AN2953
Application note

How to migrate from the STM32F10xxx firmware library V2.0.3 to the STM32F10xxx standard peripheral library V3.0.0

Introduction

The objective of this application note is to explain how to migrate an application based on the STM32F10xxx firmware library (FWLib) V2.0.3 to the STM32F10xxx standard peripheral library (StdPeriph_Lib) V3.0.0. The purpose of this document is not to provide detailed information on these two versions, but to highlight the differences between them.

Note: Throughout this document, and unless otherwise specified, the term of FWLib will be used to refer to the STM32F10xxx firmware library V2.0.3 and StdPeriph_Lib will be used to refer to the STM32F10xxx standard peripheral library*.*

Glossary

Low-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 16 and 32 Kbytes.

Medium-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 32 and 128 Kbytes.

High-density devices are STM32F101xx, STM32F102xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

Contents

List of tables

List of figures

1 Why migrate from STM32F10xxx FWLib V2.0.3 to StdPeriph_Lib V3.0.0?

The STM32F10xxx FWLib V2.0.3 is a complete firmware package for the STM32F10xxx low-, medium- and high-density devices. It is a collection of routines, data structures and macros that covers the features of all peripherals. It includes drivers and a set of examples for all the standard device peripherals.

The STM32F10xxx StdPeriph_Lib V3.0.0 is an update of FWLib V2.0.3:

- It makes the library compliant with the Cortex™ microcontroller software interface standard (CMSIS)
- the package architecture has been enhanced
- the source files are provided in the Doxygen format
- the update has no impact on the STM32 peripheral drivers' API (application programming interface)

Note: Only the STM32F10xxx CAN driver was updated to anticipate the support of STM32F10xxx connectivity line products (products with dual CAN).

To migrate to the STM32F10xxx standard peripheral library V3.0.0, you simply have to update the:

- files relative to your toolchain
- project setting
- library file organization
- you do not need to change or update the application code

The details of all updates made on the library (StdPeriph_Lib) are described below.

1.1 ARM® Cortex-M3™ microcontroller software interface standard (CMSIS) compliance

The CMSIS answers the challenges faced when software components are deployed to physical microcontroller devices based on Cortex-M0 / Cortex-M1 or Cortex-M3 processors. The CMSIS will also be expanded to future Cortex-M processor cores (the standard refers to Cortex-Mx). The CMSIS is defined in close cooperation with various silicon and software vendors and provides a common approach to interface to peripherals, real-time operating systems and middleware components. For more details, please refer to www.onarm.com.

1.1.1 CMSIS description

As part of the CMSIS, ARM provides the following software layers, which are available for various compiler implementations:

- **Core peripheral access layer:** contains name definitions, address definitions and helper functions to access core registers and peripherals. It also defines a device-independent interface for RTOS Kernels that includes debug channel definitions.
- **Middleware access layer:** provides common methods to access peripherals for the software industry. The middleware access layer is adapted by the Silicon vendor to the device-specific peripherals used by the middleware components. The middleware access layer is currently in development and not yet part of this documentation.

These software layers are expanded by Silicon partners with:

- **a device peripheral access layer:** provides definitions for all device peripherals
- **access functions for peripherals (optional):** provides additional helper functions for peripherals

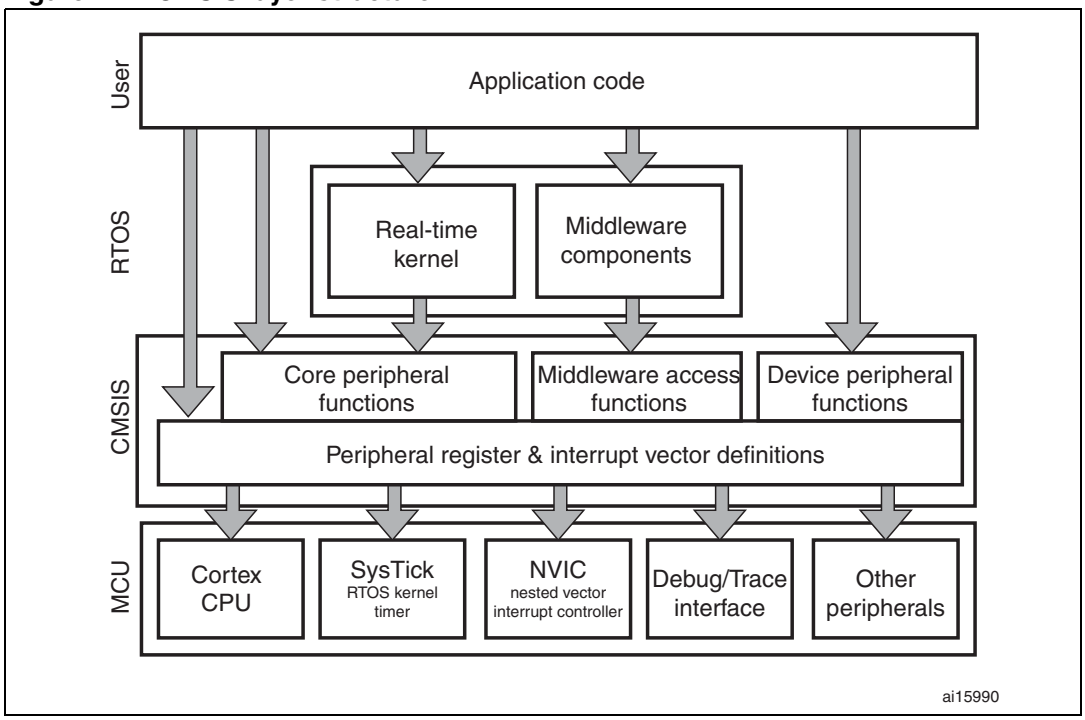
For a Cortex-Mx microcontroller system, CMSIS defines:

- A common way of accessing peripheral registers and a common way of defining exception vectors
- The register names of the core peripherals and the names of the core exception vectors
- A device-independent interface for RTOS Kernels including a debug channel
- Interfaces for middleware components (TCP/IP Stack, flash file system)

1.1.2 CMSIS structure

Figure 1 illustrates different layers for a CMSIS-based application.

Figure 1. CMSIS layer structure



CMSIS – Files for the peripheral access layer

Compiler vendor-independent files:

- Cortex-M3 core & peripheral file (*core_cm3.h* + *core_cm3.c*)
  - Access to Cortex-M3 core & peripherals: NVIC, SysTick, etc.
  - Functions to access Cortex-M3 CPU registers and core peripherals
- Device-specific header file (*device.h*)
  - Interrupt number assignment (consistent with startup file)
  - Peripheral register definitions (layout & base addresses)
  - Functions that control other chip-specific functions (optional)
- Device-specific system file (*system_device.c*)
  - `SystemInit` function that initializes the physical microcontroller device
  - `SystemInit_ExtMemCtl`: function that sets up the external memory controller. It is called in *startup_stm32f10x_xx.s/.c* before jumping to main.
  - SystemFrequency value for system-wide timing
  - Other device-related features (optional)

Compiler-vendor + device-specific startup file:

- Compiler startup code (assembly or C) (*startup_device.s*)
  - Interrupt handler table with device-specific names (consistent with header)
  - Weak interrupt handler default functions (can be overwritten by user code)

Note: The Weak keyword instructs the compiler to export symbols weakly. This keyword can be applied to function and variable declarations, and to function definitions.

