STM32L011x3/4 STM32L021x3/4

Errata sheet

STM32L011xx/L021xx device errata

Applicability

This document applies to the part numbers of STM32L011xx/L021xx devices and the device variants as stated in this page.

It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0377.

Deviation of the real device behavior from the intended device behavior is considered to be a device limitation. Deviation of the description in the reference manual or the datasheet from the intended device behavior is considered to be a documentation erratum. The term *"errata"* applies both to limitations and documentation errata.

Table 1. Device summary

Reference	Part numbers
STM32L011x3/4	STM32L011K4, STM32L011K3, STM32L011G4, STM32L011G3, STM32L011F4, STM32L011F3, STM32L011E4, STM32L011E3, STM32L011D4, STM32L011D3
STM32L021x4	STM32L021K4, STM32L021G4, STM32L021F4, STM32L021D4

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32L011x3/4	A	0x1000
STM32L021x4	1, Z	0x1008

1. *Refer to the device datasheet for how to identify this code on different types of package.*
2. *REV_ID[15:0] bitfield of DBG_IDCODE register.*

1 Summary of device errata

The following table gives a quick reference to the STM32L011xx/L021xx device limitations and their status:

A = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

“-” = limitation absent

Applicability of a workaround may depend on specific conditions of target application. Adoption of a workaround may cause restrictions to target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes, of the function concerned.

Table 3. Summary of device limitations

Function	Section	Limitation	Status	
			Rev. A	Rev. 1,Z
System	2.1.1	Delay after an RCC peripheral clock enabling	A	A
	2.1.2	I2C and USART cannot wake up the device from Stop mode	N	-
	2.1.3	LDM, STM, PUSH and POP not allowed in IOPORT bus	N	-
	2.1.4	BOOT_MODE bits do not reflect the selected boot mode	N	N
	2.1.5	NSS pin synchronization required when using bootloader with SPI1 interface on TSSOP14 package	A	A
	2.1.6	Wake-up sequence from Standby mode fails when using more than one wake-up source	A	A
	2.1.7	LSE crystal oscillator may be disturbed by transitions on PC13	N	N
GPIO	2.2.1	GPIO locking mechanism failed if critical address is read	A	A
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
ADC	2.4.1	Overrun flag is not set if EOC reset coincides with new conversion end	P	P
	2.4.2	Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield	A	A
	2.4.3	Out-of-threshold value is not detected in AWD1 Single mode	A	A
	2.4.4	Writing ADC_CFGR2 register while ADEN bit is set resets CKMODE[1:0] bitfield	A	A
COMP	2.5.1	COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register	N	N
TIM	2.6.1	PWM re-enabled in automatic output enable mode despite of system break	P	P
	2.6.3	Consecutive compare event missed in specific conditions	N	N
	2.6.4	Output compare clear not working with external counter reset	P	P
LPTIM	2.7.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A
	2.7.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P
	2.7.3	LPTIM events and PWM output are delayed by one kernel clock cycle	P	P
IWDG	2.8.1	IWDG does not always reset the device	N	-
RTC and TAMP	2.9.1	RTC calendar registers are not locked properly	A	A
	2.9.2	RTC interrupt can be masked by another RTC interrupt	A	A
	2.9.3	Calendar initialization may fail in case of consecutive INIT mode entry	A	A

Function	Section	Limitation	Status	
			Rev. A	Rev. 1,Z
RTC and TAMP	2.9.4	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.9.5	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N	N
	2.11.6	DMA channel 3 (CH3) not functional when USART2_RX used for data reception	A	A
I2C	2.10.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A	A
	2.10.3	Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period	P	P
	2.10.4	Spurious bus error detection in master mode	A	A
	2.10.5	Last-received byte loss in reload mode	P	P
	2.10.6	Spurious master transfer upon own slave address match	P	P
	2.10.8	OVR flag not set in underrun condition	N	N
	2.10.9	Transmission stalled after first byte transfer	A	A
	2.10.10	SDA held low upon SMBus timeout expiry in slave mode	A	A
USART	2.11.1	RTS is active while RE = 0 or UE = 0	A	A
	2.11.2	Receiver timeout counter wrong start in two-stop-bit configuration	A	A
	2.11.3	Data corruption due to noisy receive line	A	A
	2.11.4	Received data may be corrupted upon clearing the ABREN bit	A	A
	2.11.5	Noise error flag set while ONEBIT is set	N	N
LPUART	2.12.1	DMA channel 5 (CH5) not functional when LPUART1_RX used for data reception	A	A
SPI	2.13.1	BSY bit may stay high when SPI is disabled	A	A
	2.13.2	BSY bit may stay high at the end of data transfer in slave mode	A	A
	2.13.3	Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	A	A
	2.13.4	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A	A
	2.13.6	Anticipated communication upon SPI transit from slave receiver to master	A	A

The following table gives a quick reference to the documentation errata.

Table 4. Summary of device documentation errata

Function	Section	Documentation erratum
DMA	2.3.2	Byte and half-word accesses not supported
TIM	2.6.2	TRGO and TRGO2 trigger output failure
I2C	2.10.2	Wrong behavior in Stop mode
	2.10.7	START bit is cleared upon setting ADDRCF, not upon address match
SPI	2.13.5	CRC error in SPI slave mode if internal NSS changes before CRC transfer

2 Description of device errata

The following sections describe the errata of the applicable devices with Arm® core and provide workarounds if available. They are grouped by device functions.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2.1 System

2.1.1 Delay after an RCC peripheral clock enabling

Description

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

This delay depends on the peripheral mapping:

- If the peripheral is mapped on AHB: the delay should be equal to 1 AHB clock cycle after the clock enable bit is set in the hardware register.
  For I/O peripheral, the delay should be equal to 1 AHB clock cycle after the clock enable bit is set in the hardware register (only applicable to write accesses).
- If the peripheral is mapped on APB: No delay is necessary (no limitation).

