AN2953

应用笔记

如何从 STM32F10xxx 固件库 V2.0.3 升级为 STM32F10xxx 标准外设库 V3.0.0

介绍

本文的目的是介绍如何把基于STM32F10xxx固件库V2.0.3(FWLib)开发的应用程序升级到STM32F10xxx标准外设库V3.0.0。本文的目的不是提供2个版本的详细信息，而是重点介绍2个版本之间的差异。

注意：文中，除非特别指明，固件库(FWLib)代表STM32F10xxx固件库V2.0.3，而标准外设库(StdPeriph_Lib)代表STM32F10xxx标准外设库V3.0.0。

术语表

小容量产品是指闪存存储器容量在16K至32K字节之间的STM32F101xx，STM32F102xx和STM32F103xx微控制器。

中容量产品是指闪存存储器容量在64K至128K字节之间的STM32F101xx，STM32F102xx和STM32F103xx微控制器。

大容量产品是指闪存存储器容量在256K至512K字节之间的STM32F101xx和STM32F103xx微控制器。

译注：

本译文的英文版下载地址为：

http://www.st.com/stonline/products/literature/an/15531.pdf

STM32F10xxx标准外设库V3.0.0下载地址为：

http://www.st.com/stonline/products/support/micro/files/stm32f10x_stdperiph_lib_v3.0.0.zip

自动升级脚本下载地址为：

http://www.st.com/stonline/products/support/micro/files/an2953.zip

目录

1 为什么把STM32F10xxx固件库(FWLib)V2.0.3升级为标准外设库(StdPeriph_Lib)V3.0.0

STM32F10xxx固件库(FWLib)V2.0.3是一个完整的固件包，它适用于STM32F10xxx小容量，中容量和大容量产品。固件库由程序，数据结构和宏组成，覆盖了所有外设的特征，还包括了全部标准外设的驱动和一系列示例程序。

STM32F10xxx标准外设库(StdPeriph_Lib)V3.0.0由固件库(FWLib)V2.0.3升级而来：

- 它使库与Cortex™微控制器软件接口标准(CMSIS)兼容
- 改进了库包的体系结构
- 源代码符合Doxygen格式
- 升级不影响STM32外设驱动的API(应用编程接口)

注意：标准外设库(StdPeriph_Lib)V3.0.0只对STM32F10xxx CAN驱动进行了升级，目的是支持即将面世的STM32F10xxx连接型产品(带双CAN)。

要升级到STM32F10xxx标准外设库V3.0.0，用户只需要更新：

- 与工具链相关的文件
- 项目(project)设置
- 库文件的位置
- 用户无需改变或者更新应用程序的代码

下文详细描述了标准外设库(StdPeriph_Lib)的所有更新细节。

1.1 兼容ARM® Cortex-M3™微控制器软件接口标准(CMSIS)

CMSIS可以解决用户在基于Cortex-M0/Cortex-M1或者Cortex-M3内核的微控制器上进行软件开发时可能遇到的种种问题。CMSIS还可以扩展，应用在将来的Cortex-M系列处理器内核上(标准称为Cortex-Mx)。CMSIS是ARM公司与多家不同的芯片和软件供应商一起紧密合作定义的，提供了内核与外设、实时操作系统和中间设备之间的通用接口。可以访问网站www.onarm.com获取更多细节。

1.1.1 CMSIS描述

CMSIS可以分为多个软件层次，ARM提供了下列部分，可用于多种编译器：

- 内核设备访问层：包含了用来访问内核的寄存器设备的名称定义，地址定义和助手函数。同时也为RTOS(实时操作系统)定义了独立于微控制器的接口，该接口包括调试通道定义。
- 中间设备访问层：为软件提供了访问外设的通用方法。芯片供应商应当修改中间设备访问层，以适应中间设备组件用到的微控制器上的外设。目前中间设备访问层仍处于开发过程中，本文不做详述。

芯片供应商扩展下列软件层：

- 微控制器外设访问层：提供片上所有外设的定义。
- 外设的访问函数(可选)：为外设提供额外的助手函数。

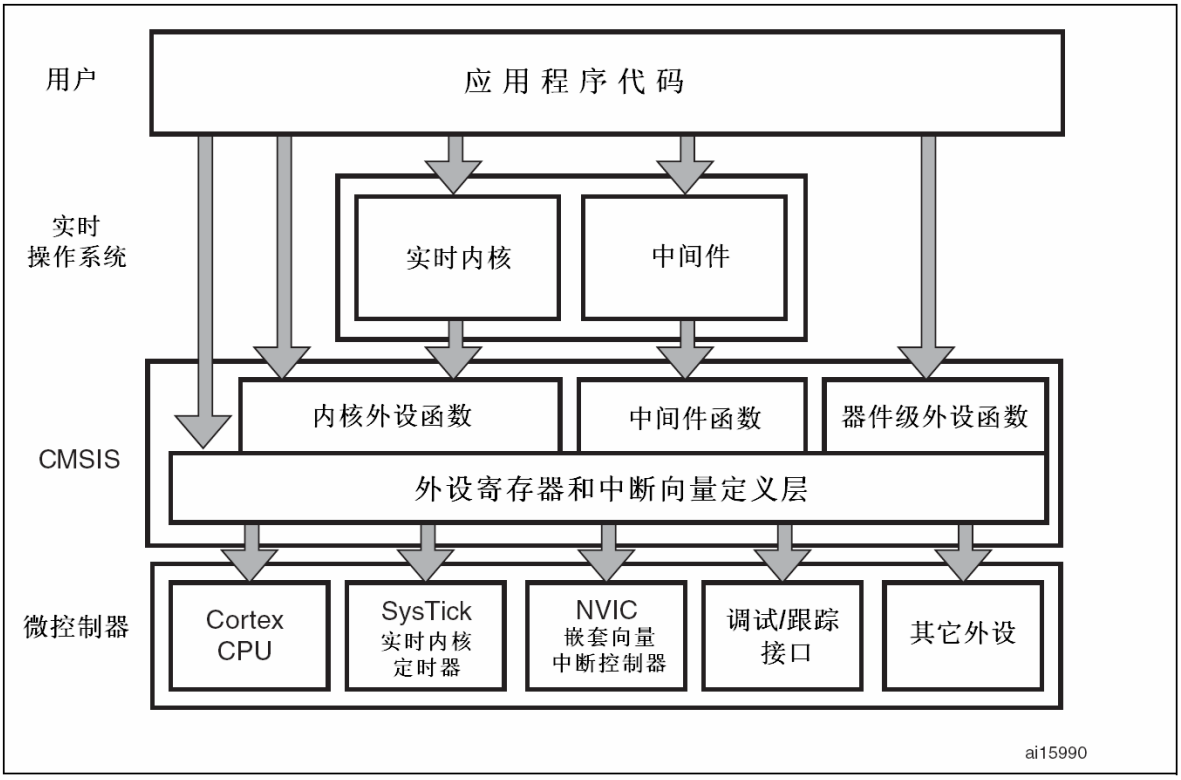
CMSIS为Cortex-Mx微控制器系统定义了：

- 访问外设寄存器的通用方法和定义异常向量的通用方法。
- 内核设备的寄存器名称和内核异常向量的名称
- 独立于微控制器的RTOS接口，带调试通道
- 中间设备组件接口(TCP/IP协议栈，闪存文件系统)

