M
	ITAMP15MF	LL_RTC_IsActiveFlag_ITAMP15M
	ITAMP1MF	LL_RTC_IsActiveFlag_ITAMP1M
	ITAMP2MF	LL_RTC_IsActiveFlag_ITAMP2M
	ITAMP3MF	LL_RTC_IsActiveFlag_ITAMP3M
	ITAMP4MF	LL_RTC_IsActiveFlag_ITAMP4M
	ITAMP5MF	LL_RTC_IsActiveFlag_ITAMP5M
	ITAMP6MF	LL_RTC_IsActiveFlag_ITAMP6M
	ITAMP7MF	LL_RTC_IsActiveFlag_ITAMP7M
	ITAMP8MF	LL_RTC_IsActiveFlag_ITAMP8M
	ITAMP9MF	LL_RTC_IsActiveFlag_ITAMP9M
	TAMP1MF	LL_RTC_IsActiveFlag_TAMP1M
	TAMP2MF	LL_RTC_IsActiveFlag_TAMP2M
	TAMP3MF	LL_RTC_IsActiveFlag_TAMP3M
	TAMP4MF	LL_RTC_IsActiveFlag_TAMP4M
	TAMP5MF	LL_RTC_IsActiveFlag_TAMP5M
	TAMP6MF	LL_RTC_IsActiveFlag_TAMP6M
	TAMP7MF	LL_RTC_IsActiveFlag_TAMP7M
	TAMP8MF	LL_RTC_IsActiveFlag_TAMP8M
TAMP_PRIVCFGR	BKPRWPRIV	LL_RTC_GetBackupRegisterPrivilege
		LL_RTC_SetBackupRegisterPrivilege
	BKPWPRIV	LL_RTC_GetBackupRegisterPrivilege
		LL_RTC_SetBackupRegisterPrivilege
	TAMPPRIV	LL_RTC_GetTampPrivilege
		LL_RTC_SetTampPrivilege
TAMP_SCR	CITAMP11F	LL_RTC_ClearFlag_ITAMP11
	CITAMP12F	LL_RTC_ClearFlag_ITAMP12
	CITAMP13F	LL_RTC_ClearFlag_ITAMP13
	CITAMP15F	LL_RTC_ClearFlag_ITAMP15
	CITAMP1F	LL_RTC_ClearFlag_ITAMP1
	CITAMP2F	LL_RTC_ClearFlag_ITAMP2
	CITAMP3F	LL_RTC_ClearFlag_ITAMP3
	CITAMP4F	LL_RTC_ClearFlag_ITAMP4
	CITAMP5F	LL_RTC_ClearFlag_ITAMP5
	CITAMP6F	LL_RTC_ClearFlag_ITAMP6

Register	Field	Function
TAMP_SCR	CITAMP7F	LL_RTC_ClearFlag_ITAMP7
	CITAMP8F	LL_RTC_ClearFlag_ITAMP8
	CITAMP9F	LL_RTC_ClearFlag_ITAMP9
	CTAMP1F	LL_RTC_ClearFlag_TAMP1
	CTAMP2F	LL_RTC_ClearFlag_TAMP2
	CTAMP3F	LL_RTC_ClearFlag_TAMP3
	CTAMP4F	LL_RTC_ClearFlag_TAMP4
	CTAMP5F	LL_RTC_ClearFlag_TAMP5
	CTAMP6F	LL_RTC_ClearFlag_TAMP6
	CTAMP7F	LL_RTC_ClearFlag_TAMP7
	CTAMP8F	LL_RTC_ClearFlag_TAMP8
TAMP_SECCFGR	BKPRWSEC	LL_RTC_GetBackupRegProtectionStartZone2
		LL_RTC_SetBackupRegProtection
	BKPWSEC	LL_RTC_GetBackupRegProtectionStartZone3
		LL_RTC_SetBackupRegProtection
	TAMPSEC	LL_RTC_GetTampSecure
		LL_RTC_SetTampSecure
TAMP_SR	ITAMP11F	LL_RTC_IsActiveFlag_ITAMP11
	ITAMP12F	LL_RTC_IsActiveFlag_ITAMP12
	ITAMP13F	LL_RTC_IsActiveFlag_ITAMP13
	ITAMP15F	LL_RTC_IsActiveFlag_ITAMP15
	ITAMP1F	LL_RTC_IsActiveFlag_ITAMP1
	ITAMP2F	LL_RTC_IsActiveFlag_ITAMP2
	ITAMP3F	LL_RTC_IsActiveFlag_ITAMP3
	ITAMP4F	LL_RTC_IsActiveFlag_ITAMP4
	ITAMP5F	LL_RTC_IsActiveFlag_ITAMP5
	ITAMP6F	LL_RTC_IsActiveFlag_ITAMP6
	ITAMP7F	LL_RTC_IsActiveFlag_ITAMP7
	ITAMP8F	LL_RTC_IsActiveFlag_ITAMP8
	ITAMP9F	LL_RTC_IsActiveFlag_ITAMP9
	TAMP1F	LL_RTC_IsActiveFlag_TAMP1
	TAMP2F	LL_RTC_IsActiveFlag_TAMP2
	TAMP3F	LL_RTC_IsActiveFlag_TAMP3
	TAMP4F	LL_RTC_IsActiveFlag_TAMP4
	TAMP5F	LL_RTC_IsActiveFlag_TAMP5
	TAMP6F	LL_RTC_IsActiveFlag_TAMP6
	TAMP7F	LL_RTC_IsActiveFlag_TAMP7
	TAMP8F	LL_RTC_IsActiveFlag_TAMP8

119.26 OPAMP

Table 50. Correspondence between OPAMP registers and OPAMP low-layer driver functions

Register	Field	Function
CSR	CALON	LL_OPAMP_GetMode
		LL_OPAMP_SetMode
	CALSEL	LL_OPAMP_GetCalibrationSelection
		LL_OPAMP_SetCalibrationSelection
	OPAHSM	LL_OPAMP_GetPowerMode
		LL_OPAMP_SetPowerMode
	OPAMPXEN	LL_OPAMP_Disable
		LL_OPAMP_Enable
		LL_OPAMP_IsEnabled
	OUTCAL	LL_OPAMP_IsCalibrationOutputSet
	PGGAIN	LL_OPAMP_GetPGAGain
		LL_OPAMP_SetPGAGain
	USERTRIM	LL_OPAMP_GetTrimmingMode
		LL_OPAMP_SetTrimmingMode
	VMSEL	LL_OPAMP_GetFunctionalMode
		LL_OPAMP_GetInputInverting
		LL_OPAMP_SetFunctionalMode
		LL_OPAMP_SetInputInverting
	VPSEL	LL_OPAMP_GetInputNonInverting
		LL_OPAMP_SetInputNonInverting
HSOTR	TRIMHSOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMHSOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
OTR	TRIMOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue

119.27 SPI

Table 51. Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CFG1	BPASS	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler
	CRCEN	LL_SPI_DisableCRC
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC

Register	Field	Function
CFG1	CRCSIZE	LL_SPI_GetCRCWidth
		LL_SPI_SetCRCWidth
	DSIZE	LL_SPI_GetDataWidth
		LL_SPI_SetDataWidth
	FTHLV	LL_SPI_GetFIFOThreshold
		LL_SPI_SetFIFOThreshold
	MBR	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler
	RXDMAEN	LL_SPI_DisableDMAReq_RX
		LL_SPI_EnableDMAReq_RX
		LL_SPI_IsEnabledDMAReq_RX
	TXDMAEN	LL_SPI_DisableDMAReq_TX
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	UDRCFG	LL_SPI_GetUDRConfiguration
		LL_SPI_SetUDRConfiguration
CFG2	AFCNTR	LL_SPI_DisableGPIOControl
		LL_SPI_EnableGPIOControl
		LL_SPI_IsEnabledGPIOControl
	COMM	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	CPHA	LL_SPI_GetClockPhase
		LL_SPI_SetClockPhase
	CPOL	LL_SPI_GetClockPolarity
		LL_SPI_SetClockPolarity
	IOSWP	LL_SPI_DisableIOSwap
		LL_SPI_EnableIOSwap
		LL_SPI_IsEnabledIOSwap
	LSBFRST	LL_SPI_GetTransferBitOrder
		LL_SPI_SetTransferBitOrder
	MASTER	LL_SPI_GetMode
		LL_SPI_SetMode
	MIDI	LL_SPI_SetInterDataIdleness
	MSSI	LL_SPI_GetMasterSSIdleness
		LL_SPI_SetMasterSSIdleness
	SP	LL_SPI_GetStandard
		LL_SPI_SetStandard
	SSIOP	LL_SPI_GetNSSPolarity
		LL_SPI_SetNSSPolarity
	SSM	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode

Register	Field	Function
CFG2	SSOE	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
	SSOM	LL_SPI_DisableNSSPulseMgt
		LL_SPI_EnableNSSPulseMgt
		LL_SPI_IsEnabledNSSPulse
CR1	CRC33_17	LL_SPI_DisableFullSizeCRC
		LL_SPI_EnableFullSizeCRC
		LL_SPI_IsEnabledFullSizeCRC
	CSTART	LL_SPI_IsActiveMasterTransfer
		LL_SPI_StartMasterTransfer
	CSUSP	LL_SPI_SuspendMasterTransfer
	HDDIR	LL_SPI_GetHalfDuplexDirection
		LL_SPI_GetTransferDirection
		LL_SPI_SetHalfDuplexDirection
		LL_SPI_SetTransferDirection
	IOLOCK	LL_SPI_EnableIOLock
		LL_SPI_IsEnabledIOLock
	MASRX	LL_SPI_DisableMasterRxAutoSuspend
		LL_SPI_EnableMasterRxAutoSuspend
		LL_SPI_IsEnabledMasterRxAutoSuspend
	RCRCINI	LL_SPI_GetRxCRCInitPattern
		LL_SPI_SetRxCRCInitPattern
	SPE	LL_SPI_Disable
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	LL_SPI_GetInternalSSLevel
		LL_SPI_SetInternalSSLevel
	TCRCINI	LL_SPI_GetTxCRCInitPattern
		LL_SPI_SetTxCRCInitPattern
CR2	TSIZE	LL_SPI_GetTransferSize
		LL_SPI_SetTransferSize
CRCPOLY	CRCPOLY	LL_SPI_GetCRCPolynomial
		LL_SPI_SetCRCPolynomial
IER	CRCEIE	LL_SPI_DisableIT_CRCERR
		LL_SPI_EnableIT_CRCERR
		LL_SPI_IsEnabledIT_CRCERR
	DXPIE	LL_SPI_DisableIT_DXP
		LL_SPI_EnableIT_DXP
		LL_SPI_IsEnabledIT_DXP
	EOTIE	LL_SPI_DisableIT_EOT
		LL_SPI_EnableIT_EOT

Register	Field	Function
IER	EOTIE	LL_SPI_IsEnabledIT_EOT
	MODFIE	LL_SPI_DisableIT_MODF
		LL_SPI_EnableIT_MODF
		LL_SPI_IsEnabledIT_MODF
	OVRIE	LL_SPI_DisableIT_OVR
		LL_SPI_EnableIT_OVR
		LL_SPI_IsEnabledIT_OVR
	RXPIE	LL_SPI_DisableIT_RXP
		LL_SPI_EnableIT_RXP
		LL_SPI_IsEnabledIT_RXP
	TIFREIE	LL_SPI_DisableIT_FRE
		LL_SPI_EnableIT_FRE
		LL_SPI_IsEnabledIT_FRE
	TXPIE	LL_SPI_DisableIT_TXP
		LL_SPI_EnableIT_TXP
		LL_SPI_IsEnabledIT_TXP
	TXTFIE	LL_SPI_DisableIT_TXTF
		LL_SPI_EnableIT_TXTF
		LL_SPI_IsEnabledIT_TXTF
	UDRIE	LL_SPI_DisableIT_UDR
		LL_SPI_EnableIT_UDR
		LL_SPI_IsEnabledIT_UDR
IFCR	CRCEC	LL_SPI_ClearFlag_CRCERR
	EOTC	LL_SPI_ClearFlag_EOT
	MODFC	LL_SPI_ClearFlag_MODF
	OVRC	LL_SPI_ClearFlag_OVR
	SUSPC	LL_SPI_ClearFlag_SUSP
	TIFREC	LL_SPI_ClearFlag_FRE
	TXTFC	LL_SPI_ClearFlag_TXTF
	UDRC	LL_SPI_ClearFlag_UDR
RXCRCR	RXCRC	LL_SPI_GetRxCRC
RXDR	.	LL_SPI_ReceiveData16
		LL_SPI_ReceiveData32
		LL_SPI_ReceiveData8
	RXDR	LL_SPI_DMA_GetRxRegAddr
SR	CRCE	LL_SPI_IsActiveFlag_CRCERR
	CTSIZE	LL_SPI_GetRemainingDataFrames
	DXP	LL_SPI_IsActiveFlag_DXP
	EOT	LL_SPI_IsActiveFlag_EOT
	MODF	LL_SPI_IsActiveFlag_MODF
	OVR	LL_SPI_IsActiveFlag_OVR

Register	Field	Function
SR	RXP	LL_SPI_IsActiveFlag_RXP
	RXPLVL	LL_SPI_GetRxFIFOPackingLevel
	RXWNE	LL_SPI_IsActiveFlag_RXWNE
	SUSP	LL_SPI_IsActiveFlag_SUSP
	TIFRE	LL_SPI_IsActiveFlag_FRE
	TXC	LL_SPI_IsActiveFlag_TXC
	TXP	LL_SPI_IsActiveFlag_TXP
	TXTF	LL_SPI_IsActiveFlag_TXTF
	UDR	LL_SPI_IsActiveFlag_UDR
TXCRCR	TXCRC	LL_SPI_GetTxCRC
TXDR	.	LL_SPI_TransmitData16
		LL_SPI_TransmitData32
		LL_SPI_TransmitData8
	TXDR	LL_SPI_DMA_GetTxRegAddr
UDRDR	UDRDR	LL_SPI_GetUDRPattern
		LL_SPI_SetUDRPattern

119.28 SYSTEM

Table 52. Correspondence between SYSTEM registers and SYSTEM low-layer driver functions

Register	Field	Function
CCCSR	CS1	LL_SBS_GetVddCellCompensationCode
		LL_SBS_SetVddCellCompensationCode
	CS2	LL_SBS_GetVddIOCellCompensationCode
		LL_SBS_SetVddIOCellCompensationCode
	EN1	LL_SBS_DisableVddCompensationCell
		LL_SBS_EnableVddCompensationCell
		LL_SBS_IsEnabled_VddCompensationCell
	EN2	LL_SBS_DisableVddIOCompensationCell
		LL_SBS_EnableVddIOCompensationCell
		LL_SBS_IsEnabled_VddIOCompensationCell
	RDY1	LL_SBS_IsActiveFlag_VddCMPCR
		LL_SBS_IsActiveFlag_VddIOCMPCR
CCSWCR	NCC1	LL_SBS_GetNMOSVddCompensationCode
	NCC2	LL_SBS_GetNMOSVddIOCompensationCode
		LL_SBS_SetNMOSVddIOCompensationCode
	PCC1	LL_SBS_GetPMOSVddCompensationCode
		LL_SBS_SetPMOSVddCompensationCode
	PCC2	LL_SBS_GetPMOSVddIOCompensationCode
		LL_SBS_SetNMOSVddCompensationCode
		LL_SBS_SetPMOSVddIOCompensationCode