Workaround

- Enable the peripheral clock some time before the peripheral read/write register is required.
- For AHB peripheral (including I/O), insert a dummy read operation to the corresponding register.

2.1.2 I2C and USART cannot wake up the device from Stop mode

Description

When the microcontroller is in Stop mode with the regulator in low-power mode, an unexpected system reset may occur if the I2C or the USART attempts to wake up the device.

This limitation also impacts LPUART when the HSI16 is used as clock source instead of LSE.

This reset is internal only and does not affect the NRST pin state and the flags in the Control/status register (RCC_CSR).

The lower the V_{DD} value, the more often this unpredictable behavior may occur.

Workaround

None.

It is recommended to avoid using the USART and I2C wakeup from Stop mode features. To disable them, keep WUPEN bit in I2C_CR1 and UESM bit in USARTx_CR1 at 0.

Two solutions are then possible to perform I2C or USART communications:

- Put the microcontroller in a mode different from Stop (or Standby mode) before initiating communications.
- Replace Stop mode with Stop mode plus regulator in main mode by keeping LPSDSR bit of PWR_CR set to 0.

2.1.3 LDM, STM, PUSH and POP not allowed in IOPORT bus

Description

The instructions Load Multiple (LM), Store Multiple (STM), PUSH and POP fail when the address points to the IOPORT bus memory area (address range = 0x5XXX XXXX).

Workaround

None.

2.1.4 BOOT_MODE bits do not reflect the selected boot mode

Description

The BOOT_MODE[1:0] bits of the SYSCFG_CFGR1 register remain set to ‘0’ while they should reflect the boot mode selected by the boot pins.

Workaround

None.

2.1.5 NSS pin synchronization required when using bootloader with SPI1 interface on TSSOP14 package

Description

When using the embedded bootloader with SPI1 interface on devices in TSSOP14 package, if the NSS pin is grounded (default status after device reset), SPI communications are not synchronized and the bootloader does not work.
To properly synchronize the SPI interface, an NSS pin falling edge is required before initiating communications.

Workaround

On devices in TSSOP14, toggle with NSS pin (PA14) after device reset.

2.1.6 Wake-up sequence from Standby mode fails when using more than one wake-up source

Description

The various wake-up sources are logically OR-ed in front of the rising-edge detector which generates the wake-up flag (WUF). The WUF flag needs to be cleared prior to the Standby mode entry, otherwise the microcontroller wakes up immediately.
If one of the configured wake-up sources is kept high during the WUF flag clearing (by setting the CWUF bit), it may mask further wake-up events on the input of the edge detector. As a consequence, the microcontroller cannot wake up from Standby mode.

Workaround

Apply the following sequence before entering Standby mode:
1. Disable all used wake-up sources.
2. Clear all related wake-up flags.
3. Re-enable all used wake-up sources.
4. Enter Standby mode.

Note: When applying this measure, if one of the wake-up sources is still kept high, be aware that the microcontroller enters Standby mode but then wakes up immediately, generating the power reset.

2.1.7 LSE crystal oscillator may be disturbed by transitions on PC13

Description

On LQFP and UFQFPN packages, the LSE crystal oscillator clock frequency can be incorrect when PC13 is toggling in input or output (for example when used for RTC_OUT1). The external clock input (LSE bypass) is not impacted by this limitation. The WLCSP and UFBGA packages are not impacted by this limitation.
Avoid toggling PC13 when LSE is used on LQFP and UFQFPN packages.

Workaround

None.

2.2 GPIO

2.2.1 GPIO locking mechanism failed if critical address is read

Description

The GPIO locking sequence requires predefined steps of writing and reading operations over the GPIOx_LCKR register. Reading from an address (SRAM/flash/peripherals) where the 10 least significant bits corresponds to the GPIOx_LCKR address (0x5000 XX1C) is interpreted by GPIO locking state machine as a dummy read from the GPIOx_LCKR register. The GPIO locking sequence fails when an additional dummy read from the GPIOx_LCKR register is inserted, resetting the locking sequence. The GPIO locking state machine is only sensitive to reading done by CPU and not by DMA.

Workaround

The GPIO locking sequence code must not use variables stored at SRAM addresses which 10 least significants bits correspond to 0x01C (each 1kB is a critical address). Avoid using variables on the stack because critical addresses can be present on the stack area in the locking sequence code. Disable interrupts during the GPIO locking sequence because interrupt service routine may perform reading from critical address. The locking sequence routine must be placed in a memory section outside of critical addresses.

Find recommended code for GPIO locking sequence below. It only uses register access with disabled interrupts in critical code part. The correct code placement (outside critical addresses) is solved here by repeating the locking sequence in case of error, so that at least one locking code is at correct placement (linker independent solution).

```
/* (1) Save interrupts enable state */
/* (2) Disable interrupts */
/* (3) Write LCKK bit to 1 and set the pin bits to lock */
/* (4) Write LCKK bit to 0 and set the pin bits to lock */
/* (5) Write LCKK bit to 1 and set the pin bits to lock */
/* (6) Restore interrupts enable state */
/* (7) Read the Lock register */
/* (8) Check the Lock register for lock (in case of error perform lock second time) */


register uint32_t tmp1; /* place critical variable into register to not use SRAM */
register uint32_t tmp2; /* place critical variable into register to not use SRAM */
uint32_t irqenabled;

tmp1 = GPIO_Pin;
tmp2 = GPIO_LCKR_LCKK | GPIO_Pin;

...
irqenabled = __get_PRIMASK();    /* (1) */ /* remember interrupts enable state */
__disable_irq();                 /* (2) */
GPIOA->LCKR = tmp2;              /* (3) */
GPIOA->LCKR = tmp1;              /* (4) */
GPIOA->LCKR = tmp2;              /* (5) */
if (!irqenabled) __enable_irq(); /* (6) */
GPIOA->LCKR;                     /* (7) */
if ((GPIOA->LCKR & GPIO_LCKR_LCKK) == 0) /* (8) */ /* check if locking failed */
{
  __disable_irq();                 /* (2) */
  GPIOA->LCKR = tmp2;              /* (3) */
  GPIOA->LCKR = tmp1;              /* (4) */
  GPIOA->LCKR = tmp2;              /* (5) */
  if (!irqenabled) __enable_irq(); /* (6) */
  GPIOA->LCKR;                     /* (7) */
  if ((GPIOA->LCKR & GPIO_LCKR_LCKK) == 0) /* (8) */
  {
   /* Manage error - optional */
```

```
    }
}
```