1.1.2 CMSIS结构

图1展示了基于CMSIS的应用程序的基本结构。

图1　　CMSIS层结构



CMSIS-外设访问层的文件

独立于编译器的文件：

- Cortex-M3内核及其设备文件(core_cm3.h + core_cm3.c)
  - 访问 Cortex-M3 内核及其设备：NVIC，SysTick 等
  - 访问 Cortex-M3 的 CPU 寄存器和内核外设的函数
- 微控制器专用头文件(device.h)
  - 指定中断号码(与启动文件一致)
  - 外设寄存器定义(寄存器的基地址和布局)
  - 控制微控制器其他特有的功能的函数(可选)
- 微控制器专用系统文件(system_device.c)
  - 函数 SystemInit，用来初始化微控制器
  - 函数 Sysem_ExtMemCtl，用来配置外部存储器控制器。它位于文件 startup_stm32f10x_xx.s /.c，在跳转到 main 前调用
  - SystemFrequncy，该值代表系统时钟频率
  - 微控制器的其他功能(可选)

编译器供应商+微控制器专用启动文件

- 编译器启动代码(汇编或者C)(startup_device.s)
  - 微控制器专用的中断处理程序列表(与头文件一致)
  - 弱定义(Weak)的中断处理程序默认函数(可以被用户代码覆盖)

注意：　关键字weak指示编译器弱导出符号。该关键字可以用于函数和变量的声明，以及函数的定义。使用__weak 定义的函数将弱导出其符号。除非将相同名称的非弱定义函数链接到相同映像上，否则弱定义函数的行为与正常定义的函数类似。 如果非弱定义函数和弱定义函数位于相同

映像中，则会将对弱定义函数的所有调用解析为对非弱函数的调用。如果有多个可用的弱定义，链接器将选择其中的一个弱定义供所有调用使用。

1.1.3 STM32 固件库V2.0.3 和CMSISV1.10 对比

CMSIS对STM32F10xxx固件库(FWLib)的某些功能采用了不同的实现方法。下面列出主要的区别：

- 使用标准C类型，文件<stdint.h>
- 对每一个Cortex-M3异常和STM32的IRQ，有：
  - 异常服务程序带后缀_Handler，中断服务程序带后缀_IRQHandler。
  - 弱定义(Weak)的默认异常/中断服务程序，包含一个无限循环
  - 带_IRQn 后缀的中断号码“#define”
- 启动文件更名为”startup_stm32f10x_xx.s/.c”，其中xx可以是hd，md或者ld，分别对应大容量，中容量，小容量产品。
- 只提供精简的NVIC和SysTick函数，其他一些常用函数作为一个新的驱动加入STM32F10xxx标准外设库，文件命名为misc.h/.c。
- 某些宏的名字与STM32F10xxx固件库V2.0.3中的相同功能宏不同(见表1)

表1　STM32F10xxx固件库V2.0.3与CMSIS宏对比[1]

STM32宏	CMSIS宏	STM32宏	CMSIS宏
-	__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. 加粗的字体表示这些宏有变化。灰色的充填色表示这些改动影响到固件库(FWLib)的驱动或者例程。

1.2 STM32F10xxx标准外设库：Doxygen格式

STM32F10xxx标准外设库的源代码采用了新的格式，所有源文件都按照doxygen格式书写，用这种书写格式的代码能够很便利地生成更加规范且内在关联性更强的文档。

ST将从意法半导体微控制器网站(www.st.com/mcu)撤下现有的STM32F10xxx固件库用户手册UM0427，而由Doxygen生成的CHM文件取而代之。该CHM文件完整地描述了STM32F10xxx标准外设库的全部组件。

Doxygen例程：

```
/**
* @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.
*/
```

其中：

- @brief：一行简洁的函数功能描述
- @param：函数的参数详解
- @retval：函数的返回值详细信息

可以参阅"stm32f10x_stdperiph_lib_um.chm"获取更多相关细节。

1.3 STM32F10xxx标准外设库体系结构

ST改进了STM32F10xxx标准外设库的体系结构并支持CMSIS层。

根据应用程序的需要，可以采取2种方法使用标准外设库(StdPeriph_Lib)：

- 使用外设驱动：这时应用程序开发基于外设驱动的API(应用编程接口)。用户只需要配置文件"stm32f10x_conf.h"，并使用相应的文件"stm32f10x_ppp.h/.c"即可。
- 不使用外设驱动：这时应用程序开发基于外设的寄存器结构和位定义文件。

标准外设库(StdPeriph_Lib)支持STM32F10xxx系列全部成员：大容量，中容量和小容量产品。根据使用的STM32产品具体型号，用户可以通过文件"stm32f10x.h"中的预处理define来配置标准外设库(StdPeriph_Lib)，一个define对应一个产品系列。下面列出支持的产品系列

- STM32F10x_LD：STM32小容量产品
- STM32F10x_MD：STM32中容量产品
- STM32F10x_HD：STM32大容量产品

这些define的作用范围是：

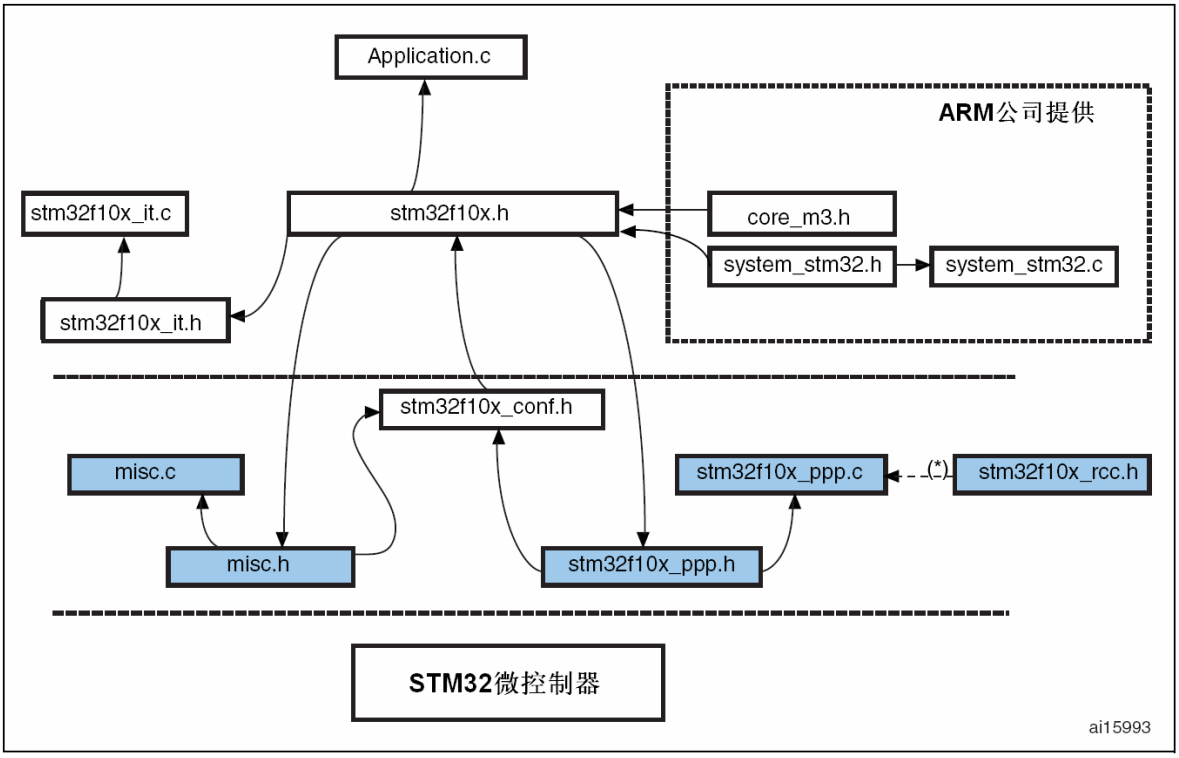
- 文件"stm3210f.h"中的中断IRQ定义
- 启动文件中的向量表，小容量，中容量，大容量产品各有一个启动文件
- 外设存储器映像和寄存器物理地址
- 产品设置：外部晶振(HSE)的值等
- 系统配置函数
- 非STM32全系列兼容或不同型号产品间有差异的功能特征