Functions defined with Weak export their symbols weakly. A weakly defined function behaves like a normally defined function unless a non-weakly defined function of the same name is linked to the same image. If both a non-weakly defined function and a weakly defined function exist in the same image then all calls to the function resolve to call the non-weak function. If multiple weak definitions are available, the linker chooses one for use by all calls.

1.1.3 STM32F10xxx firmware library V2.0.3 versus CMSIS V1.10

CMSIS provides a different implementation of some STM32F10xxx FWLib components. Here are the main differences:

- Use of standard C types, <*stdint.h*> file
- For each Cortex-M3 exception and STM32 IRQ, there will be:
  - an exception/interrupt handler with the **_Handler** postfix (for exceptions) or the **_IRQHandler** postfix (for interrupts)
  - a default exception/interrupt handler (weak definition) that contains an endless loop
  - a `#define` of the interrupt number with the **_IRQn** postfix
- Startup file renamed to ***startup_stm32f10x_xx.s/.c***, where xx can be: *hd* (high-density, *md* (medium-density) or *ld* (low-density)
- Only reduced NVIC and SysTick functions are available, some useful functions will be added in a new driver in the STM32F10xxx standard peripheral library, named ***misc.h/.c***
- Some macro names are different from those used in STM32F10xxx FWLib V2.0.3. (*Table 1*)

Table 1. STM32F10xxx FWLib V2.0.3 macros vs. CMSIS macros(1)

STM32 macros	CMSIS macros	STM32 macros	CMSIS macros
-	__NOP	**__RESETPRIMASK**	**__enable_irq**
__WFI	__WFI	**__SETPRIMASK**	**__disable_irq**
__WFE	__WFE	**__READ_PRIMASK**	**__get_PRIMASK**
__SEV	__SEV		**__set_PRIMASK(val)**
__ISB	__ISB	**__RESETFAULTMASK**	**__enable_fault_irq**
__DSB	__DSB	**__SETFAULTMASK**	**__disable_fault_irq**
__DMB	__DMB	**__READ_FAULTMASK**	**__get_FAULTMASK**
__SVC	**-**		**__set_FAULTMASK(val)**
__MRS_CONTROL	**__get_CONTROL**	**__BASEPRICONFIG**	**__set_BASEPRI**
__MSR_CONTROL	**__set_CONTROL**	**__GetBASEPRI**	**__get_BASEPRI**
__MRS_PSP	**__get_PSP**	**__REV_HalfWord**	**__REV16**
__MSR_PSP	**__set_PSP**	**__REV_Word**	**__REV**
__MRS_MSP	**__get_MSP**	**-**	**__REVSH**
__MSR_MSP	**__set_MSP**	**-**	**__RBIT**

1. Bold is used to identify the macros that changed. When the change has an impact on the FWLib (driver or examples), gray shading is added.

1.2 STM32F10xxx standard peripheral library with doxygen format

The STM32F10xxx standard peripheral library comes in a new source file format. Now, all StdPeriph_Lib files use the Doxygen format to easy documentation generation and for more interactive and effective documentation usage.

The existing STM32F10xxx firmware library user manual UM0427 will be removed from the STMicroelectronics MCU website (*www.st.com/mcu*) and replaced by a CHM file presenting all STM32F10xxx standard peripheral library components.

Doxygen example:

```
/**
  * @brief  Reads the specified input port pin.
  * @param GPIOx: where x can be (A..G) to select the GPIO
peripheral.
  * @param GPIO_Pin:  specifies the port bit to read.
  *   This parameter can be GPIO_Pin_x where x can be (0..15).
  * @retval : The input port pin value.
  */
```

Where:

- `@brief`: one-line brief function overview
- `@param`: detailed parameter explanation
- `@retval`: detailed information about return values

For more details, refer to the "*stm32f10x_stdperiph_lib_um.chm*" file.

1.3 STM32F10xxx standard peripheral library architecture

The STM32F10xxx standard peripheral library architecture is enhanced with CMSIS layer support.

The StdPeriph_Lib usage is now based on two approaches that take into account the application needs:

- it uses the peripheral drivers: in this case product programming is based on the drivers' API (application programming interface). You only have to configure the "*stm32f10x_conf.h*" file and use the corresponding *stm32f10x_ppp.h/.c* files.
- it does not use the peripheral drivers: in this case product programming is based on the peripheral register structure and bit definition file

The StdPeriph_Lib supports all STM32F10xxx family products: STM32F10xxx high-, medium- and low-density devices. The StdPeriph_Lib is configurable for the whole family products through preprocessor `defines`, one `define` per product. `Defines` are available for the following products:

- STM32F10x_LD: STM32 low-density devices
- STM32F10x_MD: STM32 medium-density devices
- STM32F10x_HD: STM32 high-density devices

The scope of these `defines` is:

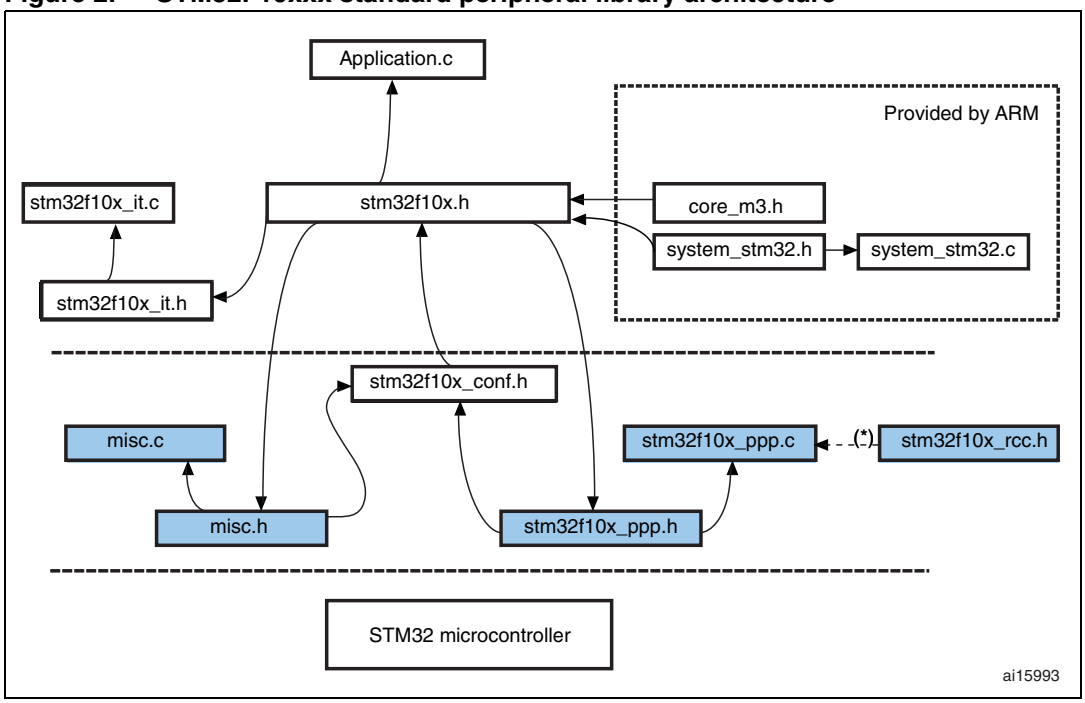
- Interrupt IRQ channel definition inside the *stm32f10x.h* file
- Vector table, one startup file per product
- Peripheral memory mapping and physical register address definition
- Product configuration: external quartz (HSE) value, etc.
- System configuration functions
- Features with different/incompatible implementations across the family

These `defines` do not apply to peripheral drivers, these drivers will always support the features of the family superset.

