```
}
```

这个时候你不需要再直接去操作 BRR 寄存器了，你只需要知道怎么使用 GPIO_ResetBits()这个函数就可以了。在你对外设的工作原理有一定的了解之后，你再去看固件库函数，基本上函数名字能告诉你这个函数的功能是什么，该怎么使用，这样是不是开发会方便很多？

任何处理器，不管它有多么的高级，归根结底都是要对处理器的寄存器进行操作。但是固件库不是万能的，您如果想要把 STM32 学透，光读 STM32 固件库是远远不够的。你还是要了解一下 STM32 的原理，而这些原理了解了，你在进行固件库开发过程中才可能得心应手游刃有余。

### 3.1.2 STM32 固件库与 CMSIS 标准讲解

前一节我们讲到，STM32 固件库就是函数的集合，那么对这些函数有什么要求呢？？这里就涉及到一个 CMSIS 标准的基础知识，这部分知识可以从《Cortex-M3 权威指南》中了解到，我们这里只是对权威指南的讲解做个概括性的介绍。经常有人问到 STM32 和 ARM 以及 ARM7 是什么关系这样的问题，其实 ARM 是一个做芯片标准的公司，它负责的是芯片内核的架构设计，而 TI，ST 这样的公司，他们并不做标准，他们是芯片公司，他们是根据 ARM 公司提供的芯片内核标准设计自己的芯片。所以，任何一个做 Cortex-M3 芯片，他们的内核结构都是一样的，不同的是他们的存储器容量，片上外设，IO 以及其他模块的区别。所以你会发现，不同公司设计的 Cortex-M3 芯片他们的端口数量，串口数量，控制方法这些都是有区别的，这些资源他们可以根据自己的需求理念来设计。同一家公司设计的多种 Cortex-m3 内核芯片的片上外设也会有很大的区别，比如 STM32F103RBT 和 STM32F103ZET，他们的片上外设就有很大的区别。我们可以通过《Cortex-M3 权威指南》中的一个图来了解一下：

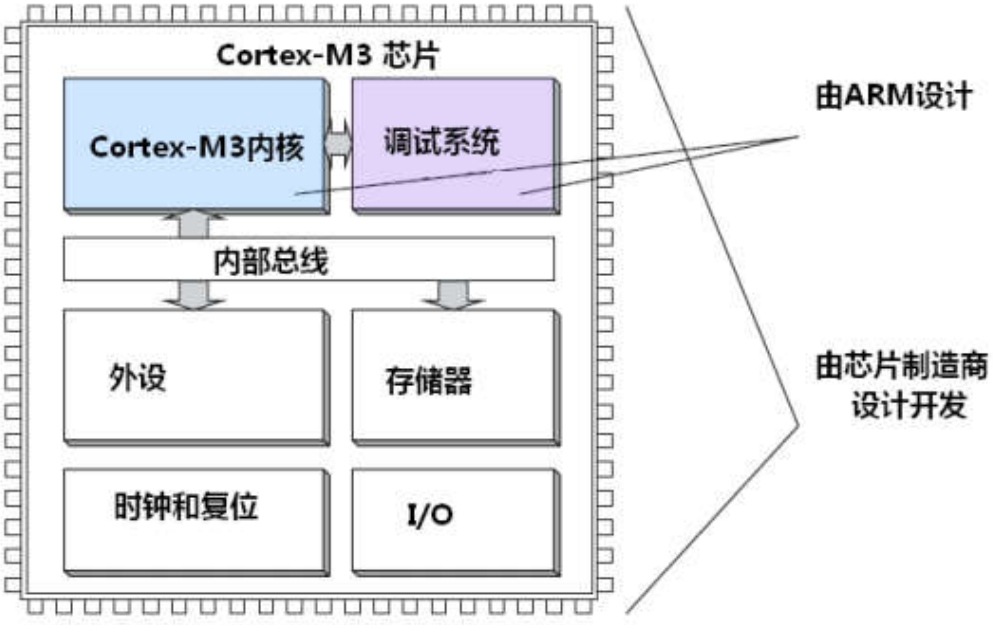


图 3.1.2.1 Cortex-m3 芯片结构

从上图可以看出，芯片虽然是芯片公司设计，但是内核却要服从 ARM 公司提出的 Cortex-M3 内核标准了，理所当然，芯片公司每卖出一片芯片，需要向 ARM 公司交一定的专利费。