注意：这些define不会影响外设的驱动，驱动支持STM32全系列全部外设的功能特征。

1.4 STM32F10xxx标准外设库体系结构：文件包含关系

图2展示了STM32F10xxx的文件包含关系

图2　　STM32F10xxx标准外设库体系结构



1.5　STM32F10xxx固件库(FWLib)V2.0.3存档

ST仍然会在STM32™网站(http://www.st.com/mcu/familiesdocs-110.html)上保留STM32F10xxx固件库(FWLib)V2.0.3及其全部相关固件。所有的文件包含在一个名称为"STM32F10x_FW_Archive.zip "的zip压缩包中，可在网页的"Firmware"栏目下载。该栏目可以直接通过下列地址访问：http://www.st.com/mcu/familiesdocs-110.html#Firmware

除了这个归档zip文件，ST还提供一个名为" STM32F10xFWLib_V2.0.3_Patch1.zip"的补丁压缩包，可以修复固件库V2.0.3的全部局限性。

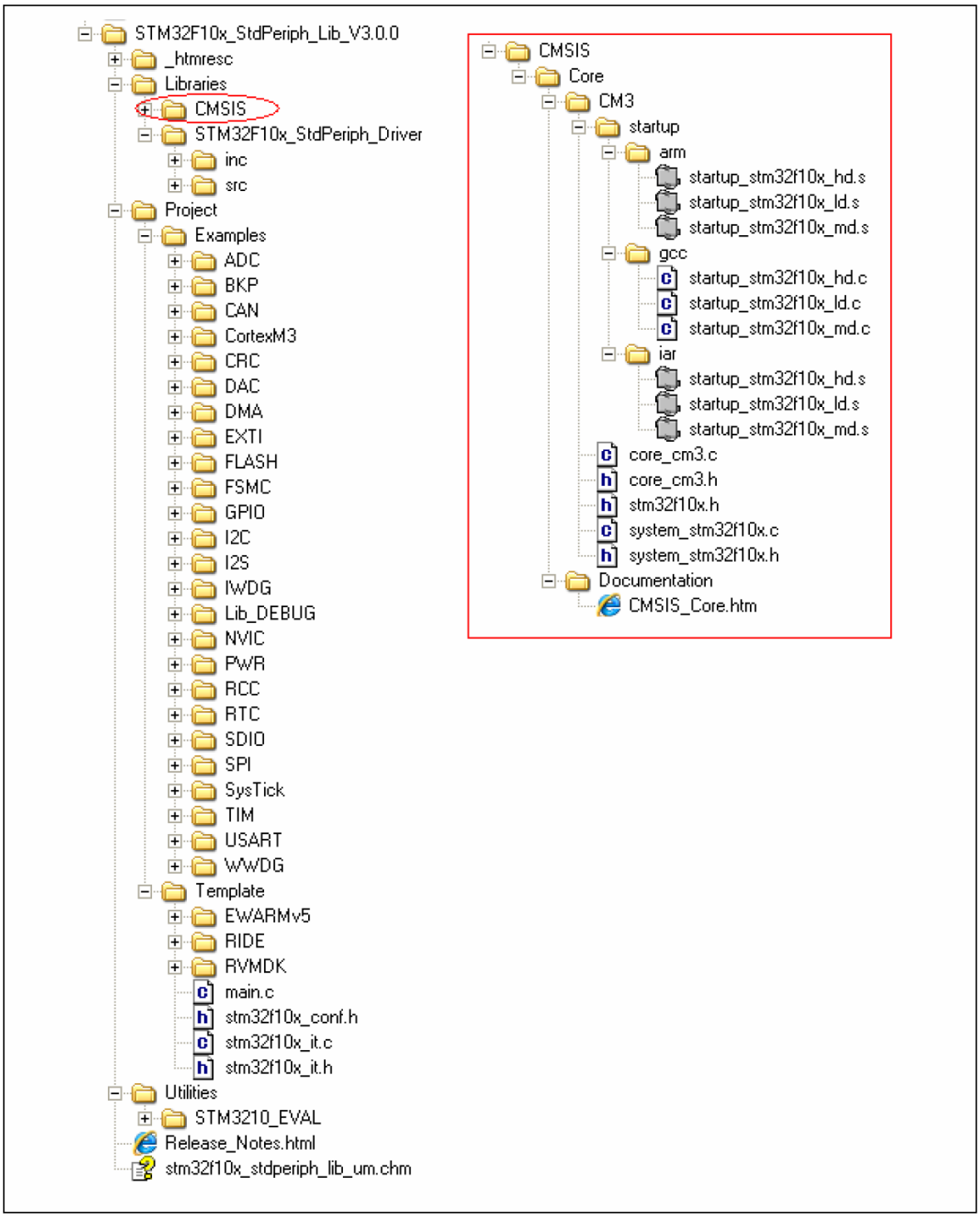
2 STM32F10xxx标准外设库包

为了使STM32F10xxx标准外设库的使用更加灵活，也为了改进了库的结构，ST更新了STM32F10xxx固件库包，添加专用子文件夹来包含CMSIS及其外设访问层相关文件。

STM32F10xxx标准外设库包重命名为STM32F10x_StdPeriph_Lib_VX.Y.Z。

新包的体系结构如图3所示。

图3 STM32F10xxx标准外设库包结构



新包描述

描述了所有在STM32F10xxx标准外设库包里的新文件夹

表2　STM32F10xxx标准外设库包文件夹描述

<table>
<tr><td colspan="9">STM32F10xxx_StdPeriph_Lib</td></tr>
<tr><td>Utilities</td><td colspan="4">Project</td><td colspan="3">Libraries</td><td>_htmresc</td></tr>
<tr><td>Template</td><td colspan="3">Template</td><td>Examples</td><td colspan="2">STM32F10x_StdPeriph_Driver</td><td>CMSIS</td><td rowspan="3">本文件夹包含了所有的html页面资源</td></tr>
<tr><td>STM3210-EVAL</td><td>RVMDK</td><td>RIDE</td><td>EWARMv5</td><td rowspan="2">标准外设库驱动的完整例程</td><td>src</td><td>inc</td><td rowspan="2">见表3</td></tr>
<tr><td>本文件夹包含了用于STM3210B-EVAL 和STM3210E-EVAL 评估板的专用驱动</td><td>KEIL RVMDK的项目模板示例</td><td>Raisonance RIDE的项目模板示例</td><td>IAR EWARMv5的项目模板示例</td><td>标准外设库驱动源文件</td><td>标准外设库驱动头文件</td></tr>
</table>