Register	Field	Function
CCVALR	NCV1	LL_SBS_GetNMOSVddCompensationValue
	NCV2	LL_SBS_GetNMOSVddIO2CompensationValue
	PCV1	LL_SBS_GetPMOSVddCompensationValue
	PCV2	LL_SBS_GetPMOSVddIO2CompensationValue
CFGR2	CLL	LL_SBS_GetTIMBreakInputs
		LL_SBS_SetTIMBreakInputs
	ECCL	LL_SBS_GetTIMBreakInputs
		LL_SBS_SetTIMBreakInputs
	PVDL	LL_SBS_GetTIMBreakInputs
		LL_SBS_SetTIMBreakInputs
	SEL	LL_SBS_GetTIMBreakInputs
		LL_SBS_SetTIMBreakInputs
CNSLCKR	LOCKNSMPU	LL_SBS_NonSecureLock
	LOCKNSVTOR	LL_SBS_NonSecureLock
CSLCKR	LOCKSAU	LL_SBS_GetSecureLock
		LL_SBS_SecureLock
	LOCKSMPU	LL_SBS_GetSecureLock
		LL_SBS_SecureLock
	LOCKSVTAIRCR	LL_SBS_GetSecureLock
		LL_SBS_SecureLock
ECCNMIR	SBS_ECCNMIR_ECCNMI_MASK_EN	LL_SBS_FLASH_DisableECCNMI
		LL_SBS_FLASH_EnableECCNMI
EPOCHSELCR	EPOCH_SEL	LL_SBS_EPOCHSelection
		LL_SBS_GetEPOCHSelection
FPUIMR	FPU_IE_0	LL_SBS_DisableIT_FPU_IOC
		LL_SBS_EnableIT_FPU_IOC
		LL_SBS_IsEnabledIT_FPU_IOC
	FPU_IE_1	LL_SBS_DisableIT_FPU_DZC
		LL_SBS_EnableIT_FPU_DZC
		LL_SBS_IsEnabledIT_FPU_DZC
	FPU_IE_2	LL_SBS_DisableIT_FPU_UFC
		LL_SBS_EnableIT_FPU_UFC
		LL_SBS_IsEnabledIT_FPU_UFC
	FPU_IE_3	LL_SBS_DisableIT_FPU_OFC
		LL_SBS_EnableIT_FPU_OFC
		LL_SBS_IsEnabledIT_FPU_OFC
	FPU_IE_4	LL_SBS_DisableIT_FPU_IDC
		LL_SBS_EnableIT_FPU_IDC
		LL_SBS_IsEnabledIT_FPU_IDC
	FPU_IE_5	LL_SBS_DisableIT_FPU_IXC
		LL_SBS_EnableIT_FPU_IXC

Register	Field	Function
FPUIMR	FPU_IE_5	LL_SBS_IsEnabledIT_FPU_IXC
HDPLCR	HDPL_INCR	LL_SBS_IncrementHDPLValue
HDPLSR	HDPL	LL_SBS_GetHDPLValue
MESR	IPMEE	LL_SBS_ClearEraseEndStatus
		LL_SBS_GetEraseEndStatus
	MCLR	LL_SBS_ClearEraseAfterResetStatus
		LL_SBS_GetEraseAfterResetStatus
NEXTHDPLCR	NEXTHDPL	LL_SBS_GetOBKHDPL
		LL_SBS_SetOBKHDPL
PMCR	EPIS_SEL	LL_SBS_GetPHYInterface
		LL_SBS_SetPHYInterface
	PBx_FMP	LL_SBS_DisableFastModePlus
		LL_SBS_EnableFastModePlus
SBS_DBGCR	AP_UNLOCK	LL_SBS_IsUnlockedAccessPort
		LL_SBS_UnlockAccessPort
	DBG_AUTH_HDPL	LL_SBS_GetAuthDbgHDPL
		LL_SBS_SetAuthDbgHDPL
	DBG_AUTH_SEC	LL_SBS_GetAuthDbgSec
		LL_SBS_SetAuthDbgSec
	DBG_UNLOCK	LL_SBS_IsUnlockedDebug
		LL_SBS_UnlockDebug
SBS_DBGLOCKR	DBGCFG_LOCK	LL_SBS_IsLockedDebugConfig
		LL_SBS_LockDebugConfig
SECCFGR	CLASSBSEC	LL_SBS_ConfigSecure
	FPUSEC	LL_SBS_ConfigSecure
	SBSSEC	LL_SBS_ConfigSecure
	SDCE_SEC_EN	LL_SBS_ConfigSecure

119.29 TIM

Table 53. Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
AF1	BKCMP1E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKCMP1P	LL_TIM_SetBreakInputSourcePolarity
	BKINE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKINP	LL_TIM_SetBreakInputSourcePolarity
	ETRSEL	LL_TIM_SetETRSource
AF2	BK2CMP1E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource

Register	Field	Function
AF2	BK2CMP1P	LL_TIM_SetBreakInputSourcePolarity
	BK2INE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BK2INP	LL_TIM_SetBreakInputSourcePolarity
ARR	ARR	LL_TIM_GetAutoReload
		LL_TIM_SetAutoReload
BDTR	AOE	LL_TIM_DisableAutomaticOutput
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BK2BID	LL_TIM_ConfigBRK2
	BK2DSRM	LL_TIM_DisarmBRK2
		LL_TIM_ReArmBRK2
	BK2E	LL_TIM_DisableBRK2
		LL_TIM_EnableBRK2
	BK2F	LL_TIM_ConfigBRK2
	BK2P	LL_TIM_ConfigBRK2
	BKBID	LL_TIM_ConfigBRK
	BKDSRM	LL_TIM_DisarmBRK
		LL_TIM_ReArmBRK
	BKE	LL_TIM_DisableBRK
		LL_TIM_EnableBRK
	BKF	LL_TIM_ConfigBRK
	BKP	LL_TIM_ConfigBRK
	DTG	LL_TIM_OC_SetDeadTime
	LOCK	LL_TIM_CC_SetLockLevel
	MOE	LL_TIM_DisableAllOutputs
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
	OSSI	LL_TIM_SetOffStates
	OSSR	LL_TIM_SetOffStates
CCER	CC1E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity

Register	Field	Function
CCER	CC1P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4E	LL_TIM_CC_DisableChannel

Register	Field	Function
CCER	CC4E	LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC4NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC4NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC5E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC5P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC6E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC6P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC2S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler

Register	Field	Function
CCMR1	IC1PSC	LL_TIM_IC_SetPrescaler
	IC2F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC1CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC1PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC2M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC2PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR2	CC3S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC4S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC3F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC3PSC	LL_TIM_IC_Config

Register	Field	Function
CCMR2	IC3PSC	LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC4F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC4PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC3CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC3FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC3M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC3PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC4FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC4PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR3	CC5S	LL_TIM_OC_ConfigOutput
	CC6S	LL_TIM_OC_ConfigOutput
	OC5CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC5FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC5M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC5PE	LL_TIM_OC_DisablePreload

Register	Field	Function
CCMR3	OC5PE	LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC6CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC6FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC6M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC6PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	LL_TIM_IC_GetCaptureCH1
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	LL_TIM_IC_GetCaptureCH2
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	LL_TIM_IC_GetCaptureCH3
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	LL_TIM_IC_GetCaptureCH4
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CCR5	CCR5	LL_TIM_OC_GetCompareCH5
		LL_TIM_OC_SetCompareCH5
	GC5C1	LL_TIM_SetCH5CombinedChannels
	GC5C2	LL_TIM_SetCH5CombinedChannels
	GC5C3	LL_TIM_SetCH5CombinedChannels
CCR6	CCR6	LL_TIM_OC_GetCompareCH6
		LL_TIM_OC_SetCompareCH6
CNT	CNT	LL_TIM_GetCounter
		LL_TIM_SetCounter
CR1	ARPE	LL_TIM_DisableARRPreload
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	LL_TIM_DisableCounter
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	LL_TIM_GetClockDivision

Register	Field	Function
CR1	CKD	LL_TIM_SetClockDivision
	CMS	LL_TIM_GetCounterMode
		LL_TIM_SetCounterMode
	DIR	LL_TIM_GetCounterMode
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	DITHEN	LL_TIM_DisableDithering
		LL_TIM_EnableDithering
		LL_TIM_IsEnabledDithering
	OPM	LL_TIM_GetOnePulseMode
		LL_TIM_SetOnePulseMode
	UDIS	LL_TIM_DisableUpdateEvent
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	UIFREMAP	LL_TIM_DisableUIFRemap
		LL_TIM_EnableUIFRemap
	URS	LL_TIM_GetUpdateSource
		LL_TIM_SetUpdateSource
CR2	CCDS	LL_TIM_CC_GetDMAReqTrigger
		LL_TIM_CC_SetDMAReqTrigger
	CCPC	LL_TIM_CC_DisablePreload
		LL_TIM_CC_EnablePreload
	CCUS	LL_TIM_CC_SetUpdate
	MMS	LL_TIM_SetTriggerOutput
	MMS2	LL_TIM_SetTriggerOutput2
	OIS1	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS4	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState

Register	Field	Function
CR2	OIS4N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS5	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS6	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	TI1S	LL_TIM_IC_DisableXORCombination
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	LL_TIM_ConfigDMABurst
	DBL	LL_TIM_ConfigDMABurst
DIER	BIE	LL_TIM_DisableIT_BRK
		LL_TIM_EnableIT_BRK
		LL_TIM_IsEnabledIT_BRK
	CC1DE	Section 114.2.1 LL_LPTIM_DisableDMAReq_CC1
		Section 114.2.1 LL_LPTIM_EnableDMAReq_CC1
		Section 114.2.1 LL_LPTIM_IsEnabledDMAReq_CC1
		LL_TIM_DisableDMAReq_CC1
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	LL_TIM_DisableIT_CC1
		LL_TIM_EnableIT_CC1
		LL_TIM_IsEnabledIT_CC1
	CC2DE	Section 114.2.1 LL_LPTIM_DisableDMAReq_CC2
		Section 114.2.1 LL_LPTIM_EnableDMAReq_CC2
		Section 114.2.1 LL_LPTIM_IsEnabledDMAReq_CC2
		LL_TIM_DisableDMAReq_CC2
		LL_TIM_EnableDMAReq_CC2
		LL_TIM_IsEnabledDMAReq_CC2
	CC2IE	LL_TIM_DisableIT_CC2
		LL_TIM_EnableIT_CC2
		LL_TIM_IsEnabledIT_CC2
	CC3DE	LL_TIM_DisableDMAReq_CC3
		LL_TIM_EnableDMAReq_CC3
		LL_TIM_IsEnabledDMAReq_CC3
	CC3IE	LL_TIM_DisableIT_CC3
		LL_TIM_EnableIT_CC3
		LL_TIM_IsEnabledIT_CC3
	CC4DE	LL_TIM_DisableDMAReq_CC4

Register	Field	Function
DIER	CC4DE	LL_TIM_EnableDMAReq_CC4
		LL_TIM_IsEnabledDMAReq_CC4
	CC4IE	LL_TIM_DisableIT_CC4
		LL_TIM_EnableIT_CC4
		LL_TIM_IsEnabledIT_CC4
	COMDE	LL_TIM_DisableDMAReq_COM
		LL_TIM_EnableDMAReq_COM
		LL_TIM_IsEnabledDMAReq_COM
	COMIE	LL_TIM_DisableIT_COM
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	DIRIE	LL_TIM_DisableIT_DIR
		LL_TIM_EnableIT_DIR
		LL_TIM_IsEnabledIT_DIR
	IDXIE	LL_TIM_DisableIT_IDX
		LL_TIM_EnableIT_IDX
		LL_TIM_IsEnabledIT_IDX
	IERRIE	LL_TIM_DisableIT_IERR
		LL_TIM_EnableIT_IERR
		LL_TIM_IsEnabledIT_IERR
	TDE	LL_TIM_DisableDMAReq_TRIG
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
	TERRIE	LL_TIM_DisableIT_TERR
		LL_TIM_EnableIT_TERR
		LL_TIM_IsEnabledIT_TERR
	TIE	LL_TIM_DisableIT_TRIG
		LL_TIM_EnableIT_TRIG
		LL_TIM_IsEnabledIT_TRIG
	UDE	LL_TIM_DisableDMAReq_UPDATE
		LL_TIM_EnableDMAReq_UPDATE
		LL_TIM_IsEnabledDMAReq_UPDATE
	UEDE	Section 114.2.1 LL_LPTIM_DisableDMAReq_UPDATE
		Section 114.2.1 LL_LPTIM_EnableDMAReq_UPDATE
		Section 114.2.1 LL_LPTIM_IsEnabledDMAReq_UPDATE
	UIE	LL_TIM_DisableIT_UPDATE
		LL_TIM_EnableIT_UPDATE
		LL_TIM_IsEnabledIT_UPDATE
DTR2	DTAE	LL_TIM_DisableAsymmetricalDeadTime
		LL_TIM_EnableAsymmetricalDeadTime
		LL_TIM_IsEnabledAsymmetricalDeadTime

Register	Field	Function
DTR2	DTGF	LL_TIM_GetFallingDeadTime
		LL_TIM_SetFallingDeadTime
	DTPE	LL_TIM_DisableDeadTimePreload
		LL_TIM_EnableDeadTimePreload
		LL_TIM_IsEnabledDeadTimePreload
ECR	FIDX	LL_TIM_ConfigIDX
		LL_TIM_DisableFirstIndex
		LL_TIM_EnableFirstIndex
		LL_TIM_IsEnabledFirstIndex
	IBLK	LL_TIM_ConfigIDX
		LL_TIM_GetIndexblanking
		LL_TIM_SetIndexblanking
	IDIR	LL_TIM_ConfigIDX
		LL_TIM_GetIndexDirection
		LL_TIM_SetIndexDirection
	IE	LL_TIM_DisableEncoderIndex
		LL_TIM_EnableEncoderIndex
		LL_TIM_IsEnabledEncoderIndex
	IPOS	LL_TIM_ConfigIDX
		LL_TIM_GetIndexPositionning
		LL_TIM_SetIndexPositionning
	PW	LL_TIM_OC_GetPulseWidth
		LL_TIM_OC_SetPulseWidth
	PWPRSC	LL_TIM_OC_GetPulseWidthPrescaler
		LL_TIM_OC_SetPulseWidthPrescaler
EGR	B2G	LL_TIM_GenerateEvent_BRK2
	BG	LL_TIM_GenerateEvent_BRK
	CC1G	LL_TIM_GenerateEvent_CC1
	CC2G	LL_TIM_GenerateEvent_CC2
	CC3G	LL_TIM_GenerateEvent_CC3
	CC4G	LL_TIM_GenerateEvent_CC4
	COMG	LL_TIM_GenerateEvent_COM
	TG	LL_TIM_GenerateEvent_TRIG
	UG	LL_TIM_GenerateEvent_UPDATE
PSC	PSC	LL_TIM_GetPrescaler
		LL_TIM_SetPrescaler
RCR	REP	LL_TIM_GetRepetitionCounter
		LL_TIM_SetRepetitionCounter
SMCR	ECE	LL_TIM_DisableExternalClock
		LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock

Register	Field	Function
SMCR	ECE	LL_TIM_SetClockSource
	ETF	LL_TIM_ConfigETR
	ETP	LL_TIM_ConfigETR
	ETPS	LL_TIM_ConfigETR
	MSM	LL_TIM_DisableMasterSlaveMode
		LL_TIM_EnableMasterSlaveMode
		LL_TIM_IsEnabledMasterSlaveMode
	OCCS	LL_TIM_SetOCRefClearInputSource
	SMS	LL_TIM_SetClockSource
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	SMSPE	LL_TIM_DisableSMSPreload
		LL_TIM_EnableSMSPreload
		LL_TIM_IsEnabledSMSPreload
	SMSPS	LL_TIM_GetSMSPreloadSource
		LL_TIM_SetSMSPreloadSource
	TS	LL_TIM_SetTriggerInput
SR	B2IF	LL_TIM_ClearFlag_BRK2
		LL_TIM_IsActiveFlag_BRK2
	BIF	LL_TIM_ClearFlag_BRK
		LL_TIM_IsActiveFlag_BRK
	CC1IF	LL_TIM_ClearFlag_CC1
		LL_TIM_IsActiveFlag_CC1
	CC1OF	LL_TIM_ClearFlag_CC1OVR
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	LL_TIM_ClearFlag_CC2
		LL_TIM_IsActiveFlag_CC2
	CC2OF	LL_TIM_ClearFlag_CC2OVR
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	LL_TIM_ClearFlag_CC3
		LL_TIM_IsActiveFlag_CC3
	CC3OF	LL_TIM_ClearFlag_CC3OVR
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	LL_TIM_ClearFlag_CC4
		LL_TIM_IsActiveFlag_CC4
	CC4OF	LL_TIM_ClearFlag_CC4OVR
		LL_TIM_IsActiveFlag_CC4OVR
	CC5IF	LL_TIM_ClearFlag_CC5
		LL_TIM_IsActiveFlag_CC5
	CC6IF	LL_TIM_ClearFlag_CC6
		LL_TIM_IsActiveFlag_CC6