既然大家都使用的是 Cortex-M3 核，也就是说，本质上大家都是一样的，这样 ARM 公司为了能让不同的芯片公司生产的 Cortex-M3 芯片能在软件上基本兼容，和芯片生产商共同提出

了一套标准 CMSIS 标准(Cortex Microcontroller Software Interface Standard) ,翻译过来是“ARM Cortex™ 微控制器软件接口标准”。ST 官方库就是根据这套标准设计的。这里我们又要引用参考资料里面的图片来看看基于 CMSIS 应用程序基本结构：

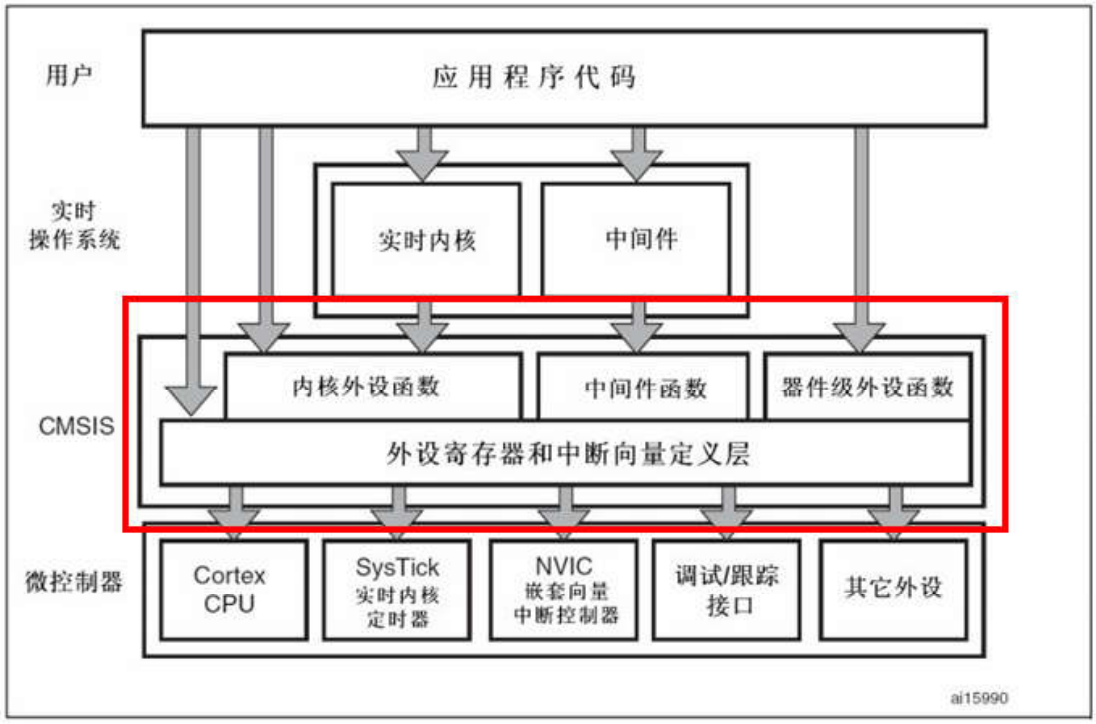


图 3.1.2.2 基于 CMSIS 应用程序基本结构

CMSIS 分为 3 个基本功能层：

1) 核内外设访问层：ARM 公司提供的访问，定义处理器内部寄存器地址以及功能函数。
2) 中间件访问层:定义访问中间件的通用 API,也是 ARM 公司提供。
3) 外设访问层：定义硬件寄存器的地址以及外设的访问函数。