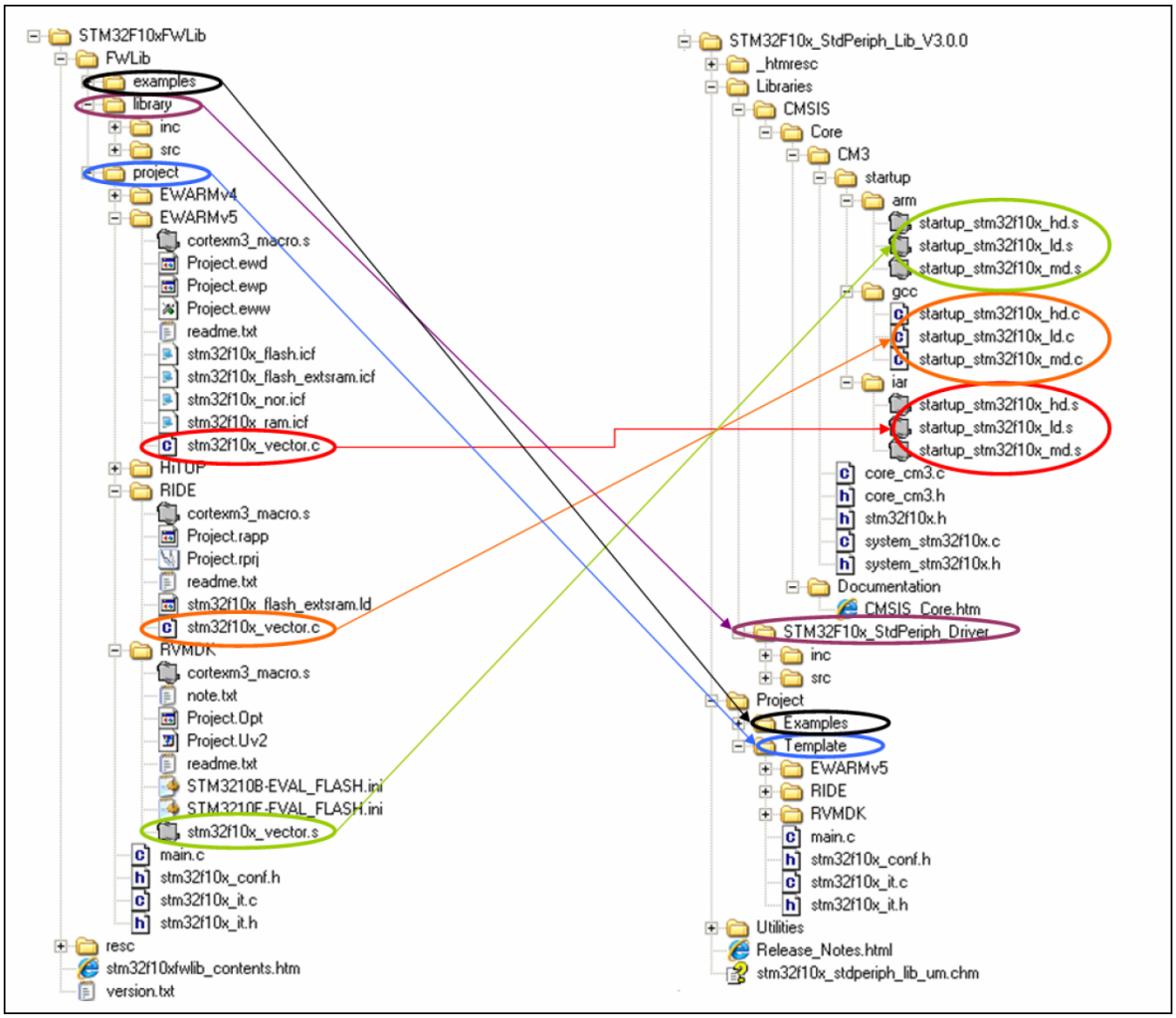
表3是CMSIS文件夹的结构

表3　CMSIS文件夹结构

<table>
<tr><td colspan="5">CMSIS</td></tr>
<tr><td colspan="5">Core</td></tr>
<tr><td rowspan="2">Documentation</td><td colspan="3">CM3</td><td rowspan="4">本文件夹包含STMF10xxx CMSIS文件：微控制器外设访问层和内核设备访问层：
– core_cm3.h ：CMSIS 的Cortex-M3内核设备访问层头文件
– core_cm3.c ：CMSIS 的Cortex-M3内核设备访问层源文件
– stm32f10x.h ：CMSIS 的Cortex-M3 STM32f10xxx微控制器外设访问层头文件
– system_stm32f10x.h：CMSIS的Cortex-M3 STM32f10xxx微控制器外设访问层头文件
– system_stm32f10x.c：CMSIS的Cortex-M3 STM32f10xxx微控制器外设访问层源文件</td></tr>
<tr><td colspan="3">Startup</td></tr>
<tr><td rowspan="2">CMSIS文档</td><td>iar</td><td>gcc</td><td>arm</td></tr>
<tr><td>IAR编译器启动文件：
– startup_stm32f10x_hd.s: 大容量产品启动文件
– startup_stm32f10x_md.s: 中容量产品启动文件
– startup_stm32f10x_ld.s: 小容量产品启动文件</td><td>GCC编译器启动文件：
– startup_stm32f10x_hd.c: 大容量产品启动文件
– startup_stm32f10x_md.c: 中容量产品启动文件
– startup_stm32f10x_ld.c: 小容量产品启动文件</td><td>ARM编译器启动文件：
– startup_stm32f10x_hd.s: 大容量产品启动文件
– startup_stm32f10x_md.s: 中容量产品启动文件
– startup_stm32f10x_ld.s: 小容量产品启动文件</td></tr>
</table>

原STM32F10xxx固件库包与新STM32F10xxx标准外设库包对比

图4　　新函数包与原函数包对比



3 STM32F10xxx标准外设库变动列表

3.1 STM32F10xxx标准外设库文件

3.1.1 库的内核文件

- 文件stm32f10x.h更名为stm32f10x.h。它包含
  - STM32 中断 IRQ 列表
  - Cortex-M3 内核的特别选项
  - STM32 外设存储器映像和寄存器物理地址定义
  - 专用 define “__STM32F10X_STDPERIPH_vERSION”表示 STM32F10xxx 标准外设库的版本
  - 配置信息：
  - a) 应用程序需要选择运行它的 STM32 产品具体型号，每个产品只要一个 define
  - b) 应用程序需要选择是否使用外设驱动
- 移除了Debug模式，因此在调试时不再可以通过一个监视窗口观察外设寄存器，但是可以利用特定工具链的调试功能监视外设的寄存器。因此，移除/更新下列文件：
  - main.c：把#ifdef DEBUG 替换为#ifdef USE_FULL_ASSERT
  - 移除文件 stm32f10x_lib.h，它的内容合并到文件 stm32f10x_conf.h
  - 移除文件 stm32f10x_lib.c
  - 更新文件 stm32f10x_conf.h：
  - a) 移除“#define DEBUG 1”，为完全断言函数增加专用 define “#define USE_FULL_ASSERT 1”
  - b) 不再需要枚举式地 define 用到的外设(如#define _USART, #define _USART1, #define _USART2)
  - c) 用户需要去掉相关行的注释符号来使用相应外设驱动，例如，想使用 SPI 驱动，去掉#include “stm32f10x_spi.h”这行的注释符号即可
  - 用文件<stdint.h>替换文件 stm32f10x_type.h，出于兼容旧版本库的目的，在文件 stm32f10x.h 中保留了原有的类型定义。在文件 stm32f10x.h 中添加一些标准外设库专用的类型定义(bool, FlagStatus, ITStatus, FunctionalState, ErrorStatus)。
  - 移除文件 cortexm3_macro.h 和 cortexm3_macro.s，因为 CMSIS 文件覆盖了它们的内容。