Register	Field	Function
SR	COMIF	LL_TIM_ClearFlag_COM
		LL_TIM_IsActiveFlag_COM
	DIRF	LL_TIM_ClearFlag_DIR
		LL_TIM_IsActiveFlag_DIR
	IDXF	LL_TIM_ClearFlag_IDX
		LL_TIM_IsActiveFlag_IDX
	IERRF	LL_TIM_ClearFlag_IERR
		LL_TIM_IsActiveFlag_IERR
	SBIF	LL_TIM_ClearFlag_SYSBRK
		LL_TIM_IsActiveFlag_SYSBRK
	TERRF	LL_TIM_ClearFlag_TERR
		LL_TIM_IsActiveFlag_TERR
	TIF	LL_TIM_ClearFlag_TRIG
		LL_TIM_IsActiveFlag_TRIG
	UIF	LL_TIM_ClearFlag_UPDATE
		LL_TIM_IsActiveFlag_UPDATE
TIM12_TISEL	TI1SEL	LL_TIM_SetRemap
TIM13_TISEL	TI1SEL	LL_TIM_SetRemap
TIM14_TISEL	TI1SEL	LL_TIM_SetRemap
TIM15_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM16_TISEL	TI1SEL	LL_TIM_SetRemap
TIM17_TISEL	TI1SEL	LL_TIM_SetRemap
TIM1_TISEL	TI1SEL	LL_TIM_SetRemap
TIM2_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
	TI4SEL	LL_TIM_SetRemap
TIM3_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM4_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM5_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM8_TISEL	TI1SEL	LL_TIM_SetRemap

119.30 UCPD

Table 54. Correspondence between UCPD registers and UCPD low-layer driver functions

Register	Field	Function
CFG1	HBITCLKDIV	LL_UCPD_SetHbitClockDiv
	IFRGAP	LL_UCPD_SetIfrGap

Register	Field	Function
CFG1	RXDMAEN	LL_UCPD_RxDMADisable
		LL_UCPD_RxDMAEnable
	RXORDSETEN	LL_UCPD_SetRxOrderSet
	TRANSWIN	LL_UCPD_SetTransWin
	TXDMAEN	LL_UCPD_TxDMADisable
		LL_UCPD_TxDMAEnable
	UCPDCLK	LL_UCPD_SetPSCClk
	UCPDEN	LL_UCPD_Disable
		LL_UCPD_Enable
		LL_UCPD_IsEnabled
CFG2	FORCECLK	LL_UCPD_ForceClockDisable
		LL_UCPD_ForceClockEnable
	RXAFILTEN	LL_UCPD_RxAnalogFilterDisable
		LL_UCPD_RxAnalogFilterEnable
	RXFILTDIS	LL_UCPD_RxFilterDisable
		LL_UCPD_RxFilterEnable
	WUPEN	LL_UCPD_WakeUpDisable
		LL_UCPD_WakeUpEnable
CR	ANAMODE	LL_UCPD_GetRole
		LL_UCPD_SetSNKRole
		LL_UCPD_SetSRCRole
	ANASUBMODE	LL_UCPD_SetRpResistor
	CC1TCDIS	LL_UCPD_TypeCDetectionCC1Disable
		LL_UCPD_TypeCDetectionCC1Enable
	CC1VCONNEN	LL_UCPD_SetccEnable
	CC2TCDIS	LL_UCPD_TypeCDetectionCC2Disable
		LL_UCPD_TypeCDetectionCC2Enable
	FRSRXEN	LL_UCPD_FRSDetectionDisable
		LL_UCPD_FRSDetectionEnable
	FRSTX	LL_UCPD_VconnDischargeDisable
	PHYCCSEL	LL_UCPD_SetCCPin
	PHYRXEN	LL_UCPD_RxDisable
		LL_UCPD_RxEnable
	RDCH	LL_UCPD_VconnDischargeDisable
		LL_UCPD_VconnDischargeEnable
	RXMODE	LL_UCPD_SetRxMode
	TXHRST	LL_UCPD_SendHardReset
	TXMODE	LL_UCPD_SetTxMode
	TXSEND	LL_UCPD_SendMessage
CR2	RXDMAEN	LL_UCPD_IsEnabledRxDMA
	TXDMAEN	LL_UCPD_IsEnabledTxDMA

Register	Field	Function
ICR	FRSEVTIE	LL_UCPD_ClearFlag_FRS
	HRSTDISCIE	LL_UCPD_ClearFlag_TxHRSTDISC
	HRSTSENTIE	LL_UCPD_ClearFlag_TxHRSTSENT
	RXHRSTDETIE	LL_UCPD_ClearFlag_RxHRST
	RXMSGENDIE	LL_UCPD_ClearFlag_RxMsgEnd
	RXORDDETIE	LL_UCPD_ClearFlag_RxOrderSet
	RXOVRIE	LL_UCPD_ClearFlag_RxOvr
	TXMSGABTIE	LL_UCPD_ClearFlag_TxMSGABT
	TXMSGDISCIE	LL_UCPD_ClearFlag_TxMSGDISC
	TXMSGSENTIE	LL_UCPD_ClearFlag_TxMSGSENT
	TXUNDIE	LL_UCPD_ClearFlag_TxUND
IIMR	TYPECEVT1IE	LL_UCPD_ClearFlag_TypeCEventCC1
	TYPECEVT2IE	LL_UCPD_ClearFlag_TypeCEventCC2
IMR	FRSEVTIE	LL_UCPD_DisableIT_FRS
		LL_UCPD_EnableIT_FRS
	HRSTDISCIE	LL_UCPD_DisableIT_TxHRSTDISC
		LL_UCPD_EnableIT_TxHRSTDISC
		LL_UCPD_IsEnableIT_TxHRSTDISC
	HRSTSENTIE	LL_UCPD_DisableIT_TxHRSTSENT
		LL_UCPD_EnableIT_TxHRSTSENT
		LL_UCPD_IsEnableIT_TxHRSTSENT
	RXHRSTDETIE	LL_UCPD_DisableIT_RxHRST
		LL_UCPD_EnableIT_RxHRST
		LL_UCPD_IsEnableIT_RxHRST
	RXMSGENDIE	LL_UCPD_DisableIT_RxMsgEnd
		LL_UCPD_EnableIT_RxMsgEnd
		LL_UCPD_IsEnableIT_RxMsgEnd
	RXNEIE	LL_UCPD_DisableIT_RxNE
		LL_UCPD_EnableIT_RxNE
		LL_UCPD_IsEnableIT_RxNE
	RXORDDETIE	LL_UCPD_DisableIT_RxOrderSet
		LL_UCPD_EnableIT_RxOrderSet
		LL_UCPD_IsEnableIT_RxOrderSet
	RXOVRIE	LL_UCPD_DisableIT_RxOvr
		LL_UCPD_EnableIT_RxOvr
		LL_UCPD_IsEnableIT_RxOvr
	TXISIE	LL_UCPD_DisableIT_TxIS
		LL_UCPD_EnableIT_TxIS
		LL_UCPD_IsEnableIT_TxIS
	TXMSGABTIE	LL_UCPD_DisableIT_TxMSGABT
		LL_UCPD_EnableIT_TxMSGABT

Register	Field	Function
IMR	TXMSGABTIE	LL_UCPD_IsEnableIT_TxMSGABT
	TXMSGDISCIE	LL_UCPD_DisableIT_TxMSGDISC
		LL_UCPD_EnableIT_TxMSGDISC
		LL_UCPD_IsEnableIT_TxMSGDISC
	TXMSGSENTIE	LL_UCPD_DisableIT_TxMSGSENT
		LL_UCPD_EnableIT_TxMSGSENT
		LL_UCPD_IsEnableIT_TxMSGSENT
	TXUNDIE	LL_UCPD_DisableIT_TxUND
		LL_UCPD_EnableIT_TxUND
		LL_UCPD_IsEnableIT_TxUND
	TYPECEVT1IE	LL_UCPD_DisableIT_TypeCEventCC1
		LL_UCPD_EnableIT_TypeCEventCC1
	TYPECEVT2IE	LL_UCPD_DisableIT_TypeCEventCC2
		LL_UCPD_EnableIT_TypeCEventCC2
IMR2	TYPECEVT1IE	LL_UCPD_IsEnableIT_TypeCEventCC1
RXDR	RXDATA	LL_UCPD_ReadData
RX_ORDEXT1	RXSOPX1	LL_UCPD_SetRxOrdExt1
RX_ORDEXT2	RXSOPX2	LL_UCPD_SetRxOrdExt2
RX_ORDSET	RXORDSET	LL_UCPD_ReadRxOrderSet
RX_PAYSZ	RXPAYSZ	LL_UCPD_ReadRxPaySize
SR	FRSEVT	LL_UCPD_IsActiveFlag_FRS
	HRSTDISC	LL_UCPD_IsActiveFlag_TxHRSTDISC
	HRSTSENT	LL_UCPD_IsActiveFlag_TxHRSTSENT
	RXHRSTDET	LL_UCPD_IsActiveFlag_RxHRST
	RXMSGEND	LL_UCPD_IsActiveFlag_RxMsgEnd
	RXNE	LL_UCPD_IsActiveFlag_RxNE
	RXORDDET	LL_UCPD_IsActiveFlag_RxOrderSet
	RXOVR	LL_UCPD_IsActiveFlag_RxOvr
	TXIS	LL_UCPD_GetTypeCVstateCC1
		LL_UCPD_GetTypeCVstateCC2
		LL_UCPD_IsActiveFlag_TxIS
	TXMSGABT	LL_UCPD_IsActiveFlag_TxMSGABT
	TXMSGDISC	LL_UCPD_IsActiveFlag_TxMSGDISC
	TXMSGSENT	LL_UCPD_IsActiveFlag_TxMSGSENT
	TXUND	LL_UCPD_IsActiveFlag_TxUND
	TYPECEVT1	LL_UCPD_IsActiveFlag_TypeCEventCC1
	TYPECEVT2	LL_UCPD_IsActiveFlag_TypeCEventCC2
TXDR	DR	LL_UCPD_WriteData
TX_ORDSET	TXORDSET	LL_UCPD_WriteTxOrderSet
TX_PAYSZ	TXPAYSZ	LL_UCPD_WriteTxPaySize

119.31 USART

Table 55. **Correspondence between USART registers and USART low-layer driver functions**

Register	Field	Function
BRR	BRR	LL_USART_GetBaudRate
		LL_USART_SetBaudRate
CR1	CMIE	LL_USART_DisableIT_CM
		LL_USART_EnableIT_CM
		LL_USART_IsEnabledIT_CM
	DEAT	LL_USART_GetDEAssertionTime
		LL_USART_SetDEAssertionTime
	DEDT	LL_USART_GetDEDeassertionTime
		LL_USART_SetDEDeassertionTime
	EOBIE	LL_USART_DisableIT_EOB
		LL_USART_EnableIT_EOB
		LL_USART_IsEnabledIT_EOB
	FIFOEN	LL_USART_DisableFIFO
		LL_USART_EnableFIFO
		LL_USART_IsEnabledFIFO
	IDLEIE	LL_USART_DisableIT_IDLE
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M0	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	M1	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	MME	LL_USART_DisableMuteMode
		LL_USART_EnableMuteMode
		LL_USART_IsEnabledMuteMode
	OVER8	LL_USART_GetOverSampling
		LL_USART_SetOverSampling
	PCE	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity
	PEIE	LL_USART_DisableIT_PE
		LL_USART_EnableIT_PE
		LL_USART_IsEnabledIT_PE
	PS	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity

Register	Field	Function
CR1	RE	LL_USART_DisableDirectionRx
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	RTOIE	LL_USART_DisableIT_RTO
		LL_USART_EnableIT_RTO
		LL_USART_IsEnabledIT_RTO
	RXFFIE	LL_USART_DisableIT_RXFF
		LL_USART_EnableIT_RXFF
		LL_USART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_USART_DisableIT_RXNE_RXFNE
		LL_USART_EnableIT_RXNE_RXFNE
		LL_USART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_USART_DisableIT_TC
		LL_USART_EnableIT_TC
		LL_USART_IsEnabledIT_TC
	TE	LL_USART_DisableDirectionTx
		LL_USART_EnableDirectionTx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	TXEIE_TXFNFIE	LL_USART_DisableIT_TXE_TXFNF
		LL_USART_EnableIT_TXE_TXFNF
		LL_USART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_USART_DisableIT_TXFE
		LL_USART_EnableIT_TXFE
		LL_USART_IsEnabledIT_TXFE
	UE	LL_USART_Disable
		LL_USART_Enable
		LL_USART_IsEnabled
	UESM	LL_USART_DisableInStopMode
		LL_USART_EnableInStopMode
		LL_USART_IsEnabledInStopMode
	WAKE	LL_USART_GetWakeUpMethod
		LL_USART_SetWakeUpMethod
CR2	ABREN	LL_USART_DisableAutoBaudRate
		LL_USART_EnableAutoBaudRate
		LL_USART_IsEnabledAutoBaud
	ABRMODE	LL_USART_GetAutoBaudRateMode
		LL_USART_SetAutoBaudRateMode
	ADD	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddress

Register	Field	Function
CR2	ADDM7	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddressLen
	CLKEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSCLKOutput
		LL_USART_EnableSCLKOutput
		LL_USART_IsEnabledSCLKOutput
	CPHA	LL_USART_ConfigClock
		LL_USART_GetClockPhase
		LL_USART_SetClockPhase
	CPOL	LL_USART_ConfigClock
		LL_USART_GetClockPolarity
		LL_USART_SetClockPolarity
	DATAINV	LL_USART_GetBinaryDataLogic
		LL_USART_SetBinaryDataLogic
	DIS_NSS	LL_USART_DisableSPISlaveSelect
		LL_USART_EnableSPISlaveSelect
		LL_USART_IsEnabledSPISlaveSelect
	LBCL	LL_USART_ConfigClock
		LL_USART_GetLastClkPulseOutput
		LL_USART_SetLastClkPulseOutput
	LBDIE	LL_USART_DisableIT_LBD
		LL_USART_EnableIT_LBD
		LL_USART_IsEnabledIT_LBD
	LBDL	LL_USART_GetLINBrkDetectionLen
		LL_USART_SetLINBrkDetectionLen
	LINEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		LL_USART_EnableLIN
		LL_USART_IsEnabledLIN

Register	Field	Function
CR2	MSBFIRST	LL_USART_GetTransferBitOrder
		LL_USART_SetTransferBitOrder
	RTOEN	LL_USART_DisableRxTimeout
		LL_USART_EnableRxTimeout
		LL_USART_IsEnabledRxTimeout
	RXINV	LL_USART_GetRXPinLevel
		LL_USART_SetRXPinLevel
	SLVEN	LL_USART_DisableSPISlave
		LL_USART_EnableSPISlave
		LL_USART_IsEnabledSPISlave
	STOP	LL_USART_ConfigCharacter
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigSmartcardMode
		LL_USART_GetStopBitsLength
		LL_USART_SetStopBitsLength
	SWAP	LL_USART_GetTXRXSwap
		LL_USART_SetTXRXSwap
	TXINV	LL_USART_GetTXPinLevel
		LL_USART_SetTXPinLevel
CR3	CTSE	LL_USART_DisableCTSHWFlowCtrl
		LL_USART_EnableCTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	CTSIE	LL_USART_DisableIT_CTS
		LL_USART_EnableIT_CTS
		LL_USART_IsEnabledIT_CTS
	DDRE	LL_USART_DisableDMADeactOnRxErr
		LL_USART_EnableDMADeactOnRxErr
		LL_USART_IsEnabledDMADeactOnRxErr
	DEM	LL_USART_DisableDEMode
		LL_USART_EnableDEMode
		LL_USART_IsEnabledDEMode
	DEP	LL_USART_GetDESignalPolarity
		LL_USART_SetDESignalPolarity
	DMAR	LL_USART_DisableDMAReq_RX
		LL_USART_EnableDMAReq_RX
		LL_USART_IsEnabledDMAReq_RX
	DMAT	LL_USART_DisableDMAReq_TX
		LL_USART_EnableDMAReq_TX
		LL_USART_IsEnabledDMAReq_TX

Register	Field	Function
CR3	EIE	LL_USART_DisableIT_ERROR
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR
	HDSEL	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	LL_USART_GetIrdaPowerMode
		LL_USART_SetIrdaPowerMode
	NACK	LL_USART_DisableSmartcardNACK
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	ONEBIT	LL_USART_DisableOneBitSamp
		LL_USART_EnableOneBitSamp
		LL_USART_IsEnabledOneBitSamp
	OVRDIS	LL_USART_DisableOverrunDetect
		LL_USART_EnableOverrunDetect
		LL_USART_IsEnabledOverrunDetect
	RTSE	LL_USART_DisableRTSHWFlowCtrl
		LL_USART_EnableRTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	RXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetRXFIFOThreshold
		LL_USART_SetRXFIFOThreshold
	RXFTIE	LL_USART_DisableIT_RXFT