从图中可以看出，CMSIS 层在整个系统中是处于中间层，向下负责与内核和各个外设直接打交道，向上提供实时操作系统用户程序调用的函数接口。如果没有 CMSIS 标准，那么各个芯片公司就会设计自己喜欢的风格的库函数，而 CMSIS 标准就是要强制规定，芯片生产公司设计的库函数必须按照 CMSIS 这套规范来设计。

其实不用这么讲这么复杂的，一个简单的例子，我们在使用 STM32 芯片的时候首先要进行系统初始化，CMSIS 规范就规定，系统初始化函数名字必须为 SystemInit，所以各个芯片公司写自己的库函数的时候就必须用 SystemInit 对系统进行初始化。CMSIS 还对各个外设驱动文件的文件名字规范化，以及函数名字规范化等等一系列规定。上一节讲的函数 GPIO_ResetBits 这个函数名字也是不能随便定义的，是要遵循 CMSIS 规范的。

至于 CMSIS 的具体内容就不必多讲了，需要了解详细的朋友可以到网上搜索资料，相关资料可谓满天飞。

### 3. 1. 3 STM32 官方库包介绍

这一节内容主要讲解 ST 官方提供的 STM32 固件库包的结构。ST 官方提供的固件库完整包可以在官方下载，我们光盘也会提供。固件库是不断完善升级的，所以有不同的版本，我们使用的是 V3.5 版本的固件库

大家可以到光盘目录: 软件资料\STM32 固件库使用参考资料\

STM32F10x_StdPeriph_Lib_V3.5.0 下面查看，这在我们论坛有下载。下面看看官方库包的目录结构：

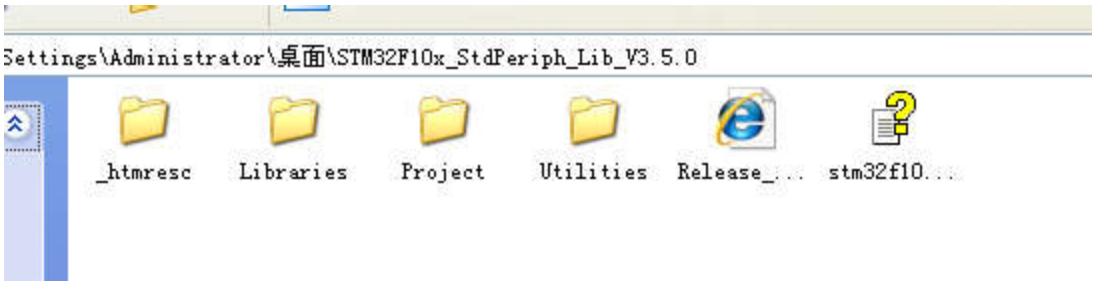


图 3.1.2.3 官方库包根目录

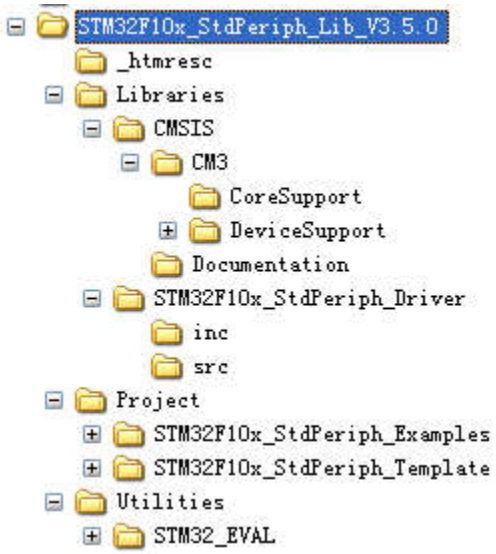


图 3.1.2.4 官方库目录列表

### 3.1.3.1 文件夹介绍：

Libraries 文件夹下面有 CMSIS 和 STM32F10x_StdPeriph_Driver 两个目录，这两个目录包含固件库核心的所有子文件夹和文件。其中 CMSIS 目录下面是启动文件，STM32F10x_StdPeriph_Driver 放的是 STM32 固件库源码文件。源文件目录下面的 inc 目录存放的是 stm32f10x_xxx.h 头文件,无需改动。src 目录下面放的是 stm32f10x_xxx.c 格式的固件库源码文件。每一个.c 文件和一个相应的.h 文件对应。这里的文件也是固件库的核心文件，每个外设对应一组文件。