1.4 STM32F10xxx standard peripheral library architecture: file inclusion

Figure 2 illustrates the STM32F10xxx file inclusion.

Figure 2. STM32F10xxx standard peripheral library architecture



1.5 STM32F10xxx FWLib V2.0.3 legacy

The STM32F10xxx FWLib V2.0.3 and all related firmware will be kept on the STM32™ website (*http://www.st.com/mcu/familiesdocs-110.html*). They will be zipped together into a single file "*STM32F10x_FW_Archive.zip*", available below the “Firmware” document category reachable at: *http://www.st.com/mcu/familiesdocs-110.html#Firmware*.

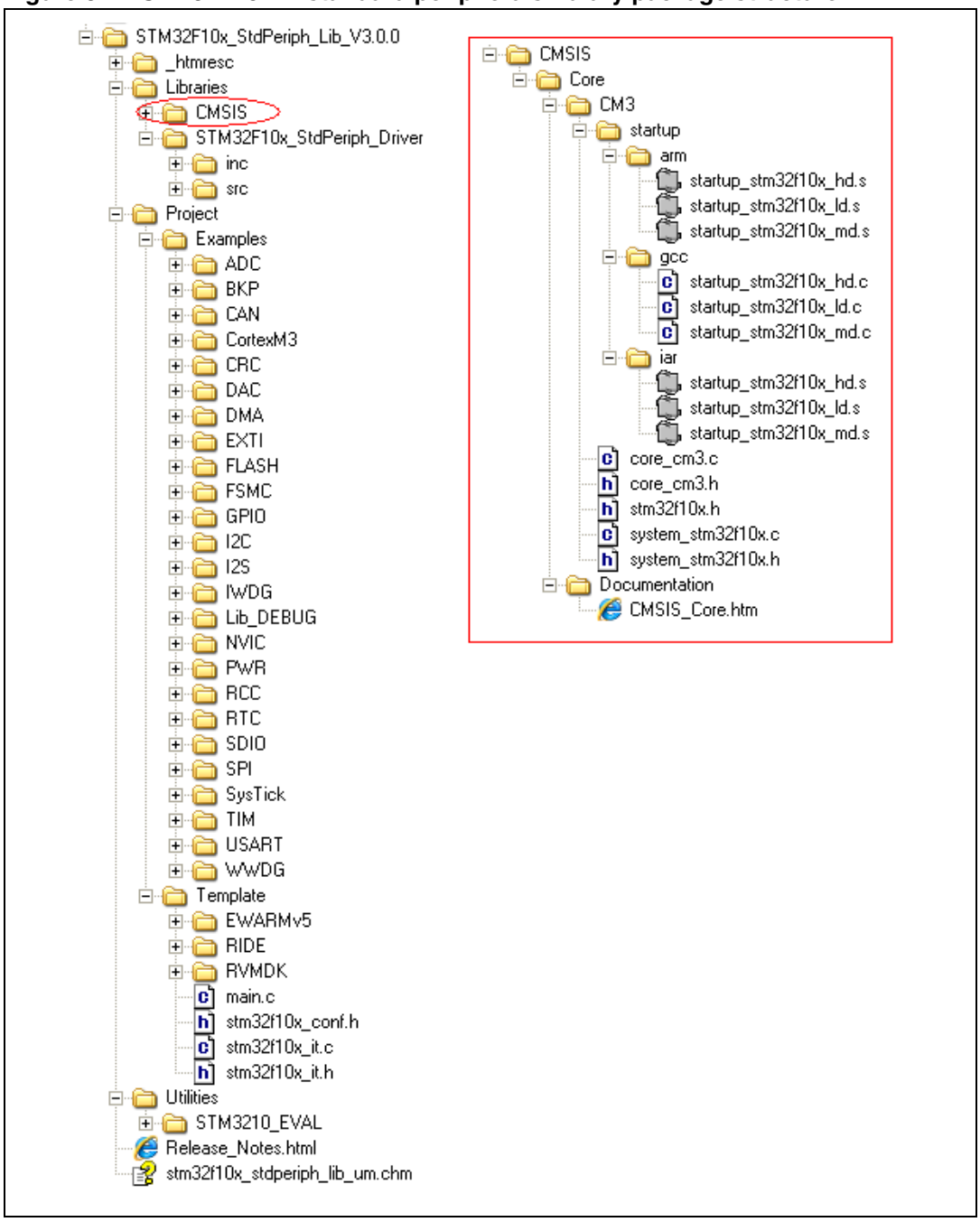
In addition to the archive file, a library patch called "*STM32F10xFWLib_V2.0.3_Patch1.zip*" is also provided. This patch fixes all FWLib V2.0.3 limitations.

2 STM32F10xxx standard peripheral library package

For better flexibility and improved structure purposes, the STM32F10xxx firmware library package was updated to include a specific folder for the CMSIS files and peripheral layer. The package is renamed to *STM32F10x_StdPeriph_Lib_VX.Y.Z* for STM32F10xxx standard peripherals library.

The new package architecture is illustrated in *Figure 3*.

Figure 3. STM32F10xxx standard peripherals library package structure



New package description

Table 2 describes all the new folders in the STM32F10xxx standard peripheral library package.

Table 2. STM32F10x_StdPeriph_Lib package folder description

STM32F10x_StdPeriph_Lib								
Utilities	**Project**				**Libraries**			**_htmresc**
Template	**Template**			**Examples**	**STM32F10x_StdPeriph_Driver**		**CMSIS**	
STM3210-EVAL	RVMDK	RIDE	EWARMv5	A full set examples for STM32F10xxx Standard peripheral drivers	src	inc	See *Table 3*	This folder contains all package html page resources
This folder contains a specific driver for the STM3210E-EVAL and STM3210B-EVAL evaluation boards	Template project Example for KEIL RVMDK	Template project Example for Raisonance RIDE	Template project Example for IAR EWARMv5		Source files for STM32F10xxx Standard peripheral drivers	Include files for STM32F10xxx Standard peripheral drivers		