2.3 DMA

2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

Upon a data transfer error in a DMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the DMA_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag when the channel is active.

Workaround

Do not clear GIFx flags when the channel is active. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.3.2 Byte and half-word accesses not supported

Description

Some reference manual revisions may wrongly state that the DMA registers are byte- and half-word-accessible. Instead, the DMA registers must always be accessed through aligned 32-bit words. Byte or half-word write accesses cause an erroneous behavior.

ST's low-level driver and HAL software only use aligned 32-bit accesses to the DMA registers.

This is a description inaccuracy issue rather than a product limitation.

Workaround

No application workaround is required.

2.4 ADC

2.4.1 Overrun flag is not set if EOC reset coincides with new conversion end

Description

If the EOC flag is cleared by an ADC_DR register read operation or by software during the same APB cycle in which the data from a new conversion are written in the ADC_DR register, the overrun event duly occurs (which results in the loss of either current or new data) but the overrun flag (OVR) may stay low.

Workaround

Clear the EOC flag, by performing an ADC_DR read operation or by software within less than one ADC conversion cycle period from the last conversion cycle end, in order to avoid the coincidence with the end of the new conversion cycle.

2.4.2 Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield

Description

Modifying the ADC_CFGR1 register while ADC is enabled (ADEN set in ADC_CR) resets RES[1:0] to 00 whatever the bitfield previous value.

Workaround

Apply the following sequence:
1. Set ADDIS to disable the ADC, and wait until ADEN is cleared.
2. Program the ADC_CFGR1 register according to the application requirements.
3. Set ADEN bit.

2.4.3 Out-of-threshold value is not detected in AWD1 Single mode

Description

AWD1 analog watchdog does not detect that the result of a converted channel has reached the programmed threshold when the ADC operates in Single mode, performs a sequence of conversions, and one of the converted channels other than the first one is monitored by the AWD1 analog watchdog.

Workaround

Apply one of the following measures:
- Use a conversion sequence of one single channel.
- Configure the monitored channel as the first one of the sequence.

2.4.4 Writing ADC_CFGR2 register while ADEN bit is set resets CKMODE[1:0] bitfield

Description

Modifying the ADC_CFGR2 register while ADC is enabled (ADEN set in ADC_CR) resets CKMODE[1:0] to 00 whatever the bitfield previous value.

Workaround

Apply the following sequence:
1. Set ADDIS to disable the ADC, and wait until ADEN is cleared.
2. Program the ADC_CFGR2 register according to the application requirements.
3. Set ADEN bit.

2.5 COMP

2.5.1 COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register

Description

When the SYSCFGRST bit of RCC_APB2RSTR register is set, the COMP1_CSR and COMP2_CSR register contents are reset even if COMP1LOCK and COMP2LOCK bits are set in COMP1_CSR and the COMP2_CSR register, respectively.

Workaround

None.
For security reasons, it is recommended to avoid using SYSCFGRST bit of RCC_APB2RSTR when COMP1LOCK and/or COMP2LOCK bits are set.

2.6 TIM

2.6.1 PWM re-enabled in automatic output enable mode despite of system break

Description

In automatic output enable mode (AOE bit set in TIMx_BDTR register), the break input can be used to do a cycle-by-cycle PWM control for a current mode regulation. A break signal (typically a comparator with a current threshold) disables the PWM output(s) and the PWM is re-armed on the next counter period.

However, a system break (typically coming from the CSS Clock security System) is supposed to stop definitively the PWM to avoid abnormal operation (for example with PWM frequency deviation).

In the current implementation, the timer system break input is not latched. As a consequence, a system break indeed disables the PWM output(s) when it occurs, but PWM output(s) is (are) re-armed on the following counter period.

Workaround

Preferably, implement control loops with the output clear enable function (OCxCE bit in the TIMx_CCMR1/CCMR2 register), leaving the use of break circuitry solely for internal and/or external fault protection (AOE bit reset).

2.6.2 TRGO and TRGO2 trigger output failure

Description

Some reference manual revisions may omit the following information.

The timers can be linked using ITRx inputs and TRGOx outputs. Additionally, the TRGOx outputs can be used as triggers for other peripherals (for example ADC). Since this circuitry is based on pulse generation, care must be taken when initializing master and slave peripherals or when using different master/slave clock frequencies:

- If the master timer generates a trigger output pulse on TRGOx prior to have the destination peripheral clock enabled, the triggering system may fail.
- If the frequency of the destination peripheral is modified on-the-fly (clock prescaler modification), the triggering system may fail.

As a conclusion, the clock of the slave timer or slave peripheral must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are being received from the master timer.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required or applicable as long as the application handles the clock as indicated.

2.6.3 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0
- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = (ARR-1)
  - second (missed) compare event: CNT = CCR = ARR
- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = 1
  - second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.6.4 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.7 LPTIM

2.7.1 Device may remain stuck in LPTIM interrupt when entering Stop mode

Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the device from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the device from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in the relevant RCC register.