注意：标准外设库提供了一个名为"Lib_DEBUG"的特别示例，示范了如何为选中的外设定义DEBUG功能。

3.1.2 库的外设驱动

- 移除NVIC和SysTick驱动，它们的功能已由CMSIS内核设备层覆盖，另添加了5个常用函数作为新的驱动(misc.h / misc.c)。
  - void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup); 用来简化 Cortex-M3 优先级位设置
  - void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct); 用来简化 NVIC IRQ 设置
  - void NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset); 用来从内部 SRAM 启动并把中断向量表重新映射到存储器不同地址
  - void NVIC_SystemLPConfig(u8 LowPowerMode, FunctionalState NewState);
  - void SysTick_CLKSourceConfig(u32 SysTick_CLKSource);
- 更新CAN驱动：在所有的CAN驱动函数中，添加新的参数CAN_TypeDef* CANx.
- 其他驱动没有变化

3.1.3 库的用户和工具链专用文件

- 启动文件stm32f10x_vector.s / stm32f10x_vector.c更名为startup_stm3210f10x_xx.s / startup_stm3210f10x_xx.c，每个启动文件对应一系列产品：
  - startup_stm3210f10x_ld.s，STM32 小容量产品
  - startup_stm3210f10x_md.s，STM32 中容量产品
  - startup_stm3210f10x_hd.s，STM32 大容量产品
- stm32f10x_it.h/ stm32f10x_it.c：在这两个文件中移除了全部STM32 IRQ服务程序，只保留了Cortex-M3的异常处理程序。这些IRQ服务程序已经弱定义(Weak)在启动文件(startup_stm32f10x_xx.s/.c)中，用户需要在文件stm32f10x_it.h/ stm32f10x_it.c中手动添加外设的中断服务程序(ISR)来替换掉启动文件中的默认中断服务程序。
- 根据CMSIS重新命名Cortex-M3异常
- main.c：移除下面的代码：

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

3.1.4 库的例程

- 移除NVIC，CM3_LPModes和Systm_Handlers的例程，其他例程(VectorTable_Relocation, DMA_WFIMode, IRQ_Channel和Priority)保留。
- 更新PWR和Cortex-M3的例程，更新了其中宏的名称
- 其他例程没有变化

3.2 代码的书写规则和惯例

3.2.1 数据类型和IO类型限定词

Cortex-Mx HAL使用标准ANSI C头文件<stdint.h>定义的标准类型。特别用类型限定词IO来访问外设的变量。类型限定词IO还用于外设寄存器的调式信息的自动生成。

表4 CMSIS IO类型限定词

IO类型限定词	#define	描述
__I	volatile const	只读访问
__O	volatile	只写访问
__IO	volatile	读和写访问

ST从库包中移除了文件"stm32f10x_type.h"，新的库使用CMSIS和<stdint.h>定义的数据类型。表5展示了STM32F10xxx和<stdint.h>之间数据类型的一一对应关系。

表5 STM32F10xxx固件库V2.0.3与CMSIS数据类型对比

STM32F10xxx固件库类型	CMSIS类型	描述
s32	int32_t	有符号32位数据
s16	int16_t	有符号16位数据
s8	int8_t	有符号8位数据
sc32	const int32_t	只读有符号32位数据
sc16	const int16_t	只读有符号16位数据
sc8	const int8_t	只读有符号8位数据
vs32	__IO int32_t	易挥发读写访问有符号32位数据
vs16	__IO int16_t	易挥发读写访问有符号16位数据
vs8	__IO int8_t	易挥发读写访问有符号8位数据

vsc32	__I int32_t	易挥发只读有符号32位数据
vsc16	__I int16_t	易挥发只读有符号16位数据
vsc8	__I int8_t	易挥发只读有符号8位数据
u32	uint32_t	无符号32位数据
u16	uint16_t	无符号16位数据
u8	uint8_t	无符号8位数据
uc32	const uint32_t	只读无符号32位数据
uc16	const uint16_t	只读无符号16位数据
uc8	const uint8_t	只读无符号8位数据
vu32	__IO uint32_t	易挥发读写访问无符号32位数据
vu16	__IO uint16_t	易挥发读写访问无符号16位数据
vu8	__IO uint8_t	易挥发读写访问无符号8位数据
vuc32	__I uint32_t	易挥发只读无符号32位数据
vuc16	__I uint16_t	易挥发只读无符号16位数据
vuc8	__I uint8_t	易挥发只读无符号8位数据