Libraries 文件夹里面的文件在我们建立工程的时候都会使用到。

Project 文件夹下面有两个文件夹。顾名思义，STM32F10x_StdPeriph_Examples 文件夹下面存放的 ST 官方提供的固件实例源码，在以后的开发过程中，可以参考修改这个官方提供的实例来快速驱动自己的外设，很多开发板的实例都参考了官方提供的例程源码，这些源码对以后的学习非常重要。STM32F10x_StdPeriph_Template 文件夹下面存放的是工程模板。

Utilities 文件下就是官方评估板的一些对应源码，这个可以忽略不看。

根目录中还有一个 stm32f10x_stdperiph_lib_um.chm 文件，直接打开可以知道，这是一个固件库的帮助文档，这个文档非常有用，只可惜是英文的，在开发过程中，这个文档会经常被使用到。

### 3.1.3.2 关键文件介绍：

下面我们要着重介绍 Libraries 目录下面几个重要的文件。

core_cm3.c 和 core_cm3.h 文件位于\Libraries\CMSIS\CM3\CoreSupport 目录下面的，这个就是 CMSIS 核心文件，提供进入 M3 内核接口，这是 ARM 公司提供，对所有 CM3 内核的芯片都一样。你永远都不需要修改这个文件，所以这里我们就点到为止。

和 CoreSupport 同一级还有一个 DeviceSupport 文件夹。DeviceSupport\ST\STM32F10xt 文件夹下面主要存放一些启动文件以及比较基础的寄存器定义以及中断向量定义的文件。

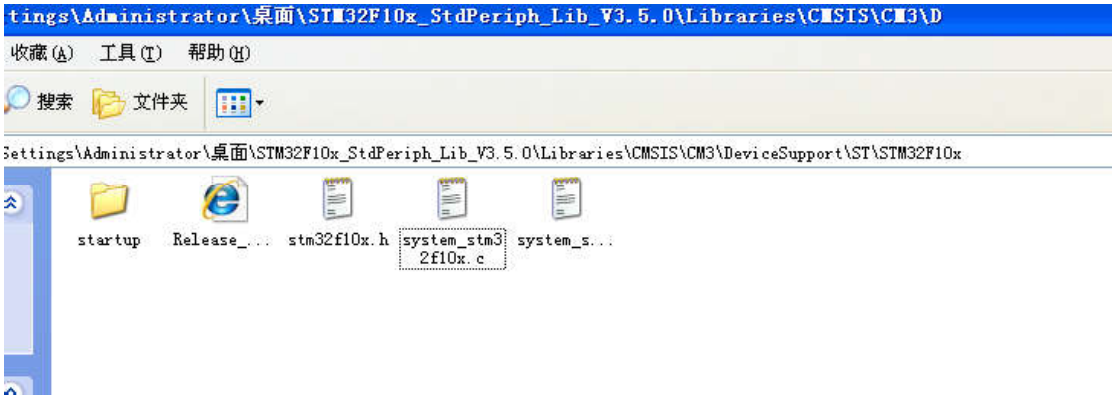


图 3.1.2.5 DeviceSupport\ST\STM32F10x 目录结构

这个目录下面有三个文件：system_stm32f10x.c, system_stm32f10x.h 以及 stm32f10x.h 文件。其中 system_stm32f10x.c 和对应的头文件 system_stm32f10x.h 文件的功能是设置系统以及总线时钟，这个里面有一个非常重要的 SystemInit()函数，这个函数在我们系统启动的时候都会调用，用来设置系统的整个时钟系统。

stm32f10x.h 这个文件就相当重要了，只要你做 STM32 开发，你几乎时刻都要查看这个文件相关的定义。这个文件打开可以看到，里面非常多的结构体以及宏定义。这个文件里面主要是系统寄存器定义申明以及包装内存操作，对于这里是怎样申明以及怎样将内存操作封装起来的，我们在后面的章节“MDK 中寄存器地址名称映射分析”中会讲到。