2.7.2 Device may remain stuck in LPTIM interrupt when clearing event flag

Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag in LPTIM_ISR register by writing its corresponding bit in LPTIM_ICR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck high.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the device cannot enter Stop mode.

Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

Note: The standard clear sequence implemented in the HAL_LPTIM_IRQHandler in the STM32Cube is considered as the proper clear sequence.

2.7.3 LPTIM events and PWM output are delayed by one kernel clock cycle

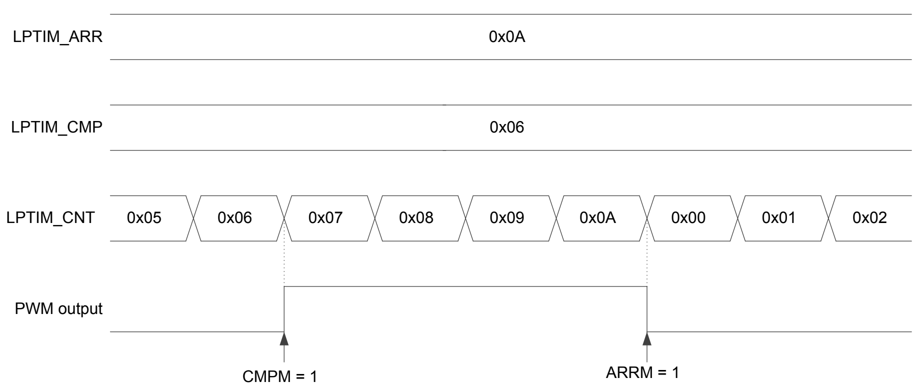
Description

The compare match event (CMPM), auto reload match event (ARRM), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:

Figure 1. Example of PWM output mode



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.8 IWDG

2.8.1 IWDG does not always reset the device

Description

The IWDG must be configured so that the counter starts counting down and generates a reset when the end of count value is reached. In some cases, the configuration does not work: the IWDG remains stopped and does not reset the device.

Workaround

None.

2.9 RTC and TAMP

2.9.1 RTC calendar registers are not locked properly

Description

When reading the calendar registers with BYPSHAD = 0, the RTC_TR and RTC_DR registers may not be locked after reading the RTC_SSR register. This happens if the read operation is initiated one APB clock period before the shadow registers are updated. This can result in a non-consistency of the three registers. Similarly, the RTC_DR register can be updated after reading the RTC_TR register instead of being locked.

Workaround

Apply one of the following measures:

- Use BYPSHAD = 1 mode (bypass shadow registers), or
- If BYPSHAD = 0, read SSR again after reading SSR/TR/DR to confirm that SSR is still the same, otherwise read the values again.

2.9.2 RTC interrupt can be masked by another RTC interrupt

Description

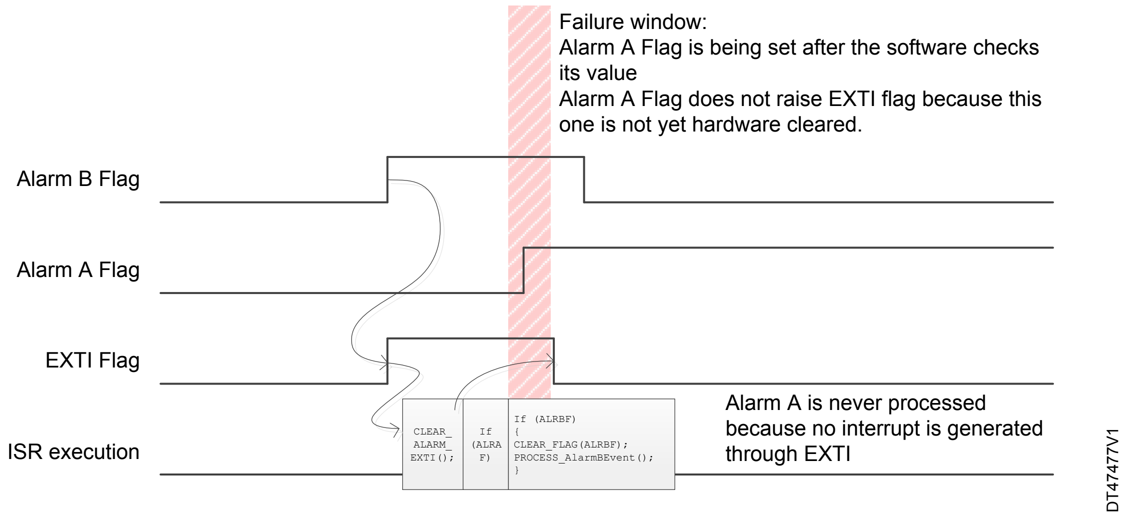
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended interrupt and event controller (EXTI)* section of the reference manual).

The following code example and figure illustrate the failure mechanism: The Alarm A event is lost (fails to generate interrupt) as it occurs in the failure window, that is, after checking the Alarm A event flag but before the effective clear of the EXTI interrupt flag by hardware. The effective clear of the EXTI interrupt flag is delayed with respect to the software instruction to clear it.