注意： *1 出于兼容旧版本的目的，文件"stm32f10x.h"中仍然定义了STM32F10xxx固件库原有数据类型。*

2 文件"stm32f10x.h"中也定义了STM32F10xxx固件库专用类型，它们是：

```
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 异常的命名

表6展示了修改后符合CMSIS命名的异常处理程序名称。

表6 STM32F10xxx固件库V2.0.3与CMSIS异常名称对比

STM32F10xxx异常	CMSIS	描述
NMIException	NMI_Handler	NMI异常
HardFaultException	HardFault_Handler	硬件错误异常
MemManageException	MemManage_Handler	存储器管理错误异常
BusFaultException	BusFault_Handler	总线错误异常
UsageFaultException	UsageFault_Handler	使用错误异常
SVCHandler	SVC_Handler	SVCall异常
DebugMonitor	DebugMon_Handler	调试监控器异常
PendSVC	PendSV_Handler	PendSVC异常
SysTickHandler	SysTick_Handler	SysTick处理程序

表7展示了将名称中CAN改为CAN1的异常处理程序

表7 STM32F10xxx CAN1异常重命名

STM32F10xxx异常	CMSIS	描述
USB_HP_CAN_TX_IRQHandler	USB_HP_CAN1_TX_IRQHandler	USB高优先级或CAN1 TX中断处理程序
USB_HP_CAN_RX0_IRQHandler	USB_HP_CAN1_RX0_IRQHandler	USB低优先级或CAN1 RX0中断处理程序

CAN_RX1_IRQHandler	CAN1_RX1_IRQHandler	CAN1 RX1中断处理程序
CAN_SCE_IRQHandler	CAN1_SCE_IRQHandler	CAN1 SCE中断处理程序

3.3 外设驱动更新

下文描述了如何把基于固件库V2.0.3的NVIC，SysTick和CAN驱动开发的应用程序，升级到标准外设库V3.0.0。

注意：在下文列举的所有例程中，字体为倾斜加粗的，体现了固件库V2.0.3和标准外设库V3.0.0之间的差别。

3.3.1 NVIC

STM32F10xxx中断IRQ命名

按照CMSIS的规范，修改了STM32F10xxx中断号码定义命名。所有中断号码的#define在它们的名称中都添加了后缀_IRQn。

表8展示了名称的变化

表8 新的异常名称

STM32F10xxx固件库V2.0.3	STM32F10xxx标准外设库V3.0.0	描述
SystemHandler_NMI	**NonMaskableInt_IRQn**	NMI处理程序
SystemHandler_HardFault	**-**	
SystemHandler_MemoryManage	**MemoryManagement_IRQn**	存储器管理处理程序
SystemHandler_BusFault	**BusFault_IRQn**	总线错误处理程序
SystemHandler_UsageFault	**UsageFault_IRQn**	使用错误处理程序
SystemHandler_SVCall	**SVCall_IRQn**	SVC处理程序
SystemHandler_DebugMonitor	**DebugMonitor_IRQn**	调试监控器处理程序
SystemHandler_PSV	**PendSV_IRQn**	Pend SV处理程序
SystemHandler_Systick	**SysTick_IRQn**	SysTick处理程序
WWDG_IRQChannel	**WWDG_IRQn**	WWDG IRQ处理程序
…	…	…

同样，在标准外设库V3.0.0中，外设CAN的名称改为CAN1。这样，如表9所示，在文件startup_stm32f10x_xx.s / startup_stm32f10x_xx.c和stm32f10x.h中，CAN中断IRQ通道名称也相应改变。

表9 CAN1 IRQ通道名称更新

固件库V2.0.3	标准外设库V3.0.0
USB_HP_**CAN**_TX_**IRQChannel**	USB_HP_**CAN1**_TX_**IRQn**
USB_HP_**CAN**_RX0_ **IRQChannel**	USB_HP_**CAN1**_RX0_ **IRQn**
CAN_RX1_ **IRQChannel**	**CAN1**_RX1_ **IRQn**
CAN_SCE_ **IRQChannel**	**CAN1**_SCE_ **IRQn**

NVIC驱动

从STM32F10xxx标准外设库中移除了NVIC驱动，因此应用程序应当调用CMSIS的NVIC函数。表10展示了CMSIS的函数。

表10 STM32F10xxx固件库NVIC函数与CMSIS NVIC函数对比

STM32F10xxx固件库NVIC函数	CMSIS NVIC函数	描述
NVIC_PriorityGroupConfig	NVIC_SetPriorityGrouping	在NVIC中断控制器中设置组优先级
NVIC_Init	NVIC_EnableIRQ	在NVIC中断控制器中使能中断

	NVIC_DisableIRQ	失能指定的外部中断线路
	NVIC_SetPriority	为某中断设置优先级
NVIC_GetIRQChannelPendingBitStatus	NVIC_GetPendingIRQ	读取指定的微控制器中断的待处理标志位
NVIC_SetIRQChannelPendingBit	NVIC_SetPendingIRQ	为某外部中断设置待处理标志位
NVIC_ClearIRQChannelPendingBit	NVIC_ClearPendingIRQ	为某外部中断清除待处理标志位
NVIC_GetIRQChannelActiveBitStatus	NVIC_GetActive	读取某中断的活动标志位
-	NVIC_GetPriority	读取某中断的优先级
NVIC_GenerateSystemReset	NVIC_SystemReset	初始化系统复位请求

STM32F10xxx标准外设库不覆盖其他的STM32F10xxx固件库NVIC函数。

为了简化NVIC和STM32中断设置，在文件"misc.h/.c"中保留了原有NVIC驱动的一些函数。

它们是：

```
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);
```

在应用程序开发中进行中断设置时，用户可以选用CMSIS的NVIC函数，或者选用在文件"misc.h/.c"中原有固件库函数。这些函数的优点是提供了相对简单的中断配置方法，使得用户无需深入研究NVIC规范。

下面是一段利用固件库V2.0.3原有函数进行NVIC中断设置的例程：

```
/* 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);
```

在标准外设库3.0.0中，用户只需要把DMA IRQ通道的名称从***DMA1_Channel6_IRQchannel***改为***DMA1_Channel6_IRQn***。

3.3.2 SysTick

在标准外设库中移除了SysTick的驱动，因此用户必须调用CMSIS定义的函数。

CMSIS只提供了一个SysTick设置的函数，替代了STM32原有SysTick驱动的全部函数。

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

该函数设置了自动重载入计数器(LOAD)的值，SysTick IRQ的优先级，复位了计数器(VAL)的值，开始计数并打开SysTick IRQ中断。SysTick时钟默认使用系统时钟。

下面的例程为使用固件库V2.0.3进行SysTick设置：

```
/* 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);
```

下面的例程为使用标准外设库V3.0.0进行SysTick设置：

```
/* 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

在STM32F10xxx标准外设库中，外设CAN重新命名为CAN1。因此所有名称中带CAN的，其中CAN也相应改为CAN1。

下面列出所有CAN1相关的改动：

- 所有CAN驱动函数使用新的参数：CAN_TypeDef* CANx
- 在文件"stm3210x_rcc.h/.c"中，RCC_APB1Periph_CAN改为RCC_APB1Periph_CAN1
- 在文件"stm32f10x_dbgmcu.h/.c"中，DBGMCU_CAN_STOP改为DBGMCU_CAN1_STOP
- 在文件"stm32f10x_gpio.h/.c"中，GPIO_Remap1_CAN改为GPIO_Remap1_CAN1，GPIO_Remap2_CAN改为GPIO_Remap2_CAN1

下面的例程为使用固件库V2.0.3进行CAN设置：

```
/* 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);
```