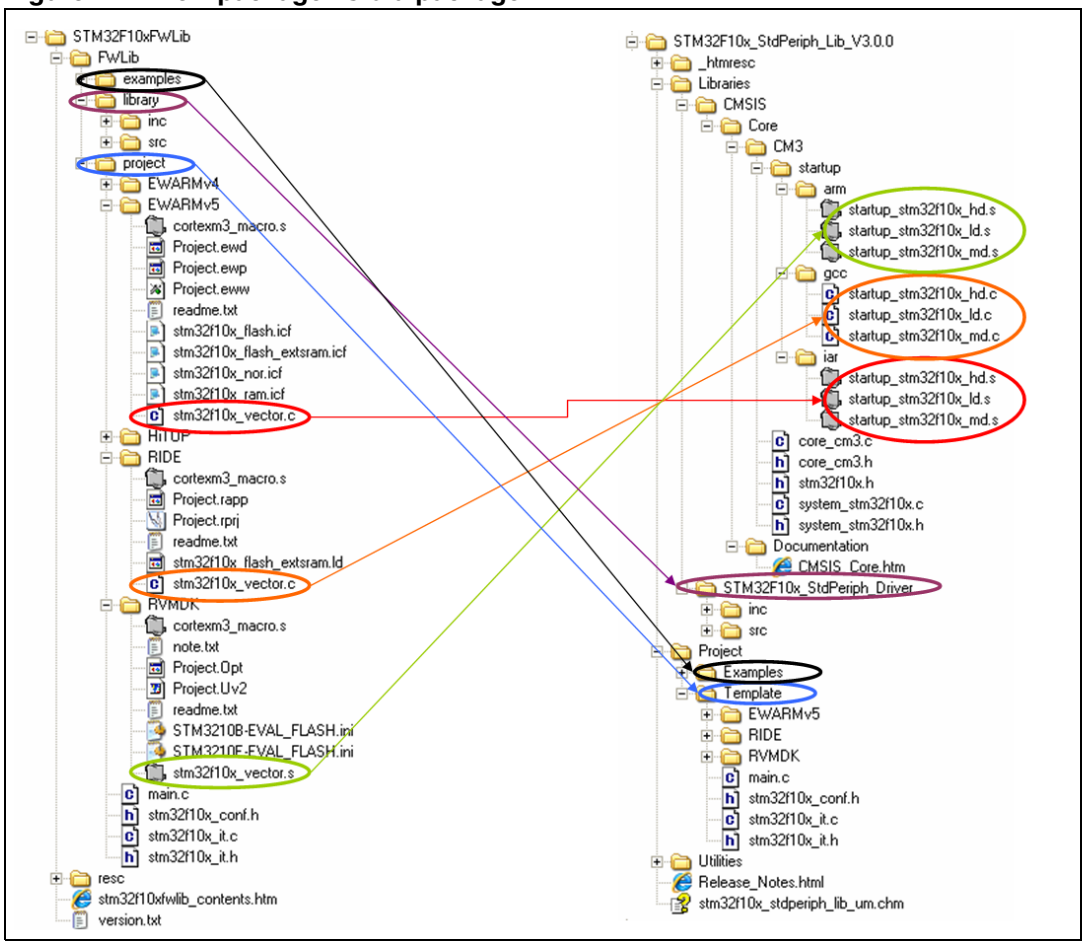
Table 3 describes the CMSIS folder structure.

Table 3. CMSIS folder structure

<table>
<tr><th colspan="5">CMSIS</th></tr>
<tr><td colspan="5">Core</td></tr>
<tr><td rowspan="2">Documentation</td><td colspan="3">CM3</td><td rowspan="3">This folder contains the STM32F10xxx CMSIS files: device peripheral access layer and core peripheral access layer
– core_cm3.h: CMSIS Cortex-M3 core peripheral access layer header file
– core_cm3.c: CMSIS Cortex-M3 core peripheral access layer source file
– stm32f10x.h: CMSIS Cortex-M3 STM32F10xxx device peripheral access layer header file
– system.stm32f10x.h: CMSIS Cortex-M3 STM32F10xxx device peripheral access layer system header file
– system.stm32f10x.c: CMSIS Cortex-M3 STM32F10xxx device peripheral access layer system source file</td></tr>
<tr><td colspan="3">Startup</td></tr>
<tr><td rowspan="2">CMSIS Documentation</td><td>iar</td><td>gcc</td><td>arm</td></tr>
<tr><td>IAR Compiler Startup files for STM32F10xxx:
– startup_stm32f10x_hd.s: high-density device startup file
– startup_stm32f10x_md.s: medium-density device startup file
– startup_stm32f10x_ld.s: low-density device startup file</td><td>GCC Compiler Startup files for STM32F10xxx:
– startup_stm32f10x_hd.c: high-density device startup file
– startup_stm32f10x_md.c: medium-density device startup file
– startup_stm32f10x_ld.c: low-density device startup file</td><td>ARM Compiler Startup files for STM32F10xxx:
– startup_stm32f10x_hd.s: high-density device startup file
– startup_stm32f10x_md.s: medium-density device startup file
– startup_stm32f10x_ld.s: low-density device startup file</td></tr>
</table>

Old STM32F10xxx FWLib package vs. new STM32F10xxx StdPeriph_Lib package

Figure 4. New package vs old package



3 STM32F10xxx standard peripheral library list of changes

3.1 STM32F10xxx standard peripheral library files

3.1.1 Library core files

- The *stm32f10x_map.h* file was renamed as *stm32f10x.h*. It contains:
  - STM32 interrupt IRQ list
  - Specific options for the Cortex-M3 core
  - STM32 peripheral memory mapping and physical register address definition
  - A specific `define` storing the STM32F10xxx standard peripheral library version: "`__STM32F10X_STDPERIPH_VERSION`"
  - Configuration options:

  a) The application has to select the STM32 product it is operating with, only one `define` per product

  b) The application has to select if the peripheral drivers are to be used or not
- The library Debug mode was removed: it is no longer possible to view the peripheral registers in a debugger watch window. However, it is now possible to access registers using specific tool chain debugger utilities. As a consequence, the following files were removed/updated:
  - *main.c*: `#ifdef DEBUG` was replaced by `#ifdef USE_FULL_ASSERT`
  - *stm32f10x_lib.h* was removed and its content were included in *stm32f10x_conf.h*
  - *stm32f10x_lib.c* file was removed
  - *stm32f10x_conf.h*:

  a) "`#define DEBUG 1`" was removed and a specific `define` for the full assert function was added: "`#define USE_FULL_ASSERT 1`"

  b) Peripheral instance `defines` (e.g. `#define _USART`, `#define _USART1`, `#define _USART2`) were removed

  c) To include the peripheral header file, you have to uncomment the corresponding line to use the driver functions. For example, uncomment `#include "stm32f10x_spi.h"` to use the SPI driver functions.
  - The *stm32f10x_type.h* file was replaced by the *<stdint.h>* file. Library-specific types were added into the *stm32f10x.h* file (bool, FlagStatus, ITStatus, FunctionalState, ErrorStatus). Old types remain unchanged. They were included in the *stm32f10x.h* file for legacy purposes.
  - *cortexm3_macro.h* and *cortexm3_macro.s* were removed, they are now covered by CMSIS files

Note: 1 A specific example is provided. It is called "Lib_DEBUG" and describes how to define a DEBUG feature for a selected peripheral.

3.1.2 Library peripheral drivers

- NVIC and SysTick drivers were removed, only five useful functions were added as a new driver (*misc.h*/*misc.c*), in addition to the functions covered by the CMSIS core peripheral layer.
  - void `NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup);` for easy Cortex-M3 priority bit configuration
  - void `NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);` for easy NVIC IRQ configuration
  - void `NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset);` used to boot from internal SRAM and to relocate the vector table to a different offset memory
  - void `NVIC_SystemLPConfig(u8 LowPowerMode, FunctionalState NewState);`
  - void `SysTick_CLKSourceConfig(u32 SysTick_CLKSource);`
- CAN driver update: in all CAN driver functions, new parameter **CAN_TypeDef* CANx** add.
- No change in the remaining drivers

3.1.3 Library user and toolchain specific files

- *stm32f10x_vector.s*/*stm32f10x_vector.c* were renamed to *startup_stm32f10x_xx.s*/*startup_stm32f10x_xx.c*, with one startup file per product
  - *startup_stm32f10x_ld.s* (STM32 low-density devices)
  - *startup_stm32f10x_md.s* (STM32 medium-density devices)
  - *startup_stm32f10x_hd.s* (STM32 high-density devices)
- *stm32f10x_it.h*/*stm32f10x_it.c*: all STM32 IRQ handler routines were removed from these files, only Cortex-M3 exceptions are available. The [Weak] directive is mandatory inside the startup file. The user has to manually add the peripheral ISR into the *stm32f10x_it.h/stm32f10x_it.c* file
- Cortex-M3 exception renamed according to CMSIS
- *main.c*: the following code should be removed:

```
#ifdef DEBUG
  debug();
#endif
```

3.1.4 Library examples

- The NVIC CM3_LPModes and System_Handlers examples were removed, the other ones were kept (VectorTable_Relocation, DMA_WFIMode, IRQ_Channels, Priority)
- PWR and Cortex-M3 examples were updated with new macro names
- No change in the remaining examples

3.2 Coding rules and conventions

3.2.1 Data types and IO type qualifiers

The Cortex-Mx HAL uses the standard types from the standard ANSI C header file *<stdint.h>*. IO type qualifiers are used to specify the access to peripheral variables. IO type qualifiers are used for the automatic generation of debug information for the peripheral registers.