Register	Field	Function
CR3	RXFTIE	LL_USART_EnableIT_RXFT
		LL_USART_IsEnabledIT_RXFT
	SCARCNT	LL_USART_GetSmartcardAutoRetryCount
		LL_USART_SetSmartcardAutoRetryCount
	SCEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSmartcard
		LL_USART_EnableSmartcard
		LL_USART_IsEnabledSmartcard
	TCBGTIE	LL_USART_DisableIT_TCBGT
		LL_USART_EnableIT_TCBGT
		LL_USART_IsEnabledIT_TCBGT
	TXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetTXFIFOThreshold
		LL_USART_SetTXFIFOThreshold
	TXFTIE	LL_USART_DisableIT_TXFT
		LL_USART_EnableIT_TXFT
		LL_USART_IsEnabledIT_TXFT
	WUFIE	LL_USART_DisableIT_WKUP
		LL_USART_EnableIT_WKUP
		LL_USART_IsEnabledIT_WKUP
	WUS	LL_USART_GetWKUPType
		LL_USART_SetWKUPType
GTPR	GT	LL_USART_GetSmartcardGuardTime
		LL_USART_SetSmartcardGuardTime
	PSC	LL_USART_GetIrdaPrescaler
		LL_USART_GetSmartcardPrescaler
		LL_USART_SetIrdaPrescaler
		LL_USART_SetSmartcardPrescaler
ICR	CMCF	LL_USART_ClearFlag_CM
	CTSCF	LL_USART_ClearFlag_nCTS
	EOBCF	LL_USART_ClearFlag_EOB
	FECF	LL_USART_ClearFlag_FE
	IDLECF	LL_USART_ClearFlag_IDLE
	LBDCF	LL_USART_ClearFlag_LBD
	NECF	LL_USART_ClearFlag_NE

Register	Field	Function
ICR	ORECF	LL_USART_ClearFlag_ORE
	PECF	LL_USART_ClearFlag_PE
	RTOCF	LL_USART_ClearFlag_RTO
	TCBGTCF	LL_USART_ClearFlag_TCBGT
	TCCF	LL_USART_ClearFlag_TC
	TXFECF	LL_USART_ClearFlag_TXFE
	UDRCF	LL_USART_ClearFlag_UDR
	WUCF	LL_USART_ClearFlag_WKUP
ISR	ABRE	LL_USART_IsActiveFlag_ABRE
	ABRF	LL_USART_IsActiveFlag_ABR
	BUSY	LL_USART_IsActiveFlag_BUSY
	CMF	LL_USART_IsActiveFlag_CM
	CTS	LL_USART_IsActiveFlag_CTS
	CTSIF	LL_USART_IsActiveFlag_nCTS
	EOBF	LL_USART_IsActiveFlag_EOB
	FE	LL_USART_IsActiveFlag_FE
	IDLE	LL_USART_IsActiveFlag_IDLE
	LBDF	LL_USART_IsActiveFlag_LBD
	NE	LL_USART_IsActiveFlag_NE
	ORE	LL_USART_IsActiveFlag_ORE
	PE	LL_USART_IsActiveFlag_PE
	REACK	LL_USART_IsActiveFlag_REACK
	RTOF	LL_USART_IsActiveFlag_RTO
	RWU	LL_USART_IsActiveFlag_RWU
	RXFF	LL_USART_IsActiveFlag_RXFF
	RXFT	LL_USART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_USART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_USART_IsActiveFlag_SBK
	TC	LL_USART_IsActiveFlag_TC
	TCBGT	LL_USART_IsActiveFlag_TCBGT
	TEACK	LL_USART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_USART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_USART_IsActiveFlag_TXFE
	TXFT	LL_USART_IsActiveFlag_TXFT
	UDR	LL_USART_IsActiveFlag_UDR
	WUF	LL_USART_IsActiveFlag_WKUP
PRESC	PRESCALER	LL_USART_GetPrescaler
		LL_USART_SetPrescaler
RDR	RDR	LL_USART_DMA_GetRegAddr
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9

Register	Field	Function
RQR	ABRRQ	LL_USART_RequestAutoBaudRate
	MMRQ	LL_USART_RequestEnterMuteMode
	RXFRQ	LL_USART_RequestRxDataFlush
	SBKRQ	LL_USART_RequestBreakSending
	TXFRQ	LL_USART_RequestTxDataFlush
RTOR	BLEN	LL_USART_GetBlockLength
		LL_USART_SetBlockLength
	RTO	LL_USART_GetRxTimeout
		LL_USART_SetRxTimeout
TDR	TDR	LL_USART_DMA_GetRegAddr
		LL_USART_TransmitData8
		LL_USART_TransmitData9

119.32 WWDG

Table 56. Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	LL_WWDG_EnableIT_EWKUP
		LL_WWDG_IsEnabledIT_EWKUP
	W	LL_WWDG_GetWindow
		LL_WWDG_SetWindow
	WDGTB	LL_WWDG_GetPrescaler
		LL_WWDG_SetPrescaler
CR	T	LL_WWDG_GetCounter
		LL_WWDG_SetCounter
	WDGA	LL_WWDG_Enable
		LL_WWDG_IsEnabled
SR	EWIF	LL_WWDG_ClearFlag_EWKUP
		LL_WWDG_IsActiveFlag_EWKUP

120 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 Series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32H5 devices. To ensure compatibility between all devices and portability with others Series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *section* ***Devices supported by the HAL drivers***.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32h5xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32h5xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32h5xx_hal.h file has to be included.

What is the difference between xx_hal_ppp.c/.h and xx_hal_ppp_ *ex* .c/.h?

The HAL driver architecture supports common features across STM32 Series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32h5xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32h5xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32h5xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32h5xx_hal_msp.c. A template is provided in the HAL driver folders (stm32h5xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).

2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() in stm32h5xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32h5xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32h5xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32h5xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

Revision history

Table 57. **Document revision history**

Date	Revision	Changes
23-Feb-2023	1	Initial release.
21-Jun-2023	2	Updated description of HAL and low-layer drivers.

Contents

1 General information . . . 3
2 Acronyms and definitions . . . 4
3 Overview of HAL drivers . . . 6
3.1 HAL and user-application files . . . 7
3.1.1 HAL driver files . . . 7
3.1.2 User-application files . . . 7
3.2 HAL data structures . . . 9
3.2.1 Peripheral handle structures . . . 9
3.2.2 Initialization and configuration structure . . . 11
3.2.3 Specific process structures . . . 11
3.3 API classification . . . 12
3.4 Devices supported by HAL/LL drivers . . . 13
3.5 HAL driver rules . . . 16
3.5.1 HAL API naming rules . . . 16
3.5.2 HAL general naming rules . . . 17
3.5.3 HAL interrupt handler and callback functions . . . 18
3.6 HAL generic APIs . . . 18
3.7 HAL extension APIs . . . 19
3.7.1 HAL extension model overview . . . 19
3.7.2 HAL extension model cases . . . 20
3.8 File inclusion model . . . 22
3.9 HAL common resources . . . 23
3.10 HAL configuration . . . 23
3.11 HAL system peripheral handling . . . 24
3.11.1 Clocks . . . 24
3.11.2 GPIOs . . . 25
3.11.3 Cortex® NVIC and SysTick timer . . . 26
3.11.4 PWR . . . 26
3.11.5 EXTI . . . 27
3.11.6 DMA . . . 28
3.12 How to use HAL drivers . . . 30
3.12.1 HAL usage models . . . 30
3.12.2 HAL initialization . . . 31
3.12.3 HAL I/O operation process . . . 33
3.12.4 Timeout and error management . . . 36

4 Overview of low-layer drivers 40
4.1 Low-layer files 40
4.2 Overview of low-layer APIs and naming rules 42
4.2.1 Peripheral initialization functions 42
4.2.2 Peripheral register-level configuration functions 44
5 Cohabiting of HAL and LL 46
5.1 Low-layer driver used in Standalone mode 46
5.2 Mixed use of low-layer APIs and HAL drivers 46
6 HAL System Driver 47
6.1 HAL Firmware driver API description 47
6.1.1 How to use this driver 47
6.1.2 Initialization and de-initialization functions 47
6.1.3 HAL Control functions 47
6.1.4 HAL Debug functions 48
6.1.5 HAL VREFBUF Control functions 48
6.1.6 HAL SBS configuration functions 48
6.1.7 HAL SBS Boot control functions 49
6.1.8 HAL SBS Hardware secure storage control functions 49
6.1.9 SBS Debug control functions 49
6.1.10 SBS lock functions 50
6.1.11 SBS attributes functions 50
6.1.12 Detailed description of functions 50
6.2 HAL Firmware driver defines 66
6.2.1 HAL 66
7 HAL ADC Generic Driver 72
7.1 ADC Firmware driver registers structures 72
7.1.1 ADC_OversamplingTypeDef 72
7.1.2 ADC_InitTypeDef 72
7.1.3 ADC_ChannelConfTypeDef 74
7.1.4 ADC_AnalogWDGConfTypeDef 75
7.1.5 ADC_InjectionConfigTypeDef 76
7.1.6 __ADC_HandleTypeDef 77
7.2 ADC Firmware driver API description 78
7.2.1 ADC peripheral features 78
7.2.2 How to use this driver 78
7.2.3 Peripheral Control functions 81
7.2.4 Peripheral state and errors functions 81

7.2.5 Detailed description of functions . . . 82
7.3 ADC Firmware driver defines . . . 93
7.3.1 ADC . . . 94
8 HAL ADC Extension Driver . . . 122
8.1 ADCEx Firmware driver registers structures . . . 122
8.1.1 ADC_InjOversamplingTypeDef . . . 122
8.1.2 ADC_InjectionConfTypeDef . . . 122
8.1.3 ADC_MultiModeTypeDef . . . 124
8.2 ADCEx Firmware driver API description . . . 125
8.2.1 IO operation functions . . . 125
8.2.2 Peripheral Control functions . . . 125
8.2.3 Detailed description of functions . . . 126
8.3 ADCEx Firmware driver defines . . . 135
8.3.1 ADCEx . . . 135
9 HAL CEC Generic Driver . . . 137
9.1 CEC Firmware driver registers structures . . . 137
9.1.1 CEC_InitTypeDef . . . 137
9.1.2 __CEC_HandleTypeDef . . . 138
9.2 CEC Firmware driver API description . . . 139
9.2.1 How to use this driver . . . 139
9.2.2 Initialization and Configuration functions . . . 139
9.2.3 IO operation functions . . . 140
9.2.4 Peripheral Control function . . . 140
9.2.5 Detailed description of functions . . . 140
9.3 CEC Firmware driver defines . . . 145
9.3.1 CEC . . . 145
10 HAL COMP Generic Driver . . . 154
10.1 COMP Firmware driver registers structures . . . 154
10.1.1 COMP_InitTypeDef . . . 154
10.1.2 __COMP_HandleTypeDef . . . 154
10.2 COMP Firmware driver API description . . . 155
10.2.1 COMP Peripheral features . . . 155
10.2.2 How to use this driver . . . 155
10.2.3 Initialization and de-initialization functions . . . 157
10.2.4 IO operation functions . . . 157
10.2.5 Peripheral Control functions . . . 157
10.2.6 Peripheral State functions . . . 157

10.2.7 Detailed description of functions . . . 157
10.3 COMP Firmware driver defines . . . 162
10.3.1 COMP . . . 162
11 HAL CORDIC Generic Driver . . . 169
11.1 CORDIC Firmware driver registers structures . . . 169
11.1.1 __CORDIC_HandleTypeDef . . . 169
11.1.2 CORDIC_ConfigTypeDef . . . 170
11.2 CORDIC Firmware driver API description . . . 170
11.2.1 How to use this driver . . . 170
11.2.2 Initialization and de-initialization functions . . . 171
11.2.3 Peripheral Control functions . . . 171
11.2.4 Callback functions . . . 172
11.2.5 IRQ handler management . . . 172
11.2.6 Peripheral State functions . . . 172
11.2.7 Detailed description of functions . . . 172
11.3 CORDIC Firmware driver defines . . . 177
11.3.1 CORDIC . . . 177
12 HAL CORTEX Generic Driver . . . 183
12.1 CORTEX Firmware driver registers structures . . . 183
12.1.1 MPU_Region_InitTypeDef . . . 183
12.1.2 MPU_Attributes_InitTypeDef . . . 183
12.2 CORTEX Firmware driver API description . . . 184
12.2.1 How to use this driver . . . 184
12.2.2 NVIC functions . . . 185
12.2.3 SYSTICK functions . . . 185
12.2.4 MPU functions . . . 185
12.2.5 Detailed description of functions . . . 185
12.3 CORTEX Firmware driver defines . . . 192
12.3.1 CORTEX . . . 192
13 HAL CRC Generic Driver . . . 197
13.1 CRC Firmware driver registers structures . . . 197
13.1.1 CRC_InitTypeDef . . . 197
13.1.2 CRC_HandleTypeDef . . . 198
13.2 CRC Firmware driver API description . . . 198
13.2.1 How to use this driver . . . 198
13.2.2 Initialization and de-initialization functions . . . 198
13.2.3 Peripheral Control functions . . . 199

13.2.4 Peripheral State functions . . . 199
13.2.5 Detailed description of functions . . . 199
13.3 CRC Firmware driver defines . . . 201
13.3.1 CRC . . . 201
14 HAL CRC Extension Driver . . . 204
14.1 CRCEx Firmware driver API description . . . 204
14.1.1 How to use this driver . . . 204
14.1.2 Extended configuration functions . . . 204
14.1.3 Detailed description of functions . . . 204
14.2 CRCEx Firmware driver defines . . . 205
14.2.1 CRCEx . . . 205
15 HAL CRYP Generic Driver . . . 207
15.1 CRYP Firmware driver registers structures . . . 207
15.1.1 CRYP_ConfigTypeDef . . . 207
15.1.2 CRYP_ContextTypeDef . . . 208
15.1.3 __CRYP_HandleTypeDef . . . 209
15.2 CRYP Firmware driver API description . . . 210
15.2.1 How to use this driver . . . 210
15.2.2 Initialization, de-initialization and Set and Get configuration functions . . . 214
15.2.3 Encrypt Decrypt functions . . . 214
15.2.4 CRYP IRQ handler management . . . 215
15.2.5 Detailed description of functions . . . 215
15.3 CRYP Firmware driver defines . . . 222
15.3.1 CRYP . . . 222
16 HAL CRYP Extension Driver . . . 229
16.1 CRYPEx Firmware driver API description . . . 229
16.1.1 Extended AES processing functions . . . 229
16.1.2 Wrap and Unwrap key . . . 229
16.1.3 Encrypt and Decrypt Shared key functions . . . 229
16.1.4 Detailed description of functions . . . 229
17 HAL DAC Generic Driver . . . 232
17.1 DAC Firmware driver registers structures . . . 232
17.1.1 __DAC_HandleTypeDef . . . 232
17.1.2 DAC_SampleAndHoldConfTypeDef . . . 232
17.1.3 DAC_ChannelConfTypeDef . . . 233
17.2 DAC Firmware driver API description . . . 234
17.2.1 DAC Peripheral features . . . 234

17.2.2 How to use this driver . . . 236
17.2.3 Initialization and de-initialization functions . . . 237
17.2.4 IO operation functions . . . 238
17.2.5 Peripheral Control functions . . . 238
17.2.6 Peripheral State and Errors functions . . . 238
17.2.7 Detailed description of functions . . . 238
17.3 DAC Firmware driver defines . . . 246
17.3.1 DAC . . . 246
18 HAL DAC Extension Driver . . . 252
18.1 DACEx Firmware driver API description . . . 252
18.1.1 How to use this driver . . . 252
18.1.2 Extended features functions . . . 252
18.1.3 Peripheral Control functions . . . 253
18.1.4 Detailed description of functions . . . 253
18.2 DACEx Firmware driver defines . . . 259
18.2.1 DACEx . . . 259
19 HAL DCACHE Generic Driver . . . 262
19.1 DCACHE Firmware driver registers structures . . . 262
19.1.1 DCACHE_InitTypeDef . . . 262
19.1.2 __DCACHE_HandleTypeDef . . . 262
19.2 DCACHE Firmware driver API description . . . 262
19.2.1 How to use this driver . . . 263
19.2.2 Initialization and de-initialization functions . . . 263
19.2.3 IO operation functions . . . 264
19.2.4 Peripheral State . . . 265
19.2.5 Detailed description of functions . . . 265
19.3 DCACHE Firmware driver defines . . . 276
19.3.1 DCACHE . . . 276
20 HAL DCMI Generic Driver . . . 280
20.1 DCMI Firmware driver registers structures . . . 280
20.1.1 DCMI_CodesInitTypeDef . . . 280
20.1.2 DCMI_SyncUnmaskTypeDef . . . 280
20.1.3 DCMI_InitTypeDef . . . 280
20.1.4 __DCMI_HandleTypeDef . . . 281
20.2 DCMI Firmware driver API description . . . 282
20.2.1 How to use this driver . . . 282
20.2.2 Initialization and Configuration functions . . . 283