下面的例程为使用标准外设库V3.0.0进行CAN设置：

```
/* 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 如何使用STM32F10xxx标准外设库

本章节描述了使用STM32F10xxx标准外设库应当遵循的步骤：

- 新建一个项目并设置工具链对应的启动文件(或者使用库提供的项目模板)
- 按照使用的产品具体型号选择启动文件，只要选中下列文件之一即可：
  - startup_stm32f10x_hd.s/.c

- startup_stm32f10x_md.s/.c
- startup_stm32f10x_ld.s/.c

● 库的入口是文件"stm32f10x.h"，用户应当在应用程序文件中包含该头文件，并对其进行设置：

- 根据使用产品所属的系列，正确地注释/去掉注释相应的 define：

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

● 然后，用户可以选择使用外设驱动与否：

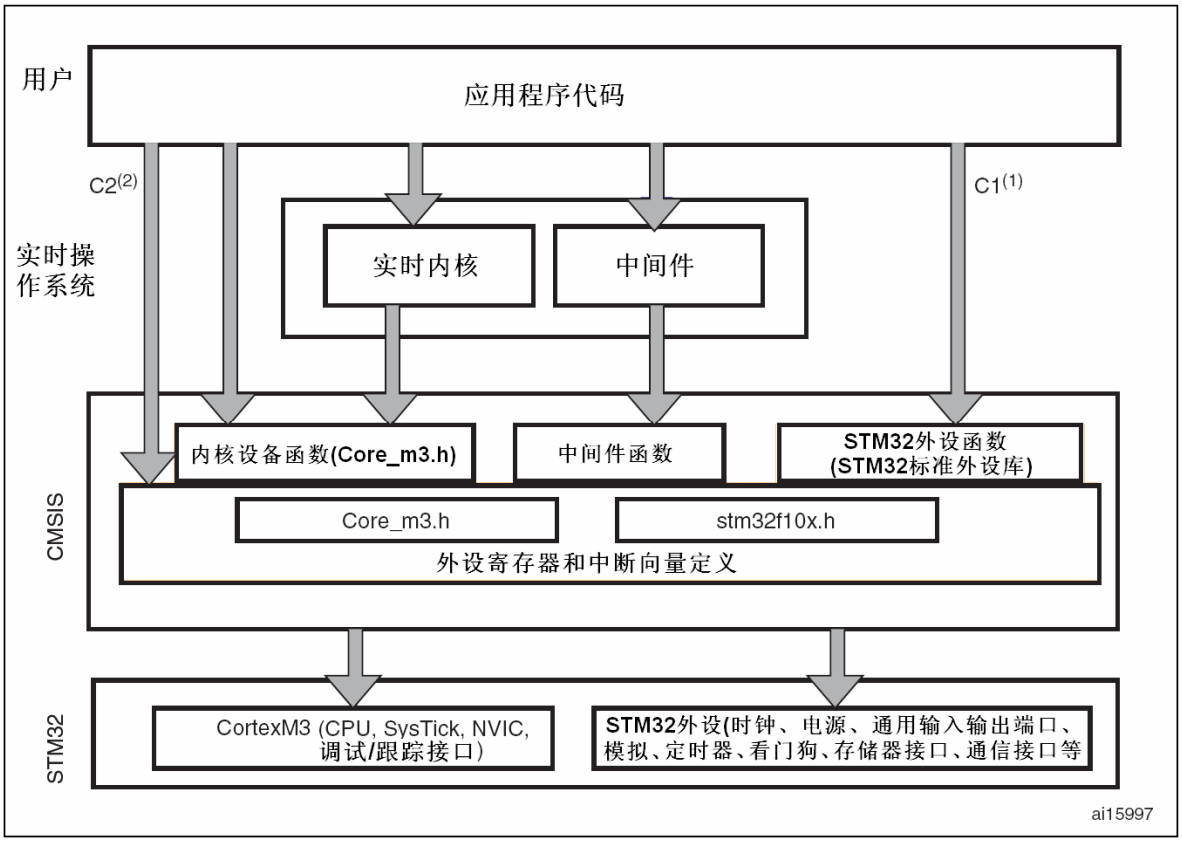
- 情况 1(C1，见图 5)：用户应用程序开发基于STM32 外设驱动API

a) 去掉文件"stm32f10x.h"中#define USE_STDPERIPHE_DRIVER 的注释符号。
b) 在文件"stm32f10x_conf.h"中，选择要用的外设(去掉包含相应头文件那行代码的注释符号)
c) 利用外设驱动 API 开发应用程序

- 情况 2(C2，见图 5)：用户应用程序开发基于直接访问STM32 外设寄存器 (stm32f10x.h)

a) 注释文件"stm32f10x.h"中的#define USE_STDPERIPHE_DRIVER。
b) 利用外设寄存器结构和位定义文件 stm32f10x.h 来开发应用程序

图5　　STM32库和CMSIS：结构



1. C1：用户应用程序开发基于STM32外设驱动API
2. C2：用户应用程序开发基于直接访问STM32外设寄存器 (stm32f10x.h)

4 使用自动脚本的升级示例

本文同时提供了一个自动Perl脚本，可以使基于旧版本固件库的应用程序升级到基于CMSIS的标准固件库V3.0.0的过程更为简便，大大节省了升级的时间。

该自动脚本包含在名为"MigrationScript.zip"的zip文件中。在文件夹下有：

- MigrationScript.exe：自动脚本二进制文件
- config.ini：升级脚本配置文件。该文件包含了将对用户代码自动进行的所有需要的变动。
- readme.txt：readme文件，描述了如何使用自动脚本

4.1 如何使用自动脚本

1. 从下面链接下载并安装ActivePerl软件：
   http://www.activestate.com/activeperl
2. 把文件"MigrationScript.exe"和"config.ini"复制入需要修改文件的文件夹。

注意：需要保证目标文件是可读写的。

3. 运行"MigrationScript.exe"
4. 在工作目录下会生成一个备份文件夹。它包含目录下所有升级前的旧文件。还有一个日志文件"trace.log"总结了所有执行的更新。

4.2 使用自动脚本的升级步骤

更新用户项目的设置

1. 更新工具链的启动文件
   a) 项目文件：产品连接和闪存自举程序。这些文件由最新版本的工具链提供的，工具链应当支持 STM32 大容量，中容量和小容量产品。参阅工具链相关文档获取更多信息。
   b) 连接器设置：这些文件已由标准外设库 V3.0.0 包提供，位于以下路径：
   STM32F10x_StdPeriph_Lib_V3.0.0\Project\Template
   c) 向量表位置文件：这些文件已由标准外设库 V3.0.0 包提供，位于以下路径：
   STM32F10x_StdPeriph_Lib_V3.0.0\\Libraries\CMSIS\Core\CM3\startup
   注意只需选中一个启动文件
2. 将文件夹STM32F10xFWLib替换为STM32F10x_StdPeriph_Lib_V3.0.0\Libraries
3. 从用户工作区设置里移除下列文件：stm32f10x_lib.c，stm32f10x_nvic.c，stm32f10x_systick.c和cortexm3_macro.s。原因是标准外设库V3.0.0包不提供这些文件
4. 如果用户使用了NVIC中断IRQ设置和SysTick时钟源设置，那么应当在用户项目中加入文件"misc.c"
5. 根据使用的工具链，用户应当从项目工作区设置中移除文件stm32f10x_vector.s/.c，将它们替换为相应产品的启动文件
6. 更新项目的包含路径
   – 移除路径：FWLib\library\inc
   – 添加下列路径：
   STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\CMSIS\Core\CM3，这样包括了 CMSIS 内核设备访问层文件和 STM32F10xxx CMSIS 微控制器外设访问层的文件
   – 如果用户应用程序开发基于标准外设库外设驱动，添加下列路径：
   STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\STM32F10x_StdPeriph_Driver\inc

更新用户应用程序源文件

7. 将文件"stm32f10x_conf.h"替换为标准外设库V3.0.0附带的最新版本，根据用户用到的外设，去掉文件中相应行的注释符号。如果用户用NVIC驱动进行STM32F10x IRQ设置，那么也要去掉#include "misc.h"那行的注释符号。

8. 如果需要使用断言函数，则去掉文件stm32f10x.h中/* #define USE_FULL_ASSERT 1 */的注释符号
9. 文件stm32f10x_it.c/.h包含外设ISR(中断服务程序)，在标准外设库V3.0.0中，为了支持新的外设中断IRQ通道命名，ST更新了这两个文件。因此，在用户把代码从旧的stm32f10x_it.c复制到新的中以后，还需要把stm32f10x_it.h替换为在标准外设库V3.0.0提供的新版本。

注意： *1. 用户只要复制用到的STM32F10xxx IRQ处理程序即可。如果使用了下列IRQ处理程序：USB_HP_CAN_TX_IRQHandler，USB_LP_CAN_RX0_IRQHandler，CAN_RX1_IRQHandler和CAN_SCE_IRQHandler，用户需要把它们更名为USB_HP_CAN1_TX_IRQHandler，USB_LP_CAN1_RX0_IRQHandler，CAN1_RX1_IRQHandler和CAN1_SCE_IRQHandler*

2. 不要忘记在文件stm32f10x_it.h中定义相应IRQ处理程序的函数原型。

10. 运行自动peril脚本，执行所有可能的固件更新(包括类型，处理程序名称，内核宏名称)。可以查阅文件"trace.log"来查看更新的细节。
11. 将用户应用程序中NVIC和SysTick的函数替换为文件core_cm3.h定义的CMSIS函数如表10。
12. 由于标准外设库不再支持Debug模式，移除下列代码

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

13. 文件stm32f10x.h中的对库进行设置的部分：

- 文件stm32f10x_it.h中，define符合用户使用产品型号的语句：STM32F10x_LD，STM32F10x_MD或者STM32F10x_HD
- 用户通过define USE_STDPERIPHE_DRIVER选择是否使用STM32F10xxx标准外设驱动