Table 4. CMSIS IO type qualifiers

IO type qualifier	#define	Description
__I	volatile const	Read access only
__O	volatile	Write access only
__IO	volatile	Read and write accesses

The "*stm32f10x_type.h*" file being removed from the package, the new library will use the CMSIS and *<stdint.h>* types. *Table 5* shows the correspondence between STM32F10xxx and *<stdint.h>* types.

Table 5. STM32F10xxx firmware library V2.0.3 types vs. CMSIS types

STM32F10xxx FWLib types	CMSIS types	Description
s32	int32_t	signed 32-bit data
s16	int16_t	signed 16-bit data
s8	int8_t	signed 8-bit data
sc32	const int32_t	read access only signed 32-bit data
sc16	const int16_t	read access only signed 16-bit data
sc8	const int8_t	read access only signed 8-bit data
vs32	__IO int32_t	volatile read and write access signed 32-bit data
vs16	__IO int16_t	volatile read and write access signed 16-bit data
vs8	__IO int8_t	volatile read and write access signed 8-bit data
vsc32	__I int32_t	volatile read access only signed 32-bit data
vsc16	__I int16_t	volatile read access only signed 16-bit data
vsc8	__I int8_t	volatile read access only signed 8-bit data
u32	uint32_t	unsigned 32-bit data
u16	uint16_t	unsigned 16-bit data
u8	uint8_t	unsigned 8-bit data
uc32	const uint32_t	read access only unsigned 32-bit data
uc16	const uint16_t	read access only unsigned 16-bit data
uc8	const uint8_t	read access only unsigned 8-bit data
vu32	__IO uint32_t	volatile read and write access unsigned 32-bit data

Table 5. STM32F10xxx firmware library V2.0.3 types vs. CMSIS types (continued)

STM32F10xxx FWLib types	CMSIS types	Description
vu16	__IO uint16_t	volatile read and write access unsigned 16-bit data
vu8	__IO uint8_t	volatile read and write access unsigned 8-bit data
vuc32	__I uint32_t	volatile read access only unsigned 32-bit data
vuc16	__I uint16_t	volatile read access only unsigned 16-bit data
vuc8	__I uint8_t	volatile read access only unsigned 8-bit data

Note: 1 The old STM32F10xxx FWLib types are still defined inside the "stm32f10x.h" file for legacy purposes.

2 The STM32F10xxx FWLib-specific types are also defined in "stm32f10x.h". These types are:

```
typedef enum {FALSE = 0, TRUE = !FALSE} bool;
typedef enum {RESET = 0, SET = !RESET} FlagStatus, ITStatus;
typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;
#define IS_FUNCTIONAL_STATE(STATE) (((STATE) == DISABLE) ||
((STATE) == ENABLE))
typedef enum {ERROR = 0, SUCCESS = !ERROR} ErrorStatus;
```

3.2.2 Exception naming

Table 6 shows the exception handler names which were changed to match CMSIS names.

Table 6. STM32F10xxx FWLib V2.0.3 exception names vs. CMSIS

STM32F10xxx exceptions	CMSIS	Description
NMIException	NMI_Handler	NMI exception
HardFaultException	HardFault_Handler	Hard fault exception
MemManageException	MemManage_Handler	Memory manage exception
BusFaultException	BusFault_Handler	Bus fault exception
UsageFaultException	UsageFault_Handler	Usage fault exception
SVCHandler	SVC_Handler	SVCall exception
DebugMonitor	DebugMon_Handler	Debug monitor exception
PendSVC	PendSV_Handler	PendSVC exception
SysTickHandler	SysTick_Handler	SysTick handler

Table 7 shows the exception handlers that were changed to change CAN to CAN1.

Table 7. STM32F10xxx CAN1 exception renaming

STM32F10xxx exceptions	CMSIS	Description
USB_HP_CAN_TX_IRQHandler	USB_HP_CAN1_TX_IRQHandler	USB high priority or CAN1 TX handler
USB_LP_CAN_RX0_IRQHandler	USB_LP_CAN1_RX0_IRQHandler	USB low priority or CAN1 RX0 handler
CAN_RX1_IRQHandler	CAN1_RX1_IRQHandler	CAN1 RX1 handler
CAN_SCE_IRQHandler	CAN1_SCE_IRQHandler	CAN1 SCE handler

3.3 Peripheral driver update

This section describes how to port an application code based on the FWLib V2.0.3's NVIC, SysTick and CAN drivers to the StdPeriph_Lib V3.0.0

Note: In all examples listed below, the source code in italic and bold format identifies the changes between FWLib V2.0.3 and StdPeriph_Lib V3.0.0.

3.3.1 NVIC

STM32F10xxx interrupt IRQ names

The STM32F10xxx interrupt number definition names were changed according to the CMSIS naming conventions. All the `#define` of interrupt numbers should have the _IRQn postfix in their names.

Table 8 shows the different names.

Table 8. New exception naming

STM32F10xxx FWLib V2.0.3	STM32F10xxx StdPeriph_Lib V3.0.0	Description
SystemHandler_NMI	**NonMaskableInt_IRQn**	NMI Handler
SystemHandler_HardFault	-	
SystemHandler_MemoryManage	**MemoryManagement_IRQn**	Memory Management Handler
SystemHandler_BusFault	**BusFault_IRQn**	Bus Fault Handler
SystemHandler_UsageFault	**UsageFault_IRQn**	Usage Fault Handler
SystemHandler_SVCall	**SVCall_IRQn**	SVC Handler
SystemHandler_DebugMonitor	**DebugMonitor_IRQn**	Debug Monitor Handler
SystemHandler_PSV	**PendSV_IRQn**	Pend SV Handler
SystemHandler_SysTick	**SysTick_IRQn**	SysTick Handler
WWDG_**IRQChannel**	WWDG_**IRQn**	WWDG IRQ Handler
...	...	...

In the same way, in V3.0.0 the CAN peripheral name was change to CAN1. For this reason the CAN Interrupt IRQ channel names were modified, in the *startup_stm32f10x_xx.s*/*startup_stm32f10x_xx.c* files and in *stm32f10x.h*, as detailed in *Table 9* below.

Table 9. CAN1 IRQ channel name update

FWLib V2.0.3	StdPeriph_Lib V3.0.0
USB_HP_**CAN**_TX_**IRQChannel**	USB_HP_**CAN1**_TX_**IRQn**
USB_LP_**CAN**_RX0_**IRQChannel**	USB_LP_**CAN1**_RX0_**IRQn**
CAN_RX1_**IRQChannel**	**CAN1**_RX1_**IRQn**
CAN_SCE_**IRQChannel**	**CAN1**_SCE_**IRQn**

NVIC driver

The NVIC driver was removed from the STM32F10xxx standard peripheral library and the application has to use CMSIS NVIC functions. *Table 10* below shows the NVIC CMSIS functions.