20.2.3 IO operation functions . . . 283
20.2.4 Peripheral Control functions . . . 283
20.2.5 Peripheral State and Errors functions . . . 284
20.2.6 Detailed description of functions . . . 284
20.3 DCMI Firmware driver defines . . . 290
20.3.1 DCMI . . . 290
21 HAL DMA Generic Driver . . . 297
21.1 DMA Firmware driver registers structures . . . 297
21.1.1 DMA_InitTypeDef . . . 297
21.1.2 DMA_InitLinkedListTypeDef . . . 298
21.1.3 __DMA_HandleTypeDef . . . 298
21.2 DMA Firmware driver API description . . . 299
21.2.1 . . . 299
21.2.2 How to use this driver . . . 299
21.2.3 . . . 299
21.2.4 . . . 300
21.2.5 Initialization and de-initialization functions . . . 300
21.2.6 . . . 300
21.2.7 . . . 301
21.2.8 IO operation functions . . . 301
21.2.9 . . . 301
21.2.10 . . . 302
21.2.11 State and Errors functions . . . 302
21.2.12 . . . 302
21.2.13 . . . 302
21.2.14 DMA Attributes functions . . . 303
21.2.15 . . . 303
21.2.16 Detailed description of functions . . . 303
21.3 DMA Firmware driver defines . . . 308
21.3.1 DMA . . . 308
22 HAL DMA Extension Driver . . . 330
22.1 DMAEx Firmware driver registers structures . . . 330
22.1.1 DMA_DataHandlingConfTypeDef . . . 330
22.1.2 DMA_TriggerConfTypeDef . . . 330
22.1.3 DMA_RepeatBlockConfTypeDef . . . 330
22.1.4 DMA_NodeConfTypeDef . . . 331
22.1.5 DMA_NodeTypeDef . . . 332
22.1.6 __DMA_QListTypeDef . . . 332

22.1.7 DMA_NodeInQInfoTypeDef . . . 332
22.2 DMAEx Firmware driver API description . . . 332
22.2.1 . . . 333
22.2.2 How to use this driver . . . 333
22.2.3 . . . 333
22.2.4 . . . 333
22.2.5 Linked-List Initialization and De-Initialization Functions . . . 333
22.2.6 . . . 333
22.2.7 . . . 333
22.2.8 Linked-List IO Operation Functions . . . 333
22.2.9 . . . 333
22.2.10 . . . 334
22.2.11 Linked-List Management Functions . . . 334
22.2.12 . . . 334
22.2.13 . . . 334
22.2.14 Data handling, repeated block and trigger configuration functions . . . 334
22.2.15 . . . 334
22.2.16 . . . 334
22.2.17 Suspend and resume operation functions . . . 335
22.2.18 . . . 335
22.2.19 . . . 335
22.2.20 Fifo status function . . . 335
22.2.21 . . . 335
22.2.22 Detailed description of functions . . . 335
22.3 DMAEx Firmware driver defines . . . 345
22.3.1 DMAEx . . . 345
23 HAL DTS Generic Driver . . . 352
23.1 DTS Firmware driver registers structures . . . 352
23.1.1 DTS_InitTypeDef . . . 352
23.1.2 __DTS_HandleTypeDef . . . 352
23.2 DTS Firmware driver API description . . . 353
23.2.1 DTS Peripheral features . . . 353
23.2.2 How to use this driver . . . 354
23.2.3 Initialization and de-initialization functions . . . 355
23.2.4 DTS Start Stop operation functions . . . 355
23.2.5 Peripheral State functions . . . 355
23.2.6 Detailed description of functions . . . 355
23.3 DTS Firmware driver defines . . . 361

23.3.1 DTS . . . 361
24 HAL ETH Generic Driver . . . 367
24.1 ETH Firmware driver registers structures . . . 367
24.1.1 ETH_DMADescTypeDef . . . 367
24.1.2 __ETH_BufferTypeDef . . . 367
24.1.3 ETH_TxDescListTypeDef . . . 367
24.1.4 ETH_TxPacketConfig . . . 368
24.1.5 ETH_TimeStampTypeDef . . . 369
24.1.6 ETH_RxDescListTypeDef . . . 369
24.1.7 ETH_MACConfigTypeDef . . . 369
24.1.8 ETH_DMAConfigTypeDef . . . 372
24.1.9 ETH_InitTypeDef . . . 373
24.1.10 __ETH_HandleTypeDef . . . 373
24.1.11 ETH_MACFilterConfigTypeDef . . . 374
24.1.12 ETH_PowerDownConfigTypeDef . . . 375
24.2 ETH Firmware driver API description . . . 376
24.2.1 How to use this driver . . . 376
24.2.2 Initialization and Configuration functions . . . 377
24.2.3 IO operation functions . . . 377
24.2.4 Peripheral Control functions . . . 378
24.2.5 Peripheral State and Errors functions . . . 378
24.2.6 Detailed description of functions . . . 378
24.3 ETH Firmware driver defines . . . 393
24.3.1 ETH . . . 393
25 HAL ETH Extension Driver . . . 415
25.1 ETHEx Firmware driver registers structures . . . 415
25.1.1 ETH_RxVLANConfigTypeDef . . . 415
25.1.2 ETH_TxVLANConfigTypeDef . . . 415
25.1.3 ETH_L3FilterConfigTypeDef . . . 416
25.1.4 ETH_L4FilterConfigTypeDef . . . 416
25.2 ETHEx Firmware driver API description . . . 417
25.2.1 Extended features functions . . . 417
25.2.2 Detailed description of functions . . . 417
25.3 ETHEx Firmware driver defines . . . 423
25.3.1 ETHEx . . . 423
26 HAL EXTI Generic Driver . . . 426
26.1 EXTI Firmware driver registers structures . . . 426

26.1.1 EXTI_HandleTypeDef . . . 426
26.1.2 EXTI_ConfigTypeDef . . . 426
26.2 EXTI Firmware driver API description . . . 426
26.2.1 EXTI Peripheral features . . . 426
26.2.2 How to use this driver . . . 427
26.2.3 EXTI attributes functions . . . 427
26.2.4 Configuration functions . . . 427
26.2.5 Detailed description of functions . . . 427
26.3 EXTI Firmware driver defines . . . 431
26.3.1 EXTI . . . 431
27 HAL FDCAN Generic Driver . . . 435
27.1 FDCAN Firmware driver registers structures . . . 435
27.1.1 FDCAN_InitTypeDef . . . 435
27.1.2 FDCAN_FilterTypeDef . . . 436
27.1.3 FDCAN_TxHeaderTypeDef . . . 436
27.1.4 FDCAN_RxHeaderTypeDef . . . 437
27.1.5 FDCAN_TxEventFifoTypeDef . . . 438
27.1.6 FDCAN_HpMsgStatusTypeDef . . . 439
27.1.7 FDCAN_ProtocolStatusTypeDef . . . 439
27.1.8 FDCAN_ErrorCountersTypeDef . . . 440
27.1.9 FDCAN_MsgRamAddressTypeDef . . . 441
27.1.10 __FDCAN_HandleTypeDef . . . 441
27.2 FDCAN Firmware driver API description . . . 443
27.2.1 How to use this driver . . . 443
27.2.2 Initialization and de-initialization functions . . . 444
27.2.3 Configuration functions . . . 444
27.2.4 Control functions . . . 445
27.2.5 Interrupts management . . . 446
27.2.6 Callback functions . . . 446
27.2.7 Peripheral State functions . . . 447
27.2.8 Detailed description of functions . . . 447
27.3 FDCAN Firmware driver defines . . . 469
27.3.1 FDCAN . . . 469
28 HAL FLASH Generic Driver . . . 482
28.1 FLASH Firmware driver registers structures . . . 482
28.1.1 FLASH_ProcessTypeDef . . . 482
28.2 FLASH Firmware driver API description . . . 482

28.2.1 FLASH peripheral features . . . 482
28.2.2 How to use this driver . . . 482
28.2.3 Programming operation functions . . . 483
28.2.4 Peripheral Control functions . . . 483
28.2.5 Peripheral Errors functions . . . 483
28.2.6 Detailed description of functions . . . 484
28.3 FLASH Firmware driver defines . . . 487
28.3.1 FLASH . . . 487
29 HAL FLASH Extension Driver . . . 511
29.1 FLASHEx Firmware driver registers structures . . . 511
29.1.1 FLASH_EraseInitTypeDef . . . 511
29.1.2 FLASH_OBProgramInitTypeDef . . . 511
29.1.3 FLASH_BBAttributesTypeDef . . . 512
29.1.4 FLASH_OperationTypeDef . . . 513
29.1.5 FLASH_HDPExtensionTypeDef . . . 513
29.2 FLASHEx Firmware driver API description . . . 513
29.2.1 Flash Extension features . . . 513
29.2.2 How to use this driver . . . 514
29.2.3 Extended programming operation functions . . . 514
29.2.4 Detailed description of functions . . . 515
29.3 FLASHEx Firmware driver defines . . . 519
29.3.1 FLASHEx . . . 519
30 HAL FMAC Generic Driver . . . 522
30.1 FMAC Firmware driver registers structures . . . 522
30.1.1 __FMAC_HandleTypeDef . . . 522
30.1.2 FMAC_FilterConfigTypeDef . . . 523
30.2 FMAC Firmware driver API description . . . 525
30.2.1 How to use this driver . . . 525
30.2.2 Callback registration . . . 527
30.2.3 Initialization and de-initialization functions . . . 527
30.2.4 Peripheral Control functions . . . 528
30.2.5 Callback functions . . . 528
30.2.6 IRQ handler management . . . 528
30.2.7 Peripheral State and Error functions . . . 528
30.2.8 Detailed description of functions . . . 529
30.3 FMAC Firmware driver defines . . . 537
30.3.1 FMAC . . . 537

31 HAL GPIO Generic Driver . . . 543
31.1 GPIO Firmware driver registers structures . . . 543
31.1.1 GPIO_InitTypeDef . . . 543
31.2 GPIO Firmware driver API description . . . 543
31.2.1 GPIO Peripheral features . . . 543
31.2.2 How to use this driver . . . 544
31.2.3 Initialization and de-initialization functions . . . 544
31.2.4 IO attributes functions . . . 544
31.2.5 Detailed description of functions . . . 544
31.3 GPIO Firmware driver defines . . . 549
31.3.1 GPIO . . . 549
32 HAL GPIO Extension Driver . . . 554
32.1 GPIOEx Firmware driver defines . . . 554
32.1.1 GPIOEx . . . 554
33 HAL GTZC Generic Driver . . . 559
33.1 GTZC Firmware driver registers structures . . . 559
33.1.1 MPCBB_Attribute_ConfigTypeDef . . . 559
33.1.2 MPCBB_ConfigTypeDef . . . 559
33.1.3 MPCWM_ConfigTypeDef . . . 559
33.2 GTZC Firmware driver API description . . . 560
33.2.1 GTZC main features . . . 560
33.2.2 How to use this driver . . . 560
33.2.3 Detailed description of functions . . . 561
33.3 GTZC Firmware driver defines . . . 567
33.3.1 GTZC . . . 567
34 HAL HASH Generic Driver . . . 573
34.1 HASH Firmware driver registers structures . . . 573
34.1.1 HASH_ConfigTypeDef . . . 573
34.1.2 __HASH_HandleTypeDef . . . 573
34.2 HASH Firmware driver API description . . . 574
34.2.1 How to use this driver . . . 574
34.2.2 Initialization and de-initialization functions . . . 575
34.2.3 HASH processing functions . . . 576
34.2.4 HMAC processing functions . . . 576
34.2.5 HASH IRQ handler management . . . 577
34.2.6 Detailed description of functions . . . 577
34.3 HASH Firmware driver defines . . . 587

34.3.1 HASH . . . 587
35 HAL HCD Generic Driver . . . 592
35.1 HCD Firmware driver registers structures . . . 592
35.1.1 __HCD_HandleTypeDef . . . 592
35.2 HCD Firmware driver API description . . . 593
35.2.1 How to use this driver . . . 593
35.2.2 Initialization and de-initialization functions . . . 593
35.2.3 IO operation functions . . . 594
35.2.4 Peripheral Control functions . . . 594
35.2.5 Peripheral State functions . . . 594
35.2.6 Detailed description of functions . . . 595
35.3 HCD Firmware driver defines . . . 606
35.3.1 HCD . . . 606
36 HAL I2C Generic Driver . . . 608
36.1 I2C Firmware driver registers structures . . . 608
36.1.1 I2C_InitTypeDef . . . 608
36.1.2 __I2C_HandleTypeDef . . . 608
36.2 I2C Firmware driver API description . . . 610
36.2.1 How to use this driver . . . 610
36.2.2 Initialization and de-initialization functions . . . 615
36.2.3 IO operation functions . . . 616
36.2.4 Peripheral State, Mode and Error functions . . . 617
36.2.5 Detailed description of functions . . . 618
36.3 I2C Firmware driver defines . . . 636
36.3.1 I2C . . . 636
37 HAL I2C Extension Driver . . . 643
37.1 I2CEx Firmware driver API description . . . 643
37.1.1 I2C peripheral Extended features . . . 643
37.1.2 How to use this driver . . . 643
37.1.3 Filter Mode Functions . . . 643
37.1.4 WakeUp Mode Functions . . . 643
37.1.5 Fast Mode Plus Functions . . . 643
37.1.6 Detailed description of functions . . . 644
37.2 I2CEx Firmware driver defines . . . 645
37.2.1 I2CEx . . . 645
38 HAL I2S Generic Driver . . . 646
38.1 I2S Firmware driver registers structures . . . 646

38.1.1 I2S_InitTypeDef . . . 646
38.1.2 __I2S_HandleTypeDef . . . 646
38.2 I2S Firmware driver API description . . . 648
38.2.1 How to use this driver . . . 648
38.2.2 Initialization and de-initialization functions . . . 650
38.2.3 IO operation functions . . . 651
38.2.4 Peripheral State and Errors functions . . . 652
38.2.5 Detailed description of functions . . . 652
38.3 I2S Firmware driver defines . . . 661
38.3.1 I2S . . . 661
39 HAL I3C Generic Driver . . . 667
39.1 I3C Firmware driver registers structures . . . 667
39.1.1 I3C_InitTypeDef . . . 667
39.1.2 I3C_FifoConfTypeDef . . . 667
39.1.3 I3C_CtrlConfTypeDef . . . 667
39.1.4 I3C_TgtConfTypeDef . . . 668
39.1.5 I3C_DeviceConfTypeDef . . . 669
39.1.6 I3C_CCCInfoTypeDef . . . 670
39.1.7 I3C_ControlTypeDef . . . 671
39.1.8 I3C_DataTypeDef . . . 671
39.1.9 I3C_CCCTypeDef . . . 671
39.1.10 I3C_PrivateTypeDef . . . 671
39.1.11 I3C_XferTypeDef . . . 672
39.1.12 __I3C_HandleTypeDef . . . 672
39.2 I3C Firmware driver API description . . . 674
39.2.1 How to use this driver . . . 674
39.2.2 Detailed description of functions . . . 679
39.3 I3C Firmware driver defines . . . 704
39.3.1 I3C . . . 704
40 HAL ICACHE Generic Driver . . . 714
40.1 ICACHE Firmware driver registers structures . . . 714
40.1.1 ICACHE_RegionConfigTypeDef . . . 714
40.2 ICACHE Firmware driver API description . . . 714
40.2.1 ICACHE main features . . . 714
40.2.2 How to use this driver . . . 714
40.2.3 Initialization and control functions . . . 715
40.2.4 IRQ and callback functions . . . 715