- 设置HSE(外部晶振)，HSE启动时间超时，HSI(内部高速振荡器)的不同值

步骤14，15可选

14. 如果用户把stm32f10x_flash驱动仅用于设置闪存延时和使能预取指缓存，那么完全可以移除stm32f10x_flash驱动。而以文件"system_stm32f10x.h/.c"的SystemInit()函数取而代之。因为该函数初始化了嵌入式闪存接口。此外，还初始化了PLL并更新变量SystemFrequency。
15. 文件"system_stm32f10x.h/.c"提供的函数SystemInit()能够完成时钟配置和闪存配置：

- 用户需要通过去掉下面相应行的注释来选择系统时钟的频率

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

- 用户如果使用STM32F10xxx大容量产品，也可以在应用程序中使用外部SRAM：
  - 去掉/* #define DATA_IN_ExtSRAM */的注释
  - 如果该存储器用于应用程序所有的数据，堆和栈，那么用户必须根据使用的工具链选择相应的连接文件。参阅例程 SRAM_DataMemory 获取更多细节。
- 用户在项目中包含文件"system_stm32f10x.c"，即可在程序中调用函数SystemInit

16. 重新编译全部文件

附录A 固件库(FWLib)V2.0.3升级到标准外设库(StdPeriph_Lib)V3.0.0的具体步骤

本章节描述了如何在不使用自动脚本的情况下，把基于固件库V2.0.3开发的应用程序，升级到标准外设库V3.0.0。

用户需要遵循以下步骤，来将应用程序升级为可在标准外设库V3.0.0下运行。

更新用户项目的设置

1. 更新工具链的启动文件
   a) 项目文件：产品连接和闪存自举程序。这些文件由最新版本的工具链一起提供的，工具链应当支持 STM32 大容量，中容量和小容量产品。参阅工具链相关文档获取更多信息。
   b) 连接器设置：这些文件已由标准外设库 V3.0.0 包提供，位于以下路径：STM32F10x_StdPeriph_Lib_V3.0.0\Project\Template
   c) 向量表位置文件：这些文件已由标准外设库 V3.0.0 包提供，位于以下路径：STM32F10x_StdPeriph_Lib_V3.0.0\\Libraries\CMSIS\Core\CM3\startup
   注意只需选中一个启动文件
2. 将文件夹STM32F10xFWLib替换为STM32F10x_StdPeriph_Lib_V3.0.0\Libraries
3. 从用户工作区设置里移除下列文件：stm32f10x_lib.c，stm32f10x_nvic.c，stm32f10x_systick.c和cortexm3_macro.s。原因是标准外设库V3.0.0包不提供这些文件
4. 如果用户使用了NVIC中断IRQ设置和SysTick时钟源设置，那么应当在用户项目中加入文件”misc.c”
5. 根据使用的工具链，用户应当从项目工作区设置中移除文件stm32f10x_vector.s/.c，将它们替换为相应产品的启动文件
6. 更新项目的包含路径
   – 移除路径：FWLib\library\inc
   – 添加下列路径：STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\CMSIS\Core\CM3，这样包括了 CMSIS 内核设备访问层文件和 STM32F10xxx CMSIS 微控制器外设访问层的文件
   – 如果用户应用程序开发基于标准外设库外设驱动，添加下列路径：STM32F10x_StdPeriph_Lib_V3.0.0\Libraries\STM32F10x_StdPeriph_Driver\inc

更新用户应用程序源文件

7. 将文件”stm32f10x_conf.h”替换为标准外设库V3.0.0附带的最新版本，根据用户用到的外设，去掉文件中相应行的注释符号。如果用户用NVIC驱动进行STM32F10x IRQ设置，那么也要去掉#include “misc.h”那行的注释符号。
8. 如果用户需要使用断言函数，那么去掉文件stm32f10x.h中/* #define USE_FULL_ASSERT 1 */的注释符号
9. 文件stm32f10x_it.c/.h包含外设ISR(中断服务程序)，在标准外设库V3.0.0中，为了支持新的外设中断IRQ通道命名，ST更新了这两个文件。因此，在用户把代码从旧的stm32f10x_it.c复制到新的中以后，还需要把stm32f10x_it.h替换为在标准外设库V3.0.0提供的新版本。

注意： *1. 用户只要复制用到的STM32F10xxx IRQ处理程序即可。如果使用了下列IRQ处理程序：USB_HP_CAN_TX_IRQHandler，USB_LP_CAN_RX0_IRQHandler，CAN_RX1_IRQHandler和CAN_SCE_IRQHandler，用户需要把它们更名为USB_HP_CAN1_TX_IRQHandler，USB_LP_CAN1_RX0_IRQHandler，CAN1_RX1_IRQHandler和CAN1_SCE_IRQHandler*

2. 不要忘记在文件stm32f10x_it.h中定义相应IRQ处理程序的函数原型。

10. 把所有的中断号码名称由PPP_IRQChannel改为PPP_IRQn。
11. 如表10，将用户应用程序中NVIC和SysTick的函数替换为文件core_cm3.h定义的CMSIS函数。
12. 按照CMSIS内核设备访问层的定义修改用户的相关宏，在项目中应当包含文件core_cm3.h

13. 把代码中所有出现的”stm32f10x_lib.h”改为”stm32f10x.h”。标准外设库的入口是文件stm32f10x.h
14. 由于标准外设库不再支持Debug模式，移除下列代码

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

15. 如果在主程序中用到断言函数，那么把”#ifdef DEBUG”改为”ifdef USE_FULL_ASSERT”
16. 把所有的类型修改成符合<stdint.h>的类型
17. 如果用户把stm32f10x_flash驱动仅用于设置闪存延时和使能预取指缓存，那么完全可以移除stm32f10x_flash驱动。而以文件”system_stm32f10x.h/.c”的SystemInit()函数取而代之。因为该函数初始化了嵌入式闪存接口。此外，还初始化了PLL并更新变量SystemFrequency。
18. 文件stm32f10x_it.h中的库设置部分：

- 文件stm32f10x_it.h中，define符合用户使用产品型号的语句：STM32F10x_LD，STM32F10x_MD或者STM32F10x_HD
- 用户通过define USE_STDPERIPHE_DRIVER选择是否使用STM32F10xxx标准外设驱动
- 设置HSE(外部晶振)，HSE启动时间超时，HSI(内部高速振荡器)的不同值

19. 文件”system_stm32f10x.h/.c”提供的函数SystemInit()能够完成时钟配置和闪存配置：

- 用户需要通过去掉下面相应行的注释来选择系统时钟的频率

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

- 用户如果使用STM32F10xxx大容量产品，也可以在应用程序中使用外部SRAM：
  - 去掉/* #define DATA_IN_ExtSRAM */的注释
  - 如果该存储器用于应用程序所有的数据，堆和栈，那么用户必须根据使用的工具链选择相应的连接文件。参阅例程 SRAM_DataMemory 获取更多细节。
- 用户在项目中包含文件”system_stm32f10x.c”，即可在程序中调用函数SystemInit