Alarm interrupt service routine:

```
void RTC_Alarm_IRQHandler(void)
{
    CLEAR_ALARM_EXTI(); /* Clear the EXTI line flag for RTC alarms*/
    If(ALRAF) /* Check if Alarm A triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the Alarm A interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process Alarm A event */
    }
    If(ALRBF) /* Check if Alarm B triggered ISR */
    {
        CLEAR_FLAG(ALRBF); /* Clear the Alarm B interrupt pending bit */
        PROCESS_AlarmBEvent(); /* Process Alarm B event */
    }
}
```

Figure 2. Masked RTC interrupt



Workaround

In the interrupt service routine, apply three consecutive event flag ckecks - source one, source two, and source one again, as in the following code example:

```
void RTC_Alarm_IRQHandler(void)
{
    CLEAR_ALARM_EXTI(); /* Clear the EXTI's line Flag for RTC Alarm */
    If(ALRAF) /* Check if AlarmA triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the AlarmA interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process AlarmA Event */
    }
    If(ALRBF) /* Check if AlarmB triggered ISR */
    {
        CLEAR_FLAG(ALRBF); /* Clear the AlarmB interrupt pending bit */
        PROCESS_AlarmBEvent(); /* Process AlarmB Event */
    }
    If(ALRAF) /* Check if AlarmA triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the AlarmA interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process AlarmA Event */
    }
}
```

2.9.3 Calendar initialization may fail in case of consecutive INIT mode entry

Description

If the INIT bit of the RTC_ISR register is set between one and two RTCCLK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLK cycles), and the initialization of registers may fail.

Depending on the INIT bit clearing and setting instants versus the RTCCLK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLK period then set again. As writes to calendar registers are ignored when INITF is low, a write during this critical period might result in the corruption of one or more calendar registers.

Workaround

After existing the initialization mode, clear the BYPSHAD bit (if set) then wait for RSF to rise, before entering the initialization mode again.

Note: It is recommended to write all registers in a single initialization session to avoid accumulating synchronization delays.

2.9.4 Alarm flag may be repeatedly set when the core is stopped in debug

Description

When the core is stopped in debug mode, the clock is supplied to subsecond RTC alarm downcounter even when the device is configured to stop the RTC in debug.

As a consequence, when the subsecond counter is used for alarm condition (the MASKSS[3:0] bitfield of the RTC_ALRMASSR and/or RTC_ALRMBSSR register set to a non-zero value) and the alarm condition is met just before entering a breakpoint or printf, the ALRAF and/or ALRBF flag of the RTC_SR register is repeatedly set by hardware during the breakpoint or printf, which makes any attempt to clear the flag(s) ineffective.

Workaround

None.

2.9.5 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF

Description

With the timestamp on tamper event enabled (TAMPTS bit of the RTC_CR register set), a tamper event is ignored if it occurs:

- within four APB clock cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, while the TSF flag is not yet effectively cleared (it fails to set the TSOVF flag)
- within two ck_apre cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, when the TSF flag is effectively cleared (it fails to set the TSF flag and timestamp the calendar registers)

Workaround

None.

2.10 I2C

2.10.1 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

Description

An I^2C-bus master generates STOP condition upon non-acknowledge of I^2C address that it sends. This applies to 7-bit address as well as to each byte of 10-bit address.

When the MCU set as I^2C-bus master transmits a 10-bit address of which the first byte (5-bit header + 2 MSBs of the address + direction bit) is not acknowledged, the MCU duly generates STOP condition but it then cannot start any new I^2C-bus transfer. In this spurious state, the NACKF flag of the I2C_ISR register and the START bit of the I2C_CR2 register are both set, while the START bit should normally be cleared.

Workaround

In 10-bit-address master mode, if both NACKF flag and START bit get simultaneously set, proceed as follows:

1. Wait for the STOP condition detection (STOPF = 1 in I2C_ISR register).
2. Disable the I2C peripheral.
3. Wait for a minimum of three APB cycles.
4. Enable the I2C peripheral again.

2.10.2 Wrong behavior in Stop mode

Description

The correct use of the I2C peripheral is to disable it (PE = 0) before entering Stop mode, and re-enable it when back in Run mode.

Some reference manual revisions may omit this information.

Failure to respect the above while the MCU operating as slave or as master in multi-master topology enters Stop mode during a transfer ongoing on the I^2C-bus may lead to the following:

1. BUSY flag is wrongly set when the MCU exits Stop mode. This prevents from initiating a transfer in master mode, as the START condition cannot be sent when BUSY is set.
2. If clock stretching is enabled (NOSTRETCH = 0), the SCL line is pulled low by I2C and the transfer stalled as long as the MCU remains in Stop mode.
   The occurrence of such condition depends on the timing configuration, peripheral clock frequency, and I^2C-bus frequency.

This is a description inaccuracy issue rather than a product limitation.

Workaround

No application workaround is required.

2.10.3 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period

Description

The I^2C-bus specification and user manual specify a minimum data setup time ($t_{SU;DAT}$) as:

- 250 ns in Standard mode
- 100 ns in Fast mode
- 50 ns in Fast mode Plus

The device does not correctly sample the I^2C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^2C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

Increase the I2C kernel clock frequency to get I2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I^2C-bus standard, the minimum I2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the I2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the I2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the I2CCLK frequency must be at least 20 MHz.

2.10.4 Spurious bus error detection in master mode

Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I^2C-bus transfer in master mode and any such transfer continues normally.

Workaround

If a bus error interrupt is generated in master mode, the BERR flag must be cleared by software. No other action is required and the ongoing transfer can be handled normally.

2.10.5 Last-received byte loss in reload mode

Description

If in master receiver mode or slave receive mode with SBC = 1 the following conditions are all met:

- I²C-bus stretching is enabled (NOSTRETCH = 0)
- RELOAD bit of the I2C_CR2 register is set
- NBYTES bitfield of the I2C_CR2 register is set to N greater than 1
- byte N is received on the I²C-bus, raising the TCR flag
- N - 1 byte is not yet read out from the data register at the instant TCR is raised,

then the SCL line is pulled low (I²C-bus clock stretching) and the transfer of the byte N from the shift register to the data register inhibited until the byte N-1 is read and NBYTES bitfield reloaded with a new value, the latter of which also clears the TCR flag. As a consequence, the software cannot get the byte N and use its content before setting the new value into the NBYTES field.