Table 10. STM32F10xxx FWLib NVIC functions vs CMSIS NVIC functions

STM32F10xxx FWLib NVIC functions	CMSIS NVIC functions	Description
NVIC_PriorityGroupConfig	NVIC_SetPriorityGrouping	Sets the priority grouping in NVIC interrupt controller
NVIC_Init	NVIC_EnableIRQ	Enables interrupt in NVIC interrupt controller
	NVIC_DisableIRQ	Disables the interrupt line for specified external interrupt
	NVIC_SetPriority	Sets the priority for an interrupt
NVIC_GetIRQChannelPendingBitStatus	NVIC_GetPendingIRQ	Reads the interrupt pending bit for a device's specific interrupt source
NVIC_SetIRQChannelPendingBit	NVIC_SetPendingIRQ	Sets the pending bit for an external interrupt
NVIC_ClearIRQChannelPendingBit	NVIC_ClearPendingIRQ	Clears the pending bit for an external interrupt
NVIC_GetIRQChannelActiveBitStatus	NVIC_GetActive	Reads the active bit for an external interrupt
-	NVIC_GetPriority	Reads the priority for an interrupt
NVIC_GenerateSystemReset	NVIC_SystemReset	Initiates a system reset request

All other STM32F10xxx FWLib NVIC driver functions are not covered by the STM32F10xxx standard peripheral library.

To ease NVIC and STM32F10xxx interrupt configuration, some old NVIC driver functions are still available in a dedicated file "***misc.h/.c***".

These functions are:

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);
void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
void NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t Offset);
void NVIC_SystemLPConfig(uint8_t LowPowerMode, FunctionalState
NewState);
void SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource);
```

For interrupt configuration, the application has the choice of using the CMSIS NVIC functions or the FWLib functions defined in the "*misc.h/.c*" file. These functions have the advantage of providing an easy way of configuring peripheral interrupts without the need for an in-depth study of the NVIC specifications.

You will find below an example of NVIC interrupt configuration and enable using FWLib V2.0.3:

```
/* 1 bits for pre-emption priority 3 bits for subpriority*/
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
/* Configure and enable DMA channel6 IRQ Channel */
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel6_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 6;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

With StdPeriph_Lib V3.0.0, you only have to change the name of the DMA IRQ channel from ***DMA1_Channel6_IRQChannel*** to ***DMA1_Channel6_IRQn***.

3.3.2 SysTick

The SysTick driver was removed from the StdPeriph_Lib library and the application has to use the defined CMSIS function.

The CMSIS offers only one function for SysTick configuration, which replaces all STM32 SysTick driver functions.

```
SysTick_Config(uint32_t ticks);
```

This function configures the auto-reload counter value (LOAD), SysTick IRQ priority, resets the counter value (VAL) and enables counting and the SysTick IRQ interrupt. By default, the SysTick clock is system clock.

The example below shows a SysTick configuration using FWLib V2.0.3:

```
/* Select the HCLK Clock as SysTick clock source (72MHz) */
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
/* SysTick end of count event each 1ms with input clock equal to
72MHz (HCLK) */
SysTick_SetReload(72000);
/* Enable SysTick interrupt */
SysTick_ITConfig(ENABLE);
```

The example below shows a SysTick configuration using StdPeriph_Lib V3.0.0:

```
/* Setup SysTick Timer for 1 msec interrupts  */
if (SysTick_Config(SystemFrequency / 1000)) /* SystemFrequency is
defined in "system_stm32f10x.h" and equal to HCLK frequency */
{
/* Capture error */
while (1);
}
```

3.3.3 CAN

In the STM32F10xxx standard peripheral library, the CAN peripheral was renamed as CAN1. All occurrences related to CAN were also renamed as CAN1 in the different drivers.

The list below gives the changes related to CAN1:

- All CAN driver functions now use a new parameter: CAN_TypeDef* CANx
- In the "*stm32f10x_rcc.h/.c*" files: RCC_APB1Periph_CAN replaced by RCC_APB1Periph_CAN1
- In the "*stm32f10x_dbgmcu.h/.c*" files: DBGMCU_CAN_STOP replaced by DBGMCU_CAN1_STOP
- In the "*stm32f10x_gpio.h/.c*" files: GPIO_Remap1_CAN replaced by GPIO_Remap1_CAN1 and GPIO_Remap2_CAN replaced by GPIO_Remap2_CAN1

The example below shows a CAN configuration using FWLib V2.0.3:

```
/* CAN1 register init */
CAN_DeInit();
CAN_StructInit(&CAN_InitStructure);
/* CAN1 cell init */
CAN_InitStructure.CAN_TTCM=DISABLE;
CAN_InitStructure.CAN_ABOM=DISABLE;
CAN_InitStructure.CAN_AWUM=DISABLE;
CAN_InitStructure.CAN_NART=DISABLE;
CAN_InitStructure.CAN_RFLM=DISABLE;
CAN_InitStructure.CAN_TXFP=DISABLE;
CAN_InitStructure.CAN_Mode=CAN_Mode_LoopBack;
CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
CAN_InitStructure.CAN_BS1=CAN_BS1_8tq;
CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;
CAN_InitStructure.CAN_Prescaler=1;
CAN_Init(&CAN_InitStructure);
```

The example below shows a CAN configuration using StdPeriph_Lib V3.0.0:

```
/* CAN1 register init */
CAN_DeInit(CAN1);
CAN_StructInit(&CAN_InitStructure);
/* CAN1 cell init */
CAN_InitStructure.CAN_TTCM=DISABLE;
CAN_InitStructure.CAN_ABOM=DISABLE;
CAN_InitStructure.CAN_AWUM=DISABLE;
CAN_InitStructure.CAN_NART=DISABLE;
CAN_InitStructure.CAN_RFLM=DISABLE;
CAN_InitStructure.CAN_TXFP=DISABLE;
CAN_InitStructure.CAN_Mode=CAN_Mode_LoopBack;
```

```
CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
CAN_InitStructure.CAN_BS1=CAN_BS1_8tq;
CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;
CAN_InitStructure.CAN_Prescaler=1;
CAN_Init(CAN1, &CAN_InitStructure);
```

3.4 How to use the STM32F10xxx standard peripheral library

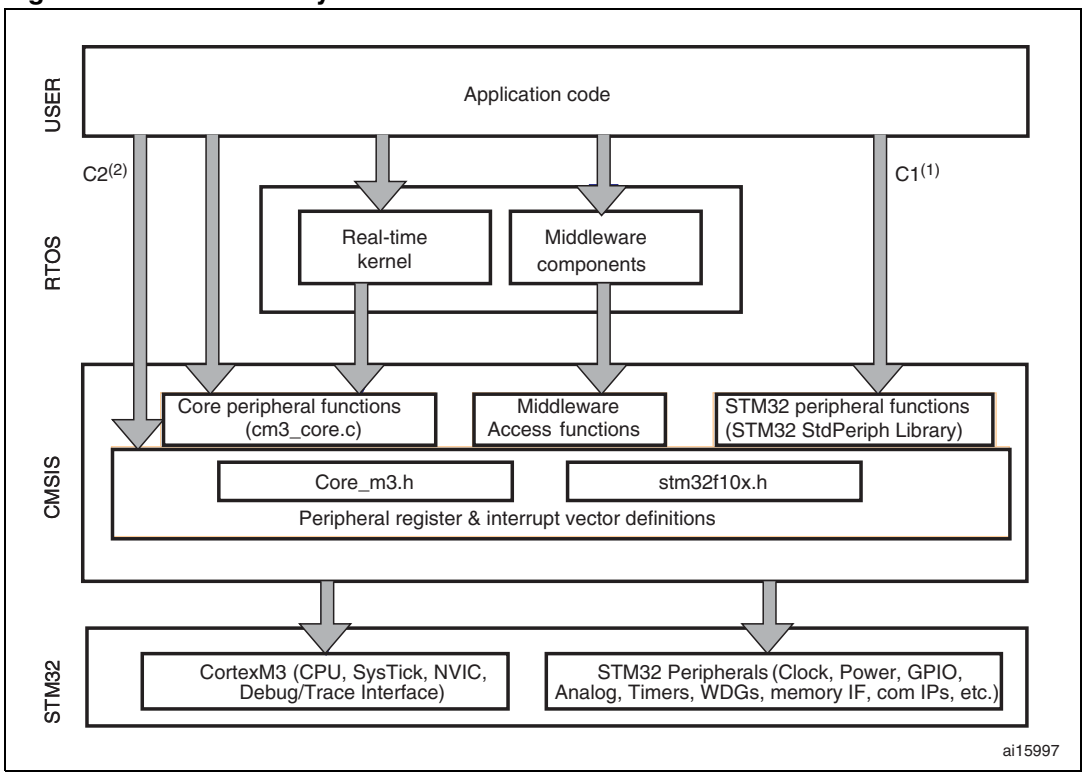
This section describes the steps to be followed to use the STM32F10xxx standard peripheral library.