在 DeviceSupport\ST\STM32F10x 同一级还有一个 startup 文件夹，这个文件夹里面放的文件顾名思义是启动文件。在\startup\arm 目录下，我们可以看到 8 个 startup 开头的.s 文件。

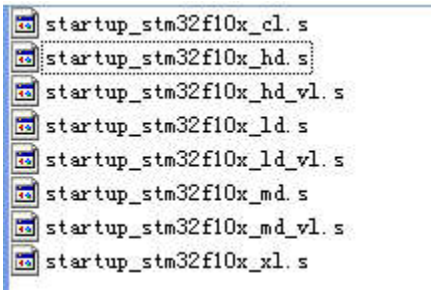


图 3.1.2.6 startup 文件

这里之所以有 8 个启动文件，是因为对于不同容量的芯片启动文件不一样。对于 103 系列，主要是用其中 3 个启动文件：

startup_stm32f10x_ld.s： 适用于小容量 产品

startup_stm32f10x_md.s ： 适用于中等容量产品

startup_stm32f10x_hd.s： 适用于大容量产品

这里的容量是指 FLASH 的大小.判断方法如下：

小容量：FLASH≤32K

中容量：64K≤FLASH≤128K

大容量：256K≤FLASH

我们 ALIENTEK STM32 战舰板，精英板以及 mini 板采用的 STM32F103ZET6 和 stm32F103RCT6 芯片都属于大容量产品，所以我们的启动文件选择 startup_stm32f10x_hd.s,对于中等容量芯片请选择 startup_stm32f10x_md.s 启动文件，小容量芯片请选择 startup_stm32f10x_ld.s。

启动文件到底什么作用，其实我们可以打开启动文件进去看看。启动文件主要是进行堆栈之类的初始化，中断向量表以及中断函数定义。启动文件要引导进入 main 函数。Reset_Handler 中断函数是唯一实现了的中断处理函数，其他的中断函数基本都是死循环。 Reset_handler 在我们系统启动的时候会调用，下面让我们看看 Reset_handler 这段代码：

```
; Reset handler
Reset_Handler    PROC
                 EXPORT  Reset_Handler             [WEAK]
                 IMPORT  __main
                 IMPORT  SystemInit
                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP
```

这段代码我也看不懂，反正就知道，这里面要引导进入 main 函数，同时在进入 main 函数之前，首先要调用 SystemInit 系统初始化函数。

还有其他几个文件 stm32f10x_it.c,stm32f10x_it.h 以及 stm32f10x_conf.h 等文件，这里就不一一介绍。stm32f10x_it.c 里面是用来编写中断服务函数，中断服务函数也可以随意编写在工程里面的任意一个文件里面，个人觉得这个文件没太大意义。

stm32f10x_conf.h 文件打开可以看到一堆的#include,这里你建立工程的时候，可以注释掉一些你不用的外设头文件。这里相信大家一看就明白。

这一节我们就简要介绍到这里，后面我们会介绍怎样建立基于 V3.5 版本固件库的工程模块。

## 3.2 MDK5 简介

MDK 源自德国的 KEIL 公司，是 RealView MDK 的简称。在全球 MDK 被超过 10 万的嵌入式开发工程师使用。目前最新版本为：MDK5.14，该版本使用 uVision5 IDE 集成开发环境，是目前针对 ARM 处理器，尤其是 Cortex M 内核处理器的最佳开发工具。

MDK5 向后兼容 MDK4 和 MDK3 等，以前的项目同样可以在 MDK5 上进行开发(但是头文件方面得全部自己添加)， MDK5 同时加强了针对 Cortex-M 微控制器开发的支持，并且对传统的开发模式和界面进行升级，MDK5 由两个部分组成：MDK Core 和 Software Packs。其中，Software Packs 可以独立于工具链进行新芯片支持和中间库的升级。如图 3.1.1 所示：