40.2.5 Memory remapped regions functions . . . 715
40.2.6 Detailed description of functions . . . 715
40.3 ICACHE Firmware driver defines . . . 720
40.3.1 ICACHE . . . 720
41 HAL IRDA Generic Driver . . . 724
41.1 IRDA Firmware driver registers structures . . . 724
41.1.1 IRDA_InitTypeDef . . . 724
41.1.2 __IRDA_HandleTypeDef . . . 724
41.2 IRDA Firmware driver API description . . . 726
41.2.1 How to use this driver . . . 726
41.2.2 Callback registration . . . 727
41.2.3 Initialization and Configuration functions . . . 728
41.2.4 IO operation functions . . . 729
41.2.5 Peripheral State and Error functions . . . 730
41.2.6 Detailed description of functions . . . 730
41.3 IRDA Firmware driver defines . . . 741
41.3.1 IRDA . . . 741
42 HAL IRDA Extension Driver . . . 752
42.1 IRDAEx Firmware driver defines . . . 752
42.1.1 IRDAEx . . . 752
43 HAL IWDG Generic Driver . . . 753
43.1 IWDG Firmware driver registers structures . . . 753
43.1.1 IWDG_InitTypeDef . . . 753
43.1.2 __IWDG_HandleTypeDef . . . 753
43.2 IWDG Firmware driver API description . . . 753
43.2.1 IWDG Generic features . . . 753
43.2.2 How to use this driver . . . 754
43.2.3 Initialization and Start functions . . . 754
43.2.4 IO operation functions . . . 755
43.2.5 Detailed description of functions . . . 755
43.3 IWDG Firmware driver defines . . . 757
43.3.1 IWDG . . . 757
44 HAL LPTIM Generic Driver . . . 760
44.1 LPTIM Firmware driver registers structures . . . 760
44.1.1 LPTIM_ClockConfigTypeDef . . . 760
44.1.2 LPTIM_ULPClockConfigTypeDef . . . 760
44.1.3 LPTIM_TriggerConfigTypeDef . . . 760

44.1.4 LPTIM_InitTypeDef . . . 760
44.1.5 LPTIM_OC_ConfigTypeDef . . . 761
44.1.6 LPTIM_IC_ConfigTypeDef . . . 761
44.1.7 __LPTIM_HandleTypeDef . . . 762
44.2 LPTIM Firmware driver API description . . . 763
44.2.1 How to use this driver . . . 763
44.2.2 Initialization and de-initialization functions . . . 765
44.2.3 LPTIM Start Stop operation functions . . . 765
44.2.4 LPTIM Read operation functions . . . 766
44.2.5 LPTIM Config function . . . 766
44.2.6 Peripheral State functions . . . 767
44.2.7 Detailed description of functions . . . 767
44.3 LPTIM Firmware driver defines . . . 785
44.3.1 LPTIM . . . 785
45 HAL MMC Generic Driver . . . 797
45.1 MMC Firmware driver registers structures . . . 797
45.1.1 HAL_MMC_CardInfoTypeDef . . . 797
45.1.2 __MMC_HandleTypeDef . . . 797
45.1.3 HAL_MMC_CardCSDTypeDef . . . 798
45.1.4 HAL_MMC_CardCIDTypeDef . . . 801
45.2 MMC Firmware driver API description . . . 801
45.2.1 How to use this driver . . . 801
45.2.2 Initialization and de-initialization functions . . . 804
45.2.3 IO operation functions . . . 804
45.2.4 Peripheral Control functions . . . 805
45.2.5 Detailed description of functions . . . 805
45.3 MMC Firmware driver defines . . . 817
45.3.1 MMC . . . 817
46 HAL MMC Extension Driver . . . 828
46.1 MMCEx Firmware driver API description . . . 828
46.1.1 How to use this driver . . . 828
46.1.2 Linked List management functions . . . 828
46.1.3 Detailed description of functions . . . 828
46.2 MMCEx Firmware driver defines . . . 831
46.2.1 MMCEx . . . 831
47 HAL NAND Generic Driver . . . 832
47.1 NAND Firmware driver registers structures . . . 832

47.1.1 NAND_IDTypeDef . . . 832
47.1.2 NAND_AddressTypeDef . . . 832
47.1.3 NAND_DeviceConfigTypeDef . . . 832
47.1.4 __NAND_HandleTypeDef . . . 833
47.2 NAND Firmware driver API description . . . 833
47.2.1 How to use this driver . . . 833
47.2.2 NAND Initialization and de-initialization functions . . . 834
47.2.3 NAND Input and Output functions . . . 834
47.2.4 NAND Control functions . . . 835
47.2.5 NAND State functions . . . 835
47.2.6 Detailed description of functions . . . 835
47.3 NAND Firmware driver defines . . . 843
47.3.1 NAND . . . 843
48 HAL NOR Generic Driver . . . 844
48.1 NOR Firmware driver registers structures . . . 844
48.1.1 NOR_IDTypeDef . . . 844
48.1.2 NOR_CFITypeDef . . . 844
48.1.3 __NOR_HandleTypeDef . . . 844
48.2 NOR Firmware driver API description . . . 845
48.2.1 How to use this driver . . . 845
48.2.2 NOR Initialization and de_initialization functions . . . 846
48.2.3 NOR Input and Output functions . . . 846
48.2.4 NOR Control functions . . . 846
48.2.5 NOR State functions . . . 846
48.2.6 Detailed description of functions . . . 847
48.3 NOR Firmware driver defines . . . 852
48.3.1 NOR . . . 852
49 HAL OPAMP Generic Driver . . . 854
49.1 OPAMP Firmware driver registers structures . . . 854
49.1.1 OPAMP_InitTypeDef . . . 854
49.1.2 __OPAMP_HandleTypeDef . . . 855
49.2 OPAMP Firmware driver API description . . . 855
49.2.1 OPAMP Peripheral Features . . . 855
49.2.2 How to use this driver . . . 856
49.2.3 Initialization and de-initialization functions . . . 857
49.2.4 IO operation functions . . . 858
49.2.5 Peripheral Control functions . . . 858

49.2.6 Peripheral State functions . . . 858
49.2.7 Detailed description of functions . . . 858
49.3 OPAMP Firmware driver defines . . . 862
49.3.1 OPAMP . . . 862
50 HAL OPAMP Extension Driver . . . 865
50.1 OPAMPEx Firmware driver API description . . . 865
50.1.1 Peripheral Control functions . . . 865
50.1.2 Detailed description of functions . . . 865
51 HAL OTFDEC Generic Driver . . . 866
51.1 OTFDEC Firmware driver registers structures . . . 866
51.1.1 OTFDEC_RegionConfigTypeDef . . . 866
51.1.2 __OTFDEC_HandleTypeDef . . . 866
51.2 OTFDEC Firmware driver API description . . . 866
51.2.1 How to use this driver . . . 867
51.2.2 Initialization and de-initialization functions . . . 868
51.2.3 OTFDEC IRQ handler management . . . 868
51.2.4 Peripheral Control functions . . . 868
51.2.5 Peripheral State functions . . . 868
51.2.6 Detailed description of functions . . . 868
51.3 OTFDEC Firmware driver defines . . . 876
51.3.1 OTFDEC . . . 876
52 HAL PCD Generic Driver . . . 881
52.1 PCD Firmware driver registers structures . . . 881
52.1.1 __PCD_HandleTypeDef . . . 881
52.2 PCD Firmware driver API description . . . 882
52.2.1 How to use this driver . . . 883
52.2.2 Initialization and de-initialization functions . . . 883
52.2.3 IO operation functions . . . 883
52.2.4 Peripheral Control functions . . . 884
52.2.5 Peripheral State functions . . . 884
52.2.6 Detailed description of functions . . . 884
52.3 PCD Firmware driver defines . . . 898
52.3.1 PCD . . . 898
53 HAL PCD Extension Driver . . . 900
53.1 PCDEx Firmware driver API description . . . 900
53.1.1 Extended features functions . . . 900
53.1.2 Detailed description of functions . . . 900

54 HAL PKA Generic Driver . . . 903
54.1 PKA Firmware driver registers structures . . . 903
54.1.1 __PKA_HandleTypeDef . . . 903
54.1.2 PKA_ECCMulInTypeDef . . . 903
54.1.3 PKA_PointCheckInTypeDef . . . 904
54.1.4 PKA_RSACRTExpInTypeDef . . . 904
54.1.5 PKA_ECDSAVerifInTypeDef . . . 905
54.1.6 PKA_ECDSASignInTypeDef . . . 906
54.1.7 PKA_ECDSASignOutTypeDef . . . 906
54.1.8 PKA_ECDSASignOutExtParamTypeDef . . . 907
54.1.9 PKA_ECCCompleteAdditionOutTypeDef . . . 907
54.1.10 PKA_ModExpInTypeDef . . . 907
54.1.11 PKA_ModExpProtectModeInTypeDef . . . 908
54.1.12 PKA_ModExpFastModeInTypeDef . . . 908
54.1.13 PKA_MontgomeryParamInTypeDef . . . 908
54.1.14 PKA_AddInTypeDef . . . 909
54.1.15 PKA_ModInvInTypeDef . . . 909
54.1.16 PKA_ModRedInTypeDef . . . 909
54.1.17 PKA_ModAddInTypeDef . . . 909
54.1.18 PKA_ECCDoubleBaseLadderInTypeDef . . . 910
54.1.19 PKA_ECCProjective2AffineInTypeDef . . . 911
54.1.20 PKA_ECCCompleteAdditionInTypeDef . . . 911
54.2 PKA Firmware driver API description . . . 912
54.2.1 How to use this driver . . . 912
54.2.2 Initialization and de-initialization functions . . . 915
54.2.3 IO operation functions . . . 915
54.2.4 Peripheral State and Error functions . . . 918
54.2.5 Detailed description of functions . . . 919
54.3 PKA Firmware driver defines . . . 937
54.3.1 PKA . . . 937
55 HAL PSSI Generic Driver . . . 942
55.1 PSSI Firmware driver registers structures . . . 942
55.1.1 PSSI_InitTypeDef . . . 942
55.1.2 PSSI_HandleTypeDef . . . 942
55.2 PSSI Firmware driver API description . . . 943
55.2.1 How to use this driver . . . 943
55.2.2 Initialization and de-initialization functions . . . 944
55.2.3 IO operation functions . . . 944

55.2.4 Peripheral State, Mode and Error functions . . . 945
55.2.5 Detailed description of functions . . . 945
55.3 PSSI Firmware driver defines . . . 949
55.3.1 PSSI . . . 949
56 HAL PWR Generic Driver . . . 955
56.1 PWR Firmware driver registers structures . . . 955
56.1.1 PWR_PVDTypeDef . . . 955
56.2 PWR Firmware driver API description . . . 955
56.2.1 Initialization and De-Initialization Functions . . . 955
56.2.2 Peripheral Control functions . . . 955
56.2.3 PWR Attributes Functions . . . 955
56.2.4 Detailed description of functions . . . 956
56.3 PWR Firmware driver defines . . . 962
56.3.1 PWR . . . 962
57 HAL PWR Extension Driver . . . 971
57.1 PWREx Firmware driver registers structures . . . 971
57.1.1 PWREx_AVDTypeDef . . . 971
57.1.2 PWREx_WakeupPinTypeDef . . . 971
57.2 PWREx Firmware driver API description . . . 971
57.2.1 Power supply control functions . . . 971
57.2.2 Voltage Monitoring Functions . . . 971
57.2.3 Wakeup Pins configuration Functions . . . 972
57.2.4 Memories Retention Functions . . . 972
57.2.5 IO and JTAG Retention Functions . . . 972
57.2.6 Detailed description of functions . . . 972
57.3 PWREx Firmware driver defines . . . 982
57.3.1 PWREx . . . 982
58 HAL RAMCFG Generic Driver . . . 989
58.1 RAMCFG Firmware driver registers structures . . . 989
58.1.1 __RAMCFG_HandleTypeDef . . . 989
58.2 RAMCFG Firmware driver API description . . . 989
58.2.1 RAMCFG Peripheral features . . . 989
58.2.2 How to use this driver . . . 989
58.2.3 Initialization and de-initialization Functions . . . 990
58.2.4 ECC Operations Functions . . . 991
58.2.5 Write Protection Functions . . . 991
58.2.6 Erase Operation Functions . . . 991

58.2.7 Handle Interrupt and Callbacks Functions . . . 991
58.2.8 State and Error Functions . . . 992
58.2.9 Detailed description of functions . . . 992
58.3 RAMCFG Firmware driver defines . . . 998
58.3.1 RAMCFG . . . 998
59 HAL RCC Generic Driver . . . 1001
59.1 RCC Firmware driver registers structures . . . 1001
59.1.1 RCC_PLLInitTypeDef . . . 1001
59.1.2 RCC_OscInitTypeDef . . . 1001
59.1.3 RCC_ClkInitTypeDef . . . 1002
59.2 RCC Firmware driver API description . . . 1003
59.2.1 RCC specific features . . . 1003
59.2.2 Initialization and de-initialization functions . . . 1003
59.2.3 Peripheral Control functions . . . 1006
59.2.4 RCC attributes functions . . . 1006
59.2.5 Detailed description of functions . . . 1006
59.3 RCC Firmware driver defines . . . 1012
59.3.1 RCC . . . 1012
60 HAL RCC Extension Driver . . . 1071
60.1 RCCEx Firmware driver registers structures . . . 1071
60.1.1 RCC_PLL2InitTypeDef . . . 1071
60.1.2 RCC_PLL3InitTypeDef . . . 1071
60.1.3 PLL1_ClocksTypeDef . . . 1072
60.1.4 PLL2_ClocksTypeDef . . . 1072
60.1.5 PLL3_ClocksTypeDef . . . 1073
60.1.6 RCC_PeriphCLKInitTypeDef . . . 1073
60.1.7 RCC_CRSInitTypeDef . . . 1076
60.1.8 RCC_CRSSynchroInfoTypeDef . . . 1076
60.2 RCCEx Firmware driver API description . . . 1077
60.2.1 Extended Peripheral Control functions . . . 1077
60.2.2 Extended clock management functions . . . 1077
60.2.3 Extended Clock Recovery System Control functions . . . 1077
60.2.4 Detailed description of functions . . . 1079
60.3 RCCEx Firmware driver defines . . . 1089
60.3.1 RCCEx . . . 1089
61 HAL RNG Generic Driver . . . 1139
61.1 RNG Firmware driver registers structures . . . 1139

61.1.1 RNG_InitTypeDef . . . 1139
61.1.2 __RNG_HandleTypeDef . . . 1139
61.2 RNG Firmware driver API description . . . 1139
61.2.1 How to use this driver . . . 1140
61.2.2 Callback registration . . . 1140
61.2.3 Initialization and configuration functions . . . 1140
61.2.4 Peripheral Control functions . . . 1141
61.2.5 Peripheral State functions . . . 1141
61.2.6 Detailed description of functions . . . 1141
61.3 RNG Firmware driver defines . . . 1146
61.3.1 RNG . . . 1146
62 HAL RNG Extension Driver . . . 1151
62.1 RNGEx Firmware driver registers structures . . . 1151
62.1.1 RNG_ConfigTypeDef . . . 1151
62.2 RNGEx Firmware driver defines . . . 1151
62.2.1 RNGEx . . . 1151
63 HAL RTC Generic Driver . . . 1152
63.1 RTC Firmware driver registers structures . . . 1152
63.1.1 RTC_InitTypeDef . . . 1152
63.1.2 RTC_TimeTypeDef . . . 1152
63.1.3 RTC_DateTypeDef . . . 1153
63.1.4 RTC_AlarmTypeDef . . . 1153
63.1.5 __RTC_HandleTypeDef . . . 1154
63.2 RTC Firmware driver API description . . . 1156
63.2.1 RTC Operating Condition . . . 1156
63.2.2 Backup Domain Reset . . . 1156
63.2.3 Backup Domain Access . . . 1156
63.2.4 How to use RTC Driver . . . 1157
63.2.5 RTC and low power modes . . . 1157
63.2.6 Initialization and de-initialization functions . . . 1157
63.2.7 RTC Time and Date functions . . . 1158
63.2.8 RTC Alarm functions . . . 1158
63.2.9 Peripheral Control functions . . . 1158
63.2.10 Peripheral State functions . . . 1158
63.2.11 Detailed description of functions . . . 1158
63.3 RTC Firmware driver defines . . . 1169
63.3.1 RTC . . . 1169