- Create a project and set up all your toolchain's startup files (or use the template project provided within the library)
- Select the corresponding startup file depending on the used product, only one startup file should be selected at a time:
  - *startup_stm32f10x_ld.s/.c*
  - *startup_stm32f10x_md.s/.c*
  - *startup_stm32f10x_hd.s/.c*
- The library entry point is "*stm32f10x.h*", you should include it in the application main and configure it:
  - Select the target product family, comment/uncomment the right `define`:

```
/* #define STM32F10X_LD */ /* STM32 Low density devices */
/* #define STM32F10X_MD */ /* STM32 Medium density devices */
/* #define STM32F10X_HD */ /* STM32 High density devices */
```

- Then you have the choice of using the peripheral drivers or not
  - Case 1 (C1, see *Figure 5*): application code based on STM32 peripheral driver API
  a) Uncomment `#define USE_STDPERIPH_DRIVER` in the "*stm32f10x.h*" file.
  b) In the "*stm32f10x_conf.h*" file, select the peripherals to be used (for header file inclusion)
  c) Use peripheral driver API to built the application
  - Case2 (C2, see *Figure 5*): application code based on direct access to the STM32 peripheral registers (*stm32f10x.h*)
  a) Comment `#define USE_STDPERIPH_DRIVER` in the "*stm32f10x.h*" file.
  b) Use the peripheral register structure and bit definition file, *stm32f10x.h*, to build the application

Figure 5. STM32 library & CMSIS: structure



1. C1: Application code based on STM32 peripheral driver API.
2. C2: Application code based on direct access to STM32 peripheral registers (*stm32f10x.h* file).

4 Migration example using the automatic script

To save migration time, an automatic Perl script is provided with this application note, that makes it easier to port an old library version project to the CMSIS-based library StdPeriph_Lib V3.0.0.

This automatic script is provided inside the "*MigrationScript.zip*" zip file. This folder contains:

- *MigrationScript.exe*: automatic script binary file
- *config.ini*: migration script configuration file. It contains all needed and automatic changes that will be applied on your code
- *readme.txt*: readme file describing how to use the automatic script

4.1 How to use the automatic script

1. Download and Install the ActivePerl SW from this link: http://www.activestate.com/activeperl/
2. Copy the "*MigrationScript.exe*" and "*config.ini*" files into the parent directory of the files to be modified

Note: Please make sure that the target files are in read/write mode.

3. Launch "*MigrationScript.exe*"
4. A backup folder is created in your work folder. It contains all the old data in your directory. A trace file "*trace.log*" that summarizes all updated files is also created.

4.2 Migration steps using the automatic script

Update your project settings

1. Update your toolchain startup files
   a) Project files: device connections and Flash boot loader. These files are provided with the latest version of your toolchain that supports STM32 high-density, medium-density devices and low-density devices. For more information please refer to your toolchain documentation.
   b) Linker configuration: these files are already provided within the StdPeriph_Lib V3.0.0 package under the following directory: *STM32F10x_StdPeriph_Lib_V3.0.0\Project\Template*
   c) Vector table location files: these files are already provided within the StdPeriph_Lib V3.0.0 package under the following directory: *STM32F10x_StdPeriph_Lib_V3.0.0\\Libraries\CMSIS\Core\CM3\startup*

   Note that only one startup file should be selected
2. Replace the ***STM32F10xFWLib*** folder by ***STM32F10x_StdPeriph_Lib_V3.0.0\Librairies***
3. Remove the following files from your project workspace setting: *stm32f10x_lib.c*, *stm32f10x_nvic.c*, *stm32f10x_systick.c*, *cortexm3_macro.s* as all these files are not provided in the StdPeriph_Lib V3.0.0 package.
4. If you are using the NVIC interrupt IRQ configuration and SysTick clock source configuration, add the "*misc.c*" file to your projects.

5. Remove the *stm32f10x_vector.s/.c* files from your project workspace according to your toolchain and replace them by the corresponding product's startup file.
6. Update the project include paths:
   - remove the following path: *FWLib\library\inc*
   - add the following path: *STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\CMSIS\Core\CM3* to include the CMSIS core peripheral access layer files and STM32F10xxx CMSIS device peripheral access layer file.
   - add the following path: *STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\STM32F10x_StdPeriph_Driver\inc* if your application is based on StdPeriph_Lib drivers.

Update your application sources files

7. Replace your application's *stm32f10x_conf.h* file by the latest version provided in V3.0.0. Uncomment the corresponding lines according to the used peripherals in your application. If you are using STM32F10x IRQ configuration with NVIC you also have to uncomment `#include "misc.h"`.
8. If you want to use the assert function, in the *stm32f10x.h* file uncomment `/* #define USE_FULL_ASSERT 1 */`.
9. The *stm32f10x_it.c/.h* files, which contain the peripheral ISRs (interrupt service routines), were updated to support the new peripheral interrupt IRQ channel names. Thus, you have to replace the existing *stm32f10x_it.h* file by the new version provided in V3.0.0, after copying your code from the existing *stm32f10x_it.c* file to the new one.

Note: 1 Copy only the STM32F10xxx IRQ handlers you use. If you use one of the following IRQ handlers: USB_HP_CAN_TX_IRQHandler, USB_LP_CAN_RX0_IRQHandler, CAN_RX1_IRQHandler, CAN_SCE_IRQHandler, you have to rename them to USB_HP_CAN1_TX_IRQHandler, USB_LP_CAN1_RX0_IRQHandler, CAN1_RX1_IRQHandler and CAN1_SCE_IRQHandler, respectively.