Workaround

- In master mode or in slave mode with SBC = 1, use the reload mode with NBYTES = 1.
- In master receiver mode, if the number of bytes to transfer is greater than 255, do not use the reload mode. Instead, split the transfer into sections not exceeding 255 bytes and separate them with repeated START conditions.
- Make sure, for example through the use of DMA, that the byte N - 1 is always read before the TCR flag is raised.

The last workaround in the list must be evaluated carefully for each application as the timing depends on factors such as the bus speed, interrupt management, software processing latencies, and DMA channel priority.

2.10.6 Spurious master transfer upon own slave address match

Description

When the device is configured to operate at the same time as master and slave (in a multi- master I²C-bus application), a spurious master transfer may occur under the following condition:

- Another master on the bus is in process of sending the slave address of the device (the bus is busy).
- The device initiates a master transfer by bit set before the slave address match event (the ADDR flag set in the I2C_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write I2C_CR2 before clearing the ADDR flag, or
  - the device writes I2C_CR2 earlier than three I2C kernel clock cycles before clearing the ADDR flag

In these circumstances, even though the START bit is automatically cleared by the circuitry handling the ADDR flag, the device spuriously proceeds to the master transfer as soon as the bus becomes free. The transfer configuration depends on the content of the I2C_CR2 register when the master transfer starts. Moreover, if the I2C_CR2 is written less than three kernel clocks before the ADDR flag is cleared, the I2C peripheral may fall into an unpredictable state.

Workaround

Upon the address match event (ADDR flag set), apply the following sequence.

Normal mode (SBC = 0):

1. Set the ADDRCF bit.
2. Before Stop condition occurs on the bus, write I2C_CR2 with the START bit low.

Slave byte control mode (SBC = 1):

1. Write I2C_CR2 with the slave transfer configuration and the START bit low.
2. Wait for longer than three I2C kernel clock cycles.
3. Set the ADDRCF bit.
4. Before Stop condition occurs on the bus, write I2C_CR2 again with its current value.

The time for the software application to write the I2C_CR2 register before the Stop condition is limited, as the clock stretching (if enabled), is aborted when clearing the ADDR flag.

Polling the BUSY flag before requesting the master transfer is not a reliable workaround as the bus may become busy between the BUSY flag check and the write into the I2C_CR2 register with the START bit set.

2.10.7 START bit is cleared upon setting ADDRCF, not upon address match

Description

Some reference manual revisions may state that the START bit of the I2C_CR2 register is cleared upon slave address match event.

Instead, the START bit is cleared upon setting, by software, the ADDRCF bit of the I2C_ICR register, which does not guarantee the abort of master transfer request when the device is being addressed as slave. This product limitation and its workaround are the subject of a separate erratum.

Workaround

No application workaround is required for this description inaccuracy issue.

2.10.8 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I^{2}C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.10.9 Transmission stalled after first byte transfer

Description

When the first byte to transmit is not prepared in the TXDATA register, two bytes are required successively, through TXIS status flag setting or through a DMA request. If the first of the two bytes is written in the I2C_TXDR register in less than two I2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and I2C kernel clock frequencies is between 1.5 and 3, the second byte written in the I2C_TXDR is not internally detected. This causes a state in which the I2C peripheral is stalled in master mode or in slave mode, with clock stretching enabled (NOSTRETCH = 0). This state can only be released by disabling the peripheral (PE = 0) or by resetting it.

Workaround

Apply one of the following measures:

- Write the first data in I2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the I2C kernel clock frequency is lower than 1.5 or higher than 3.

2.10.10 SDA held low upon SMBus timeout expiry in slave mode

Description

For the slave mode, the SMBus specification defines $t_{TIMEOUT}$ (detect clock low timeout) and $t_{LOW:SEXT}$ (cumulative clock low extend time) timeouts. When one of them expires while the I2C peripheral in slave mode drives SDA low to acknowledge either its address or a data transmitted by the master, the device is expected to report such an expiry and release the SDA line.

However, although the device duly reports the timeout expiry, it fails to release SDA. This stalls the I^2C bus and prevents the master from generating RESTART or STOP condition.

Workaround

When a timeout is reported in slave mode (TIMEOUT bit of the I2C_ISR register is set), apply this sequence:

1. Wait until the frame is expected to end.
2. Read the STOPF bit of the I2C_ISR register. If it is low, reset the I2C kernel by clearing the PE bit of the I2C_CR1 register.
3. Wait for at least three APB clock cycles before enabling again the I2C peripheral.

2.11 USART

2.11.1 RTS is active while RE = 0 or UE = 0

Description

The RTS line is driven low as soon as RTSE bit is set, even if the USART is disabled (UE = 0) or the receiver is disabled (RE = 0), that is, not ready to receive data.

Workaround

Upon setting the UE and RE bits, configure the I/O used for RTS into alternate function.

2.11.2 Receiver timeout counter wrong start in two-stop-bit configuration

Description

In two-stop-bit configuration, the receiver timeout counter starts counting from the end of the second stop bit of the last character instead of starting from the end of the first stop bit.

Workaround

Subtract one bit duration from the value in the RTO bitfield of the USARTx_RTOR register.

2.11.3 Data corruption due to noisy receive line

Description

In all modes, except synchronous slave mode, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

Apply one of the following measures:

- Either use a noiseless receive line, or
- add a filter to remove the glitches if the receive line is noisy.

2.11.4 Received data may be corrupted upon clearing the ABREN bit

Description

The USART receiver may miss data or receive corrupted data when the auto baud rate feature is disabled by software (ABREN bit cleared in the USART_CR2 register) after an auto baud rate detection, while a reception is ongoing.

Workaround

Do not clear the ABREN bit.

2.11.5 Noise error flag set while ONEBIT is set

Description

When the ONEBIT bit is set in the USART_CR3 register (one sample bit method is used), the noise error (NE) flag must remain cleared. Instead, this flag is set upon noise detection on the START bit.

Workaround

None.