64 HAL RTC Extension Driver 1179
64.1 RTCEx Firmware driver registers structures 1179
64.1.1 RTC_TamperTypeDef 1179
64.1.2 RTC_ATampInputTypeDef 1179
64.1.3 RTC_ActiveTampersTypeDef 1180
64.1.4 RTC_InternalTamperTypeDef 1180
64.1.5 RTC_SecureStateTypeDef 1180
64.1.6 RTC_PrivilegeStateTypeDef 1181
64.2 RTCEx Firmware driver API description 1182
64.2.1 How to use this driver 1182
64.2.2 RTC TimeStamp and Tamper functions 1184
64.2.3 RTC Wake-up functions 1184
64.2.4 Extended Peripheral Control functions 1184
64.2.5 Extended features functions 1185
64.2.6 Tamper functions 1185
64.2.7 Extended RTC Backup register functions 1186
64.2.8 Extended RTC security functions 1186
64.2.9 Extended RTC privilege functions 1186
64.2.10 Detailed description of functions 1186
64.3 RTCEx Firmware driver defines 1209
64.3.1 RTCEx 1209
65 HAL SAI Generic Driver 1238
65.1 SAI Firmware driver registers structures 1238
65.1.1 SAI_PdmInitTypeDef 1238
65.1.2 SAI_InitTypeDef 1238
65.1.3 SAI_FrameInitTypeDef 1239
65.1.4 SAI_SlotInitTypeDef 1240
65.1.5 __SAI_HandleTypeDef 1240
65.2 SAI Firmware driver API description 1242
65.2.1 How to use this driver 1242
65.2.2 Initialization and de-initialization functions 1244
65.2.3 IO operation functions 1245
65.2.4 Peripheral State and Errors functions 1245
65.2.5 Detailed description of functions 1246
65.3 SAI Firmware driver defines 1254
65.3.1 SAI 1254
66 HAL SAI Extension Driver 1263

66.1 SAIEx Firmware driver registers structures . . . 1263
66.1.1 SAIEx_PdmMicDelayParamTypeDef . . . 1263
66.2 SAIEx Firmware driver API description . . . 1263
66.2.1 Extended features functions . . . 1263
66.2.2 Detailed description of functions . . . 1263
67 HAL SDRAM Generic Driver . . . 1264
67.1 SDRAM Firmware driver registers structures . . . 1264
67.1.1 __SDRAM_HandleTypeDef . . . 1264
67.2 SDRAM Firmware driver API description . . . 1264
67.2.1 How to use this driver . . . 1264
67.2.2 SDRAM Initialization and de_initialization functions . . . 1265
67.2.3 SDRAM Input and Output functions . . . 1266
67.2.4 SDRAM Control functions . . . 1266
67.2.5 SDRAM State functions . . . 1266
67.2.6 Detailed description of functions . . . 1266
67.3 SDRAM Firmware driver defines . . . 1274
67.3.1 SDRAM . . . 1274
68 HAL SD Generic Driver . . . 1275
68.1 SD Firmware driver registers structures . . . 1275
68.1.1 HAL_SD_CardInfoTypeDef . . . 1275
68.1.2 __SD_HandleTypeDef . . . 1275
68.1.3 HAL_SD_CardCSDTypeDef . . . 1276
68.1.4 HAL_SD_CardCIDTypeDef . . . 1279
68.1.5 HAL_SD_CardStatusTypeDef . . . 1279
68.2 SD Firmware driver API description . . . 1280
68.2.1 How to use this driver . . . 1280
68.2.2 Initialization and de-initialization functions . . . 1283
68.2.3 IO operation functions . . . 1283
68.2.4 Peripheral Control functions . . . 1283
68.2.5 Detailed description of functions . . . 1284
68.3 SD Firmware driver defines . . . 1293
68.3.1 SD . . . 1293
69 HAL SD Extension Driver . . . 1304
69.1 SDEx Firmware driver API description . . . 1304
69.1.1 How to use this driver . . . 1304
69.1.2 Linked List management functions . . . 1304
69.1.3 Detailed description of functions . . . 1304

69.2 SDEx Firmware driver defines . . . 1307
69.2.1 SDEx . . . 1307
70 HAL SMARTCARD Generic Driver . . . 1308
70.1 SMARTCARD Firmware driver registers structures . . . 1308
70.1.1 SMARTCARD_InitTypeDef . . . 1308
70.1.2 SMARTCARD_AdvFeatureInitTypeDef . . . 1309
70.1.3 __SMARTCARD_HandleTypeDef . . . 1310
70.2 SMARTCARD Firmware driver API description . . . 1312
70.2.1 How to use this driver . . . 1312
70.2.2 Callback registration . . . 1313
70.2.3 Initialization and Configuration functions . . . 1314
70.2.4 IO operation functions . . . 1315
70.2.5 Peripheral State and Errors functions . . . 1316
70.2.6 Detailed description of functions . . . 1316
70.3 SMARTCARD Firmware driver defines . . . 1326
70.3.1 SMARTCARD . . . 1326
71 HAL SMARTCARD Extension Driver . . . 1339
71.1 SMARTCARDEx Firmware driver API description . . . 1339
71.1.1 SMARTCARD peripheral extended features . . . 1339
71.1.2 Peripheral Control functions . . . 1339
71.1.3 IO operation functions . . . 1339
71.1.4 Peripheral FIFO Control functions . . . 1339
71.1.5 Detailed description of functions . . . 1340
71.2 SMARTCARDEx Firmware driver defines . . . 1343
71.2.1 SMARTCARDEx . . . 1343
72 HAL SMBUS Generic Driver . . . 1348
72.1 SMBUS Firmware driver registers structures . . . 1348
72.1.1 SMBUS_InitTypeDef . . . 1348
72.1.2 __SMBUS_HandleTypeDef . . . 1349
72.2 SMBUS Firmware driver API description . . . 1350
72.2.1 How to use this driver . . . 1350
72.2.2 Initialization and de-initialization functions . . . 1352
72.2.3 IO operation functions . . . 1353
72.2.4 Peripheral State and Errors functions . . . 1353
72.2.5 Detailed description of functions . . . 1354
72.3 SMBUS Firmware driver defines . . . 1363
72.3.1 SMBUS . . . 1363

73 HAL SMBUS Extension Driver . . . 1371
73.1 SMBUSEx Firmware driver API description . . . 1371
73.1.1 SMBUS peripheral Extended features . . . 1371
73.1.2 How to use this driver . . . 1371
73.1.3 WakeUp Mode Functions . . . 1371
73.1.4 Fast Mode Plus Functions . . . 1371
73.1.5 Detailed description of functions . . . 1371
73.2 SMBUSEx Firmware driver defines . . . 1372
73.2.1 SMBUSEx . . . 1372
74 HAL SPI Generic Driver . . . 1373
74.1 SPI Firmware driver registers structures . . . 1373
74.1.1 SPI_InitTypeDef . . . 1373
74.1.2 __SPI_HandleTypeDef . . . 1374
74.2 SPI Firmware driver API description . . . 1376
74.2.1 How to use this driver . . . 1376
74.2.2 Initialization and de-initialization functions . . . 1378
74.2.3 IO operation functions . . . 1379
74.2.4 Peripheral State and Errors functions . . . 1379
74.2.5 Detailed description of functions . . . 1380
74.3 SPI Firmware driver defines . . . 1389
74.3.1 SPI . . . 1389
75 HAL SPI Extension Driver . . . 1402
75.1 SPIEx Firmware driver API description . . . 1402
75.1.1 IO operation functions . . . 1402
75.1.2 Detailed description of functions . . . 1402
76 HAL SRAM Generic Driver . . . 1404
76.1 SRAM Firmware driver registers structures . . . 1404
76.1.1 __SRAM_HandleTypeDef . . . 1404
76.2 SRAM Firmware driver API description . . . 1404
76.2.1 How to use this driver . . . 1404
76.2.2 SRAM Initialization and de_initialization functions . . . 1405
76.2.3 SRAM Input and Output functions . . . 1406
76.2.4 SRAM Control functions . . . 1406
76.2.5 SRAM State functions . . . 1406
76.2.6 Detailed description of functions . . . 1406
76.3 SRAM Firmware driver defines . . . 1412
76.3.1 SRAM . . . 1413

77 HAL TIM Generic Driver 1414
77.1 TIM Firmware driver registers structures 1414
77.1.1 TIM_Base_InitTypeDef 1414
77.1.2 TIM_OC_InitTypeDef 1414
77.1.3 TIM_OnePulse_InitTypeDef 1415
77.1.4 TIM_IC_InitTypeDef 1416
77.1.5 TIM_Encoder_InitTypeDef 1416
77.1.6 TIM_ClockConfigTypeDef 1417
77.1.7 TIM_ClearInputConfigTypeDef 1417
77.1.8 TIM_MasterConfigTypeDef 1418
77.1.9 TIM_SlaveConfigTypeDef 1418
77.1.10 TIM_BreakDeadTimeConfigTypeDef 1418
77.1.11 __TIM_HandleTypeDef 1419
77.2 TIM Firmware driver API description 1422
77.2.1 TIMER Generic features 1422
77.2.2 How to use this driver 1422
77.2.3 Time Base functions 1424
77.2.4 TIM Output Compare functions 1424
77.2.5 TIM PWM functions 1425
77.2.6 TIM Input Capture functions 1425
77.2.7 TIM One Pulse functions 1426
77.2.8 TIM Encoder functions 1426
77.2.9 TIM Callbacks functions 1427
77.2.10 Detailed description of functions 1427
77.3 TIM Firmware driver defines 1469
77.3.1 TIM 1469
78 HAL TIM Extension Driver 1500
78.1 TIMEx Firmware driver registers structures 1500
78.1.1 TIM_HallSensor_InitTypeDef 1500
78.1.2 TIMEx_BreakInputConfigTypeDef 1500
78.1.3 TIMEx_EncoderIndexConfigTypeDef 1500
78.2 TIMEx Firmware driver API description 1501
78.2.1 TIMER Extended features 1501
78.2.2 How to use this driver 1501
78.2.3 Timer Hall Sensor functions 1502
78.2.4 Timer Complementary Output Compare functions 1502
78.2.5 Timer Complementary PWM functions 1503
78.2.6 Timer Complementary One Pulse functions 1503

78.2.7 Peripheral Control functions . . . 1503
78.2.8 Extended Callbacks functions . . . 1504
78.2.9 Extended Peripheral State functions . . . 1504
78.2.10 Detailed description of functions . . . 1505
78.3 TIMEx Firmware driver defines . . . 1531
78.3.1 TIMEx . . . 1531
79 HAL UART Generic Driver . . . 1540
79.1 UART Firmware driver registers structures . . . 1540
79.1.1 UART_InitTypeDef . . . 1540
79.1.2 UART_AdvFeatureInitTypeDef . . . 1541
79.1.3 __UART_HandleTypeDef . . . 1541
79.2 UART Firmware driver API description . . . 1544
79.2.1 How to use this driver . . . 1544
79.2.2 Callback registration . . . 1544
79.2.3 Initialization and Configuration functions . . . 1545
79.2.4 IO operation functions . . . 1546
79.2.5 Peripheral Control functions . . . 1547
79.2.6 Peripheral State and Error functions . . . 1547
79.2.7 Detailed description of functions . . . 1548
79.3 UART Firmware driver defines . . . 1564
79.3.1 UART . . . 1565
80 HAL UART Extension Driver . . . 1585
80.1 UARTEx Firmware driver registers structures . . . 1585
80.1.1 UART_WakeUpTypeDef . . . 1585
80.2 UARTEx Firmware driver API description . . . 1585
80.2.1 UART peripheral extended features . . . 1585
80.2.2 Initialization and Configuration functions . . . 1585
80.2.3 IO operation functions . . . 1586
80.2.4 Peripheral Control functions . . . 1586
80.2.5 Detailed description of functions . . . 1587
80.3 UARTEx Firmware driver defines . . . 1593
80.3.1 UARTEx . . . 1593
81 HAL USART Generic Driver . . . 1595
81.1 USART Firmware driver registers structures . . . 1595
81.1.1 USART_InitTypeDef . . . 1595
81.1.2 __USART_HandleTypeDef . . . 1595
81.2 USART Firmware driver API description . . . 1597

81.2.1 How to use this driver . . . 1597
81.2.2 Callback registration . . . 1598
81.2.3 Initialization and Configuration functions . . . 1599
81.2.4 IO operation functions . . . 1599
81.2.5 Peripheral State and Error functions . . . 1601
81.2.6 Detailed description of functions . . . 1601
81.3 USART Firmware driver defines . . . 1611
81.3.1 USART . . . 1611
82 HAL USART Extension Driver . . . 1623
82.1 USARTEx Firmware driver API description . . . 1623
82.1.1 USART peripheral extended features . . . 1623
82.1.2 IO operation functions . . . 1623
82.1.3 Peripheral Control functions . . . 1623
82.1.4 Detailed description of functions . . . 1623
82.2 USARTEx Firmware driver defines . . . 1626
82.2.1 USARTEx . . . 1626
83 HAL WWDG Generic Driver . . . 1629
83.1 WWDG Firmware driver registers structures . . . 1629
83.1.1 WWDG_InitTypeDef . . . 1629
83.1.2 __WWDG_HandleTypeDef . . . 1629
83.2 WWDG Firmware driver API description . . . 1629
83.2.1 WWDG Specific features . . . 1629
83.2.2 How to use this driver . . . 1630
83.2.3 Initialization and Configuration functions . . . 1631
83.2.4 IO operation functions . . . 1631
83.2.5 Detailed description of functions . . . 1631
83.3 WWDG Firmware driver defines . . . 1633
83.3.1 WWDG . . . 1633
84 HAL XSPI Generic Driver . . . 1637
84.1 XSPI Firmware driver registers structures . . . 1637
84.1.1 XSPI_InitTypeDef . . . 1637
84.1.2 __XSPI_HandleTypeDef . . . 1638
84.1.3 XSPI_RegularCmdTypeDef . . . 1639
84.1.4 XSPI_HyperbusCfgTypeDef . . . 1640
84.1.5 XSPI_HyperbusCmdTypeDef . . . 1641
84.1.6 XSPI_AutoPollingTypeDef . . . 1641
84.1.7 XSPI_MemoryMappedTypeDef . . . 1642

84.2 XSPI Firmware driver API description . . . 1642
84.2.1 How to use this driver . . . 1642
84.2.2 Initialization and Configuration functions . . . 1645
84.2.3 IO operation functions . . . 1645
84.2.4 Peripheral Control and State functions . . . 1646
84.2.5 Delay Block function . . . 1646
84.2.6 Detailed description of functions . . . 1646
84.3 XSPI Firmware driver defines . . . 1659
84.3.1 XSPI . . . 1659
85 LL ADC Generic Driver . . . 1673
85.1 ADC Firmware driver registers structures . . . 1673
85.1.1 LL_ADC_CommonInitTypeDef . . . 1673
85.1.2 LL_ADC_InitTypeDef . . . 1673
85.1.3 LL_ADC_REG_InitTypeDef . . . 1673
85.1.4 LL_ADC_INJ_InitTypeDef . . . 1674
85.2 ADC Firmware driver API description . . . 1675
85.2.1 Detailed description of functions . . . 1675
85.3 ADC Firmware driver defines . . . 1792
85.3.1 ADC . . . 1792
86 LL BUS Generic Driver . . . 1832
86.1 BUS Firmware driver API description . . . 1832
86.1.1 Detailed description of functions . . . 1832
86.2 BUS Firmware driver defines . . . 1889
86.2.1 BUS . . . 1889
87 LL COMP Generic Driver . . . 1894
87.1 COMP Firmware driver registers structures . . . 1894
87.1.1 LL_COMP_InitTypeDef . . . 1894
87.2 COMP Firmware driver API description . . . 1894
87.2.1 Detailed description of functions . . . 1894
87.3 COMP Firmware driver defines . . . 1906
87.3.1 COMP . . . 1906
88 LL CORDIC Generic Driver . . . 1910
88.1 CORDIC Firmware driver API description . . . 1910
88.1.1 Detailed description of functions . . . 1910
88.2 CORDIC Firmware driver defines . . . 1922
88.2.1 CORDIC . . . 1922
89 LL CORTEX Generic Driver . . . 1926