2 Do not forget to define the corresponding IRQ handler function prototype inside the stm32f10x_it.h file.

10. Launch the automatic Perl script that makes all possible firmware changes (including types, handler names, core macro names). For more details on these changes, please refer to the "*trace.log*" file.
11. Replace the NVIC and SysTick functions used in your applications by the CMSIS functions defined in "*core_cm3.h*" according to *Table 10.*
12. Remove the following code as the Debug mode is not supported by StdPeriph_Lib:

```
#ifdef DEBUG
  debug();
#endif
```

13. Library configuration section inside the "*stm32f10x.h*" file:

- In the "*stm32f10x.h*" file, select the case that corresponds to the product that you are using: STM32F10X_LD, STM32F10X_MD or STM32F10X_HD
- Select if you want to use the STM32F10xxx standard peripheral drivers through the `USE_STDPERIPH_DRIVER` define
- Configure the different values for: HSE (external quartz), HSE startup timeout, HSI (internal high speed oscillator)

Step 14, 15 are optional

14. If your use the *stm32f10x_flash* driver routines only to set the Flash latency and prefetch buffer enable, you can remove the *stm32f10x_flash* driver and, instead, use the `SystemInit()` function from "*system_stm32f10x.h/.c*". This function initializes the embedded Flash interface. Then, initialize the PLL and update the SystemFrequency variable.
15. The *system_stm32f10x.h/.c* files provide a ready clock configuration and Flash configuration "`SystemInit()`":

- You have to select the target frequency for the system clock by uncommenting the corresponding line:

```
/* SYSCLK_FREQ_HSE */
/* SYSCLK_FREQ_20MHz */
/* SYSCLK_FREQ_36MHz */
/* SYSCLK_FREQ_48MHz */
/* SYSCLK_FREQ_56MHz */
/* SYSCLK_FREQ_72MHz */
```

- For STM32F10xxx high-density devices, you can select the external SRAM for your application:
  - Uncomment `/* #define DATA_IN_ExtSRAM */`
  - This memory will be used for all data, stack and heap. In this case you must use the corresponding linker file to your toolchain. For more details refer to *SRAM_DataMemory* example.
- You can use the `SystemInit` function by including the "*system_stm32f10x.c*" file in your project

16. Rebuild all.

Appendix A FWLib V2.0.3 to StdPeriph_Lib V3.0.0: detailed migration steps

This section describes how to migrate an application based on FWLib V2.0.3 to StdPeriph_Lib V3.0.0, without the automatic script.

To update your application code to run on StdPeriph_Lib V3.0.0 you have to follow the steps listed below:

Update your project settings

1. Update your toolchain startup files
   a) Project files: device connections and Flash boot loader, these files are provided with the latest version of your toolchain that supports STM32 high-density, medium-density devices and low-density devices. For more information please refer to your toolchain documentation.
   b) Linker configuration: these files are already provided within the StdPeriph_Lib V3.0.0 package under the following directory: *STM32F10x_StdPeriph_Lib_V3.0.0\Project\Template*
   c) Vector table location files: these files are already provided within the StdPeriph_Lib V3.0.0 package under the following directory: *STM32F10x_StdPeriph_Lib_V3.0.0\\Libraries\CMSIS\Core\CM3\startup*

   Note that only one startup file should be selected.
2. Replace the **STM32F10xxxFWLib** folder by **STM32F10xxx_StdPeriph_Lib_V3.0.0**
3. Remove the following files from your project workspace setting: *stm32f10x_lib.c*, *stm32f10x_nvic.c*, *stm32f10x_systick.c*, *cortexm3_macro.s* as all these files are not provided in the StdPeriph_Lib V3.0.0 package.
4. If you are using the NVIC interrupt IRQ configuration and Systick clock source configuration, add the "*misc.c*" file to your projects.
5. Remove the *stm32f10x_vector.s/.c* files from your project workspace according to your toolchain and replace them by the corresponding product's startup file.
6. Update the project include paths:
   – remove the following path FWLib\library\inc
   – add the following path: "*STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\CMSIS\Core\CM3"* to include the CMSIS core peripheral access layer files and STM32F10xxx CMSIS device peripheral access layer file.
   – add the following path: "*STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\STM32F10x_StdPeriph_Driver\inc*" if your application is based on StdPeriph_Lib drivers.

Update your application source files

7. Replace your application's *stm32f10x_conf.h* file by the latest version provided in V3.0.0. Uncomment the corresponding lines according to the used peripherals in your application. If you are using the STM32F10xxx IRQ configuration with NVIC, you have also to uncomment `#include "misc.h"`.
8. If you want to use the assert function, in the *stm32f10x.h* file uncomment `/* #define USE_FULL_ASSERT 1 */`.

9. The *stm32f10x_it.c/.h* files, which contain the peripheral ISRs (interrupt service routines), were updated to support the new peripheral interrupt IRQ channel names. Thus you have to replace the existing *stm32f10x_it.h* file in your application by the new version provided in V3.0.0, after copying your code from the existing *stm32f10x_it.c* file to the new one.

Note: 1 Copy only the STM32F10xxx IRQ handlers you use. If you use one of the following IRQ handlers: USB_HP_CAN_TX_IRQHandler, USB_LP_CAN_RX0_IRQHandler, CAN_RX1_IRQHandler, CAN_SCE_IRQHandler, you have to rename them to USB_HP_CAN1_TX_IRQHandler, USB_LP_CAN1_RX0_IRQHandler, CAN1_RX1_IRQHandler, CAN1_SCE_IRQHandler, respectively.

2 Do not forget to define the corresponding IRQ handler function prototypes inside the stm32f10x_it.h file.

10. Change all your Interrupt number names from PPP_IRQChannel to PPP_IRQn.
11. Replace the NVIC and SysTick functions used in your applications by the CMSIS functions defined in "*core_cm3.h*" according to *Table 10*.
12. Change your macro according to the CMSIS core peripheral access layer. The "*core_cm3.c*" file should be included in your project.
13. Replace all occurrences of "*stm32f10x_lib.h*" to "*stm32f10x.h*" in your application files. The StdPeriph_Lib entry point is the "*stm32f10x.h*" file.
14. Remove the following code as the Debug mode is not supported in StdPeriph_Lib:

```
#ifdef DEBUG
  debug();
#endif
```

15. Replace "`#ifdef DEBUG`" by "`#ifdef USE_FULL_ASSERT`" if you use the assert function in your main file.
16. Change all types to match the *<stdint.h>* file types.
17. If you use the stm32f10x_flash driver routines only to set the Flash latency and prefetch buffer enable, you can remove the "*stm32f10x_flash*" driver and, instead, use the `SystemInit()` function from "*system_stm32f10x.h/.c*". This function initializes the

embedded Flash Interface. Then initialize the PLL and update the SystemFrequency variable.

18. Library configuration section inside the "*stm32f10x.h*" file:

- In the "*stm32f10x.h*" file, select the case that correspond to the product you are using: STM32F10X_LD, STM32F10X_MD or STM32F10X_HD.
- Select if you want to use the STM32F10xxx standard peripheral drivers through the `USE_STDPERIPH_DRIVER` define.
- Configure the different values for: HSE (external quartz), HSE startup timeout, HSI (internal high speed oscillator).

19. The *system_stm32f10x.h/.c* files provide a ready clock configuration and Flash configuration "`SystemInit()`":

- Select the target frequency for the system clock by uncommenting the corresponding line:

```
/* SYSCLK_FREQ_HSE */
/* SYSCLK_FREQ_20MHz */
/* SYSCLK_FREQ_36MHz */
/* SYSCLK_FREQ_48MHz */
/* SYSCLK_FREQ_56MHz */
/* SYSCLK_FREQ_72MHz */
```

- For STM32F10xxx high-density devices, you can select the external SRAM for your application:
  - Uncomment `/* #define DATA_IN_ExtSRAM */`
  - This memory will then be used for all data, stack and heap. In this case you must use the corresponding linker file to your toolchain. For more details refer to *SRAM_DataMemory* example.
- You can use the `SystemInit` function by including the "*system_stm32f10x.c*" file in your project.

Revision history

Table 11. Document revision history

Date	Revision	Changes
06-Apr-2009	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com