Note: Having noise on the START bit is contradictory with the fact that the one sample bit method is used in a noise free environment.

2.11.6 DMA channel 3 (CH3) not functional when USART2_RX used for data reception

Description

When USART2 uses DMA channel 3 for data reception, data transfers are blocked after the first byte has been received. As a result, DMA channel 3 cannot be used for USART2 data reception.

Workaround

Use DMA channel 5 (CH5) for USART2 data reception.

2.12 LPUART

2.12.1 DMA channel 5 (CH5) not functional when LPUART1_RX used for data reception

Description

When LPUART1 uses DMA channel 5 for data reception, the data transfer is blocked after the first byte has been received. As a result, DMA channel 5 cannot be used for LPUART1 data reception.

Workaround

Use DMA channel 3 (CH3) for LPUART1 data reception.

2.13 SPI

2.13.1 BSY bit may stay high when SPI is disabled

Description

The BSY flag may remain high upon disabling the SPI while operating in:

- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

Workaround

When the SPI operates in:

- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

2.13.2 BSY bit may stay high at the end of data transfer in slave mode

Description

BSY flag may sporadically remain high at the end of a data transfer in slave mode. This occurs upon coincidence of internal CPU clock and external SCK clock provided by master.

In such an event, if the software only relies on BSY flag to detect the end of SPI slave data transaction (for example to enter low-power mode or to change data line direction in half-duplex bidirectional mode), the detection fails.

As a conclusion, the BSY flag is unreliable for detecting the end of data transactions.

Workaround

Depending on SPI operating mode, use the following means for detecting the end of transaction:

- When NSS hardware management is applied and NSS signal is provided by master, use NSS flag.
- In SPI receiving mode, use the corresponding RXNE event flag.
- In SPI transmit-only mode, use the BSY flag in conjunction with a timeout expiry event. Set the timeout such as to exceed the expected duration of the last data frame and start it upon TXE event that occurs with the second bit of the last data frame. The end of the transaction corresponds to either the BSY flag becoming low or the timeout expiry, whichever happens first.

Prefer one of the first two measures to the third as they are simpler and less constraining.

Alternatively, apply the following sequence to ensure reliable operation of the BSY flag in SPI transmit mode:

1. Write last data to data register.
2. Poll the TXE flag until it becomes high, which occurs with the second bit of the data frame transfer.
3. Disable SPI by clearing the SPE bit mandatorily before the end of the frame transfer.
4. Poll the BSY bit until it becomes low, which signals the end of transfer.

Note: The alternative method can only be used with relatively fast CPU speeds versus relatively slow SPI clocks or/and long last data frames. The faster is the software execution, the shorter can be the duration of the last data frame.

2.13.3 Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback

Description

In master receive transaction, the last bit of the transacted frame is not captured when signal provided by internal feedback loop from the SCK pin exceeds a critical delay. The lastly transacted bit of the stored data then keeps value from the previously received pattern. As a consequence, the last receive data bit may be wrong and/or the CRCERR flag can be unduly asserted if any data under check sum and/or the CRC pattern is wrongly captured.

A delay of up to two APB clock periods can be tolerated for the internal feedback delay.

Main factors contributing to the delay increase are low VDD level, high temperature, high SCK pin capacitive load and low SCK I/O output speed. The SPI communication speed has no impact.

The following table gives the maximum allowable APB frequency versus GPIOx_OSPEEDR output speed bitfield setting for the SCK pin, at 30pF of capacitive load. The operation is safe up to that frequency.

Workaround

The following measures can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the I/O pad of the SCK pin to higher speed.

2.13.4 Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters

Description

When SPI is handled by DMA in full-duplex master or slave mode with CRC enabled, the CRC computation may temporarily freeze for the ongoing frame, which results in corrupted CRC.

This happens when the receive counter reaches zero upon the receipt of the CRC pattern (as the receive counter was set to a value greater, by CRC length, than the transmit counter). An internal signal dedicated to receive-only mode is left unduly pending. Consequently, the signal can cause the CRC computation to freeze during a next transaction in which DMA TXE event service is accidentally delayed (for example, due to DMA servicing a request from another channel).

Workaround

Apply one of the following measures prior to each full-duplex SPI transaction:

- Set the DMA transmission and reception data counters to equal values. Upon the transaction completion, read the CRC pattern out from RxFIFO separately by software.
- Reset the SPI peripheral via peripheral reset register.

2.13.5 CRC error in SPI slave mode if internal NSS changes before CRC transfer

Description

Some reference manual revisions may omit the information that the device operating as SPI slave must be configured in software NSS control if the SPI master pulses the NSS (for (for example in NSS pulse mode).

Otherwise, the transition of the internal NSS signal after the CRCNEXT flag is set might result in wrong CRC value computed by the device and, as a consequence, in a CRC error. As a consequence, the NSS pulse mode cannot be used along with the CRC function.

This is a documentation error rather than a product limitation.

Workaround

No application workaround is required as long as the device operating as SPI slave is duly configured in software NSS control.

2.13.6 Anticipated communication upon SPI transit from slave receiver to master

Description

Regardless of the master mode configured, the communication clock starts upon setting the MSTR bit even though the SPI is disabled, if transiting from receive-only (RXONLY = 1) or half-duplex receive (BIDIMODE = 1 and BIDIOE = 0) slave mode to master mode.

Workaround

Apply one of the following measures:

- Before transiting to master mode, hardware-reset the SPI via the reset controller.
- Set the MSTR and SPE bits of the SPI configuration register simultaneously, which forces the immediate start of the communication clock. In transmitter configuration, load the data register in advance with the data to send.

Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture (www.psacertified.org) and/or Security Evaluation standard for IoT Platforms (www.trustcb.com). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on www.st.com for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

Revision history

Table 5. Document revision history

Date	Version	Changes
30-Nov-2015	1	Initial release.
02-Feb-2016	2	Updated Section 2.1.5: NSS pin synchronization required when using bootloader with SPI1 interface on TSSOP14 package and extended to device revision Z. Added Section 2.2.1: Overrun flag might not be set when converted data have not been read before new data are written.
14-Nov-2016	3	Section 2.5: I2C peripheral limitations: • Added Section 2.5.3: 10-bit master Master mode: new transfer cannot be launched if first part of the address has not been acknowledged by the slave. Section 2.6: SPI peripheral limitations: • Updated Section 2.6.1: BSY bit may stay high at the end of a SPI data transfer in Slave mode. • Updated Section 2.6.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback. • Added Section 2.6.3: Wrong CRC transmitted in Master mode with delayed SCK feedback Section 2.7: USART limitations: • Added Section 2.7.2: DMA channel 3 (CH3) not functional when USART2_RX used for data reception. Added Section 2.8: LPUART limitation and Section 2.8.1: DMA channel 5 (CH5) not functional when LPUART1_RX used for data reception.
25-Nov-2021	4	Distinction made between device and documentation errata. Added DMA, TIM, LPTIM errata. Updated ADC, RTC and TAMP, I2C, and SPI errata. Distinction made between USART and LPUART and USART errata updated.
2-Feb-2022	5	Updated Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield erratum. Renamed SPI/I2S section into SPI and removed limitation *I2S slave in PCM short pulse mode sensitive to timing between WS and CK*.
01-Aug-2024	6	Added errata: • Wake-up sequence from Standby mode fails when using more than one wake-up source • LSE crystal oscillator may be disturbed by transitions on PC13 • GPIO locking mechanism failed if critical address is read • Writing ADC_CFGR2 register while ADEN bit is set resets CKMODE[1:0] bitfield • A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF • SDA held low upon SMBus timeout expiry in slave mode • Received data may be corrupted upon clearing the ABREN bit • Noise error flag set while ONEBIT is set

Contents

1 **Summary of device errata** 2

2 **Description of device errata** 4

2.1 System 4

2.1.1 Delay after an RCC peripheral clock enabling 4

2.1.2 I2C and USART cannot wake up the device from Stop mode 4

2.1.3 LDM, STM, PUSH and POP not allowed in IOPORT bus 4

2.1.4 BOOT_MODE bits do not reflect the selected boot mode 5

2.1.5 NSS pin synchronization required when using bootloader with SPI1 interface on TSSOP14 package 5

2.1.6 Wake-up sequence from Standby mode fails when using more than one wake-up source . 5

2.1.7 LSE crystal oscillator may be disturbed by transitions on PC13 5

2.2 GPIO 6

2.2.1 GPIO locking mechanism failed if critical address is read 6

2.3 DMA 7

2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear 7

2.3.2 Byte and half-word accesses not supported 7

2.4 ADC 7

2.4.1 Overrun flag is not set if EOC reset coincides with new conversion end 7

2.4.2 Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield 7

2.4.3 Out-of-threshold value is not detected in AWD1 Single mode 8

2.4.4 Writing ADC_CFGR2 register while ADEN bit is set resets CKMODE[1:0] bitfield 8

2.5 COMP 8

2.5.1 COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register 8

2.6 TIM 9

2.6.1 PWM re-enabled in automatic output enable mode despite of system break 9

2.6.2 TRGO and TRGO2 trigger output failure 9

2.6.3 Consecutive compare event missed in specific conditions 9

2.6.4 Output compare clear not working with external counter reset 10

2.7 LPTIM 10

2.7.1 Device may remain stuck in LPTIM interrupt when entering Stop mode 10

2.7.2 Device may remain stuck in LPTIM interrupt when clearing event flag 11

2.7.3 LPTIM events and PWM output are delayed by one kernel clock cycle 11

2.8 IWDG 12

2.8.1 IWDG does not always reset the device 12

2.9 RTC and TAMP 12

2.9.1 RTC calendar registers are not locked properly 12
2.9.2 RTC interrupt can be masked by another RTC interrupt 12
2.9.3 Calendar initialization may fail in case of consecutive INIT mode entry 13
2.9.4 Alarm flag may be repeatedly set when the core is stopped in debug 14
2.9.5 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF 14
2.10 I2C 14
2.10.1 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave 14
2.10.2 Wrong behavior in Stop mode 15
2.10.3 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period 15
2.10.4 Spurious bus error detection in master mode 15
2.10.5 Last-received byte loss in reload mode 16
2.10.6 Spurious master transfer upon own slave address match 16
2.10.7 START bit is cleared upon setting ADDRCF, not upon address match 17
2.10.8 OVR flag not set in underrun condition 17
2.10.9 Transmission stalled after first byte transfer 17
2.10.10 SDA held low upon SMBus timeout expiry in slave mode 18
2.11 USART 18
2.11.1 RTS is active while RE = 0 or UE = 0 18
2.11.2 Receiver timeout counter wrong start in two-stop-bit configuration 18
2.11.3 Data corruption due to noisy receive line. 18
2.11.4 Received data may be corrupted upon clearing the ABREN bit. 19
2.11.5 Noise error flag set while ONEBIT is set 19
2.11.6 DMA channel 3 (CH3) not functional when USART2_RX used for data reception 19
2.12 LPUART 19
2.12.1 DMA channel 5 (CH5) not functional when LPUART1_RX used for data reception 19
2.13 SPI 19
2.13.1 BSY bit may stay high when SPI is disabled 19
2.13.2 BSY bit may stay high at the end of data transfer in slave mode 20
2.13.3 Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback 20
2.13.4 Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters 21
2.13.5 CRC error in SPI slave mode if internal NSS changes before CRC transfer 21
2.13.6 Anticipated communication upon SPI transit from slave receiver to master. 21

Important security notice 22

Revision history 23