89.1 CORTEX Firmware driver API description . . . 1926
89.1.1 Detailed description of functions . . . 1926
89.2 CORTEX Firmware driver defines . . . 1949
89.2.1 CORTEX . . . 1949
90 LL CRC Generic Driver . . . 1953
90.1 CRC Firmware driver API description . . . 1953
90.1.1 Detailed description of functions . . . 1953
90.2 CRC Firmware driver defines . . . 1960
90.2.1 CRC . . . 1960
91 LL CRS Generic Driver . . . 1962
91.1 CRS Firmware driver API description . . . 1962
91.1.1 Detailed description of functions . . . 1962
91.2 CRS Firmware driver defines . . . 1975
91.2.1 CRS . . . 1975
92 LL DAC Generic Driver . . . 1978
92.1 DAC Firmware driver registers structures . . . 1978
92.1.1 LL_DAC_InitTypeDef . . . 1978
92.2 DAC Firmware driver API description . . . 1978
92.2.1 Detailed description of functions . . . 1979
92.3 DAC Firmware driver defines . . . 2011
92.3.1 DAC . . . 2011
93 LL DCACHE Generic Driver . . . 2018
93.1 DCACHE Firmware driver API description . . . 2018
93.1.1 Detailed description of functions . . . 2018
93.2 DCACHE Firmware driver defines . . . 2030
93.2.1 DCACHE . . . 2030
94 LL DLYB Generic Driver . . . 2032
94.1 DLYB Firmware driver registers structures . . . 2032
94.1.1 LL_DLYB_CfgTypeDef . . . 2032
94.2 DLYB Firmware driver API description . . . 2032
94.2.1 DelayBlock peripheral features . . . 2032
94.2.2 How to use this driver . . . 2032
94.2.3 Detailed description of functions . . . 2032
94.3 DLYB Firmware driver defines . . . 2034
94.3.1 DLYB . . . 2034
95 LL DMA Generic Driver . . . 2035

95.1 DMA Firmware driver registers structures . . . 2035
95.1.1 LL_DMA_InitTypeDef . . . 2035
95.1.2 LL_DMA_InitLinkedListTypeDef . . . 2038
95.1.3 LL_DMA_InitNodeTypeDef . . . 2039
95.1.4 LL_DMA_LinkNodeTypeDef . . . 2041
95.2 DMA Firmware driver API description . . . 2041
95.2.1 LL DMA driver acronyms . . . 2041
95.2.2 Detailed description of functions . . . 2042
95.3 DMA Firmware driver defines . . . 2159
95.3.1 DMA . . . 2159
96 LL EXTI Generic Driver . . . 2185
96.1 EXTI Firmware driver registers structures . . . 2185
96.1.1 LL_EXTI_InitTypeDef . . . 2185
96.2 EXTI Firmware driver API description . . . 2185
96.2.1 Detailed description of functions . . . 2185
96.3 EXTI Firmware driver defines . . . 2231
96.3.1 EXTI . . . 2231
97 LL FMAC Generic Driver . . . 2236
97.1 FMAC Firmware driver API description . . . 2236
97.1.1 Detailed description of functions . . . 2236
97.2 FMAC Firmware driver defines . . . 2258
97.2.1 FMAC . . . 2258
98 LL GPIO Generic Driver . . . 2260
98.1 GPIO Firmware driver registers structures . . . 2260
98.1.1 LL_GPIO_InitTypeDef . . . 2260
98.2 GPIO Firmware driver API description . . . 2260
98.2.1 Detailed description of functions . . . 2260
98.3 GPIO Firmware driver defines . . . 2286
98.3.1 GPIO . . . 2286
99 LL I2C Generic Driver . . . 2290
99.1 I2C Firmware driver registers structures . . . 2290
99.1.1 LL_I2C_InitTypeDef . . . 2290
99.2 I2C Firmware driver API description . . . 2290
99.2.1 Detailed description of functions . . . 2290
99.3 I2C Firmware driver defines . . . 2342
99.3.1 I2C . . . 2342
100 LL I2S Generic Driver . . . 2348

100.1 I2S Firmware driver registers structures . . . 2348
100.1.1 LL_I2S_InitTypeDef . . . 2348
100.2 I2S Firmware driver API description . . . 2348
100.2.1 Detailed description of functions . . . 2348
100.3 I2S Firmware driver defines . . . 2373
100.3.1 I2S . . . 2373
101 LL I3C Generic Driver . . . 2376
101.1 I3C Firmware driver registers structures . . . 2376
101.1.1 LL_I3C_CtrlBusConfTypeDef . . . 2376
101.1.2 LL_I3C_TgtBusConfTypeDef . . . 2376
101.1.3 LL_I3C_InitTypeDef . . . 2377
101.2 I3C Firmware driver API description . . . 2377
101.2.1 Detailed description of functions . . . 2377
101.3 I3C Firmware driver defines . . . 2474
101.3.1 I3C . . . 2474
102 LL ICACHE Generic Driver . . . 2484
102.1 ICACHE Firmware driver registers structures . . . 2484
102.1.1 LL_ICACHE_RegionTypeDef . . . 2484
102.2 ICACHE Firmware driver API description . . . 2484
102.2.1 Detailed description of functions . . . 2484
102.3 ICACHE Firmware driver defines . . . 2497
102.3.1 ICACHE . . . 2497
103 LL IWDG Generic Driver . . . 2500
103.1 IWDG Firmware driver API description . . . 2500
103.1.1 Detailed description of functions . . . 2500
103.2 IWDG Firmware driver defines . . . 2507
103.2.1 IWDG . . . 2507
104 LL LPTIM Generic Driver . . . 2509
104.1 LPTIM Firmware driver registers structures . . . 2509
104.1.1 LL_LPTIM_InitTypeDef . . . 2509
104.2 LPTIM Firmware driver API description . . . 2509
104.2.1 Detailed description of functions . . . 2509
104.3 LPTIM Firmware driver defines . . . 2558
104.3.1 LPTIM . . . 2558
105 LL LPUART Generic Driver . . . 2567
105.1 LPUART Firmware driver registers structures . . . 2567

105.1.1 LL_LPUART_InitTypeDef . . . 2567
105.2 LPUART Firmware driver API description . . . 2567
105.2.1 Detailed description of functions . . . 2567
105.3 LPUART Firmware driver defines . . . 2622
105.3.1 LPUART . . . 2622
106 LL OPAMP Generic Driver . . . 2630
106.1 OPAMP Firmware driver registers structures . . . 2630
106.1.1 LL_OPAMP_InitTypeDef . . . 2630
106.2 OPAMP Firmware driver API description . . . 2630
106.2.1 Detailed description of functions . . . 2630
106.3 OPAMP Firmware driver defines . . . 2640
106.3.1 OPAMP . . . 2640
107 LL PKA Generic Driver . . . 2644
107.1 PKA Firmware driver registers structures . . . 2644
107.1.1 LL_PKA_InitTypeDef . . . 2644
107.2 PKA Firmware driver API description . . . 2644
107.2.1 Detailed description of functions . . . 2644
107.3 PKA Firmware driver defines . . . 2654
107.3.1 PKA . . . 2655
108 LL PWR Generic Driver . . . 2658
108.1 PWR Firmware driver API description . . . 2658
108.1.1 Detailed description of functions . . . 2658
108.2 PWR Firmware driver defines . . . 2696
108.2.1 PWR . . . 2696
109 LL RCC Generic Driver . . . 2702
109.1 RCC Firmware driver registers structures . . . 2702
109.1.1 LL_RCC_ClocksTypeDef . . . 2702
109.1.2 LL_PLL_ClocksTypeDef . . . 2702
109.2 RCC Firmware driver API description . . . 2702
109.2.1 Detailed description of functions . . . 2702
109.3 RCC Firmware driver defines . . . 2816
109.3.1 RCC . . . 2816
110 LL RNG Generic Driver . . . 2847
110.1 RNG Firmware driver registers structures . . . 2847
110.1.1 LL_RNG_InitTypeDef . . . 2847
110.2 RNG Firmware driver API description . . . 2847

110.2.1 Detailed description of functions . . . 2847
110.3 RNG Firmware driver defines . . . 2860
110.3.1 RNG . . . 2860
111 LL RTC Generic Driver . . . 2863
111.1 RTC Firmware driver registers structures . . . 2863
111.1.1 LL_RTC_InitTypeDef . . . 2863
111.1.2 LL_RTC_TimeTypeDef . . . 2863
111.1.3 LL_RTC_DateTypeDef . . . 2863
111.1.4 LL_RTC_AlarmTypeDef . . . 2864
111.2 RTC Firmware driver API description . . . 2864
111.2.1 Detailed description of functions . . . 2864
111.3 RTC Firmware driver defines . . . 3000
111.3.1 RTC . . . 3000
112 LL SPI Generic Driver . . . 3020
112.1 SPI Firmware driver registers structures . . . 3020
112.1.1 LL_SPI_InitTypeDef . . . 3020
112.2 SPI Firmware driver API description . . . 3021
112.2.1 Detailed description of functions . . . 3021
112.3 SPI Firmware driver defines . . . 3073
112.3.1 SPI . . . 3073
113 LL SYSTEM Generic Driver . . . 3081
113.1 SYSTEM Firmware driver API description . . . 3081
113.1.1 Detailed description of functions . . . 3081
113.2 SYSTEM Firmware driver defines . . . 3114
113.2.1 SYSTEM . . . 3114
114 LL TIM Generic Driver . . . 3122
114.1 TIM Firmware driver registers structures . . . 3122
114.1.1 LL_TIM_InitTypeDef . . . 3122
114.1.2 LL_TIM_OC_InitTypeDef . . . 3122
114.1.3 LL_TIM_IC_InitTypeDef . . . 3123
114.1.4 LL_TIM_ENCODER_InitTypeDef . . . 3123
114.1.5 LL_TIM_HALLSENSOR_InitTypeDef . . . 3124
114.1.6 LL_TIM_BDTR_InitTypeDef . . . 3125
114.2 TIM Firmware driver API description . . . 3126
114.2.1 Detailed description of functions . . . 3127
114.3 TIM Firmware driver defines . . . 3243
114.3.1 TIM . . . 3243

115 LL UCPD Generic Driver 3271
115.1 UCPD Firmware driver registers structures 3271
115.1.1 LL_UCPD_InitTypeDef 3271
115.2 UCPD Firmware driver API description 3271
115.2.1 Detailed description of functions 3271
115.3 UCPD Firmware driver defines 3311
115.3.1 UCPD 3312
116 LL USART Generic Driver 3318
116.1 USART Firmware driver registers structures 3318
116.1.1 LL_USART_InitTypeDef 3318
116.1.2 LL_USART_ClockInitTypeDef 3318
116.2 USART Firmware driver API description 3319
116.2.1 Detailed description of functions 3319
116.3 USART Firmware driver defines 3415
116.3.1 USART 3415
117 LL UTILS Generic Driver 3425
117.1 UTILS Firmware driver registers structures 3425
117.1.1 LL_UTILS_PLLInitTypeDef 3425
117.1.2 LL_UTILS_ClkInitTypeDef 3425
117.2 UTILS Firmware driver API description 3426
117.2.1 System Configuration functions 3426
117.2.2 Detailed description of functions 3426
117.3 UTILS Firmware driver defines 3431
117.3.1 UTILS 3431
118 LL WWDG Generic Driver 3433
118.1 WWDG Firmware driver API description 3433
118.1.1 Detailed description of functions 3433
118.2 WWDG Firmware driver defines 3437
118.2.1 WWDG 3437
119 Correspondence between API registers and API low-layer driver functions 3439
119.1 ADC 3439
119.2 BUS 3448
119.3 COMP 3464
119.4 CORDIC 3465
119.5 CORTEX 3466
119.6 CRC 3468

119.7 CRS . . . 3469
119.8 DAC . . . 3470
119.9 DCACHE . . . 3473
119.10 DMA . . . 3474
119.11 EXTI . . . 3479
119.12 FMAC . . . 3481
119.13 GPIO . . . 3483
119.14 I2C . . . 3484
119.15 I2S . . . 3488
119.16 I3C . . . 3490
119.17 ICACHE . . . 3498
119.18 IWDG . . . 3499
119.19 LPTIM . . . 3500
119.20 LPUART . . . 3503
119.21 PKA . . . 3508
119.22 PWR . . . 3508
119.23 RCC . . . 3513
119.24 RNG . . . 3522
119.25 RTC . . . 3523
119.26 OPAMP . . . 3536
119.27 SPI . . . 3536
119.28 SYSTEM . . . 3540
119.29 TIM . . . 3542
119.30 UCPD . . . 3554
119.31 USART . . . 3558
119.32 WWDG . . . 3565
120 FAQs . . . 3566
Revision history . . . 3569
List of tables . . . 3607
List of figures . . . 3609

List of tables

Table 1. Acronyms and definitions . . . 4
Table 2. HAL driver files . . . 7
Table 3. User-application files . . . 7
Table 4. API classification . . . 12
Table 5. List of devices supported by HAL/LL drivers . . . 13
Table 6. HAL API naming rules . . . 16
Table 7. Macros handling interrupts and specific clock configurations . . . 17
Table 8. Callback functions . . . 18
Table 9. HAL generic APIs . . . 19
Table 10. HAL extension APIs . . . 19
Table 11. Define statements used for HAL configuration . . . 23
Table 12. Description of GPIO_InitTypeDef structure . . . 25
Table 13. Description of EXTI configuration macros . . . 27
Table 14. MSP functions . . . 33
Table 15. Timeout values . . . 36
Table 16. LL driver files . . . 40
Table 17. Common peripheral initialization functions . . . 43
Table 18. Optional peripheral initialization functions . . . 43
Table 19. Specific Interrupt, DMA request and status flags management . . . 44
Table 20. Available function formats . . . 44
Table 21. Peripheral clock activation/deactivation management . . . 45
Table 22. Peripheral activation/deactivation management . . . 45
Table 23. Peripheral configuration management . . . 45
Table 24. Peripheral register management . . . 45
Table 25. Correspondence between ADC registers and ADC low-layer driver functions . . . 3439
Table 26. Correspondence between BUS registers and BUS low-layer driver functions . . . 3448
Table 27. Correspondence between COMP registers and COMP low-layer driver functions . . . 3464
Table 28. Correspondence between CORDIC registers and CORDIC low-layer driver functions . . . 3465
Table 29. Correspondence between CORTEX registers and CORTEX low-layer driver functions . . . 3466
Table 30. Correspondence between CRC registers and CRC low-layer driver functions . . . 3468
Table 31. Correspondence between CRS registers and CRS low-layer driver functions . . . 3469
Table 32. Correspondence between DAC registers and DAC low-layer driver functions . . . 3470
Table 33. Correspondence between DCACHE registers and DCACHE low-layer driver functions . . . 3473
Table 34. Correspondence between DMA registers and DMA low-layer driver functions . . . 3474
Table 35. Correspondence between EXTI registers and EXTI low-layer driver functions . . . 3479
Table 36. Correspondence between FMAC registers and FMAC low-layer driver functions . . . 3481
Table 37. Correspondence between GPIO registers and GPIO low-layer driver functions . . . 3483
Table 38. Correspondence between I2C registers and I2C low-layer driver functions . . . 3484
Table 39. Correspondence between I2S registers and I2S low-layer driver functions . . . 3488
Table 40. Correspondence between I3C registers and I3C low-layer driver functions . . . 3490
Table 41. Correspondence between ICACHE registers and ICACHE low-layer driver functions . . . 3498
Table 42. Correspondence between IWDG registers and IWDG low-layer driver functions . . . 3499
Table 43. Correspondence between LPTIM registers and LPTIM low-layer driver functions . . . 3500
Table 44. Correspondence between LPUART registers and LPUART low-layer driver functions . . . 3503
Table 45. Correspondence between PKA registers and PKA low-layer driver functions . . . 3508
Table 46. Correspondence between PWR registers and PWR low-layer driver functions . . . 3508
Table 47. Correspondence between RCC registers and RCC low-layer driver functions . . . 3513
Table 48. Correspondence between RNG registers and RNG low-layer driver functions . . . 3522
Table 49. Correspondence between RTC registers and RTC low-layer driver functions . . . 3523
Table 50. Correspondence between OPAMP registers and OPAMP low-layer driver functions . . . 3536
Table 51. Correspondence between SPI registers and SPI low-layer driver functions . . . 3536
Table 52. Correspondence between SYSTEM registers and SYSTEM low-layer driver functions . . . 3540
Table 53. Correspondence between TIM registers and TIM low-layer driver functions . . . 3542

Table 54. Correspondence between UCPD registers and UCPD low-layer driver functions 3554
Table 55. Correspondence between USART registers and USART low-layer driver functions 3558
Table 56. Correspondence between WWDG registers and WWDG low-layer driver functions 3565
Table 57. Document revision history 3569

List of figures

Figure 1. Example of project template . . . 9
Figure 2. Adding device-specific functions . . . 20
Figure 3. Adding family-specific functions . . . 20
Figure 4. Adding new peripherals . . . 21
Figure 5. Updating existing APIs . . . 21
Figure 6. File inclusion model . . . 22
Figure 7. HAL driver model . . . 31
Figure 8. Low-layer driver folders . . . 41
Figure 9. Low-layer driver CMSIS files . . . 42

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2023 STMicroelectronics – All rights reserved