零死角玩转STM32

与野火同行 乐意惬无边

1 原创教程，完全开源。

2 通俗易懂，详尽解读。

3 由浅入深，结合实操。

4 配套板子，全面玩转。

5 强强联合，不断更新。

野火团队 Wild Fire Team

0、友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬无边。

另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！

4、初识 STM32 库

本章通过简单介绍 STM32 库的各个文件及其关系，让读者建立 STM32 库的概念，看完后对库有个总体印象即可，在后期实际开发时接触了具体的库时，再回头看看这一章，相信你对 STM32 库又会有一个更深刻的认识。

4.1 STM32 神器之库开发

4.1.1 什么是 STM32 库？

在 51 单片机的程序开发中，我们直接配置 51 单片机的寄存器，控制芯片的工作方式，如中断，定时器等。配置的时候，我们常常要查阅寄存器表，看用到哪些配置位，为了配置某功能，该置 1 还是置 0。这些都是很琐碎的、机械的工作，因为 51 单片机的软件相对来说较简单，而且资源很有限，所以可以直接配置寄存器的方式来开发。

STM32 库是由 ST 公司针对 STM32 提供的*函数接口，即 **API (Application Program Interface)***，开发者可调用这些函数接口来配置 STM32 的寄存器，使开发人员得以*脱离最底层的寄存器操作，有开发快速，易于阅读，维护成本低等优点。*

当我们调用库的 API 的时候可以不用挖空心思去了解库底层的寄存器操作，就像当年我们学习 C 语言的时候，用 prinft()函数时只是学习它的使用格式，并没有去研究它的源码实现，如非必要，可以说是老死不相往来。

实际上，*库是架设在寄存器与用户驱动层之间的代码，向下处理与寄存器直接相关的配置，向上为用户提供配置寄存器的接口。*库开发方式与直接配置寄存器方式的区别见**错误！未找到引用源。**4-1。

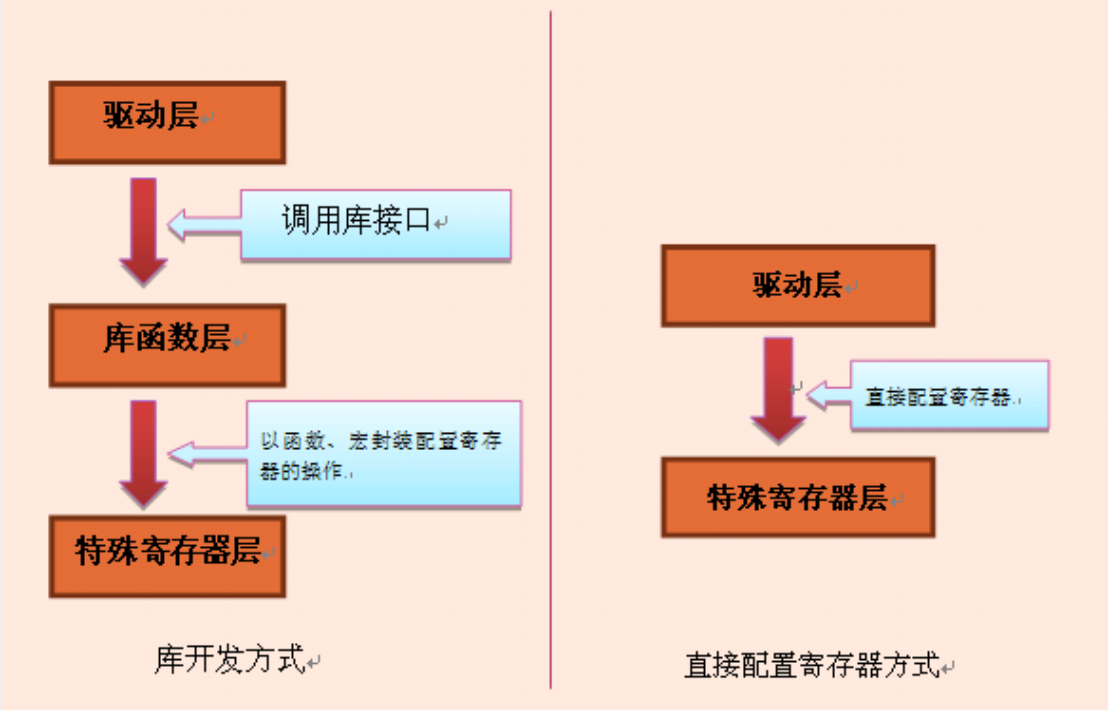


图 4-1

4.1.2 为什么采用库来开发？

对于 STM32，因为外设资源丰富，带来的必然是寄存器的数量和复杂度的增加，这时直接配置寄存器方式的缺陷就突显出来了：

1) 开发速度慢
2) 程序可读性差

这两个缺陷直接影响了开发效率，程序维护成本，交流成本。库开发方式则正好弥补了这两个缺陷。

而坚持采用直接配置寄存器的方式开发的程序员，会列举以下原因：

1) 更直观
2) 程序运行占用资源少

初学 STM32 的读者，普遍因为第一个原因而选择以直接配置寄存器的方法来学习。认为这种方法直观，能够了解到是配置了哪些寄存器，怎样配置寄存器。*事实上，库函数的底层实现恰恰是直接配置寄存器方式的最佳例子，想深入了解芯片是如何工作的话，只要追踪到库的最底层实现就能理解，相信你会为它严谨、优美的实现方式而陶醉。要想修炼 C 语言，就从 ST 的库开始吧。*这将在 **错误！未找到引用源。**进行详细分析。

相对于库开发的方式，直接配置寄存器方式生成的代码量的确会少一点，但因为 STM32 有充足的资源，权衡库的优势与不足，绝大部分时候，我们愿意牺牲一点资源，选择库开发。一般只有在对代码运行时间要求极苛刻的地方，才用直接配置寄存器的方式代替，如频繁调用的中断服务函数。

对于库开发与直接配置寄存器的方式，在 STM32 刚推出时引起程序员的激烈争论，但是，随着 ST 库的完善与大家对库的了解，更多的程序员选择了库开发。

本书采用 STM32 的库进行讲解，既介绍如何使用库接口，也讲解库接口的实现方式。使读者既能利用库进行快速开发，也能深入了解 STM32 的工作原理。

为进一步解答读者为什么使用库开发，请读者先思考一下为会么采用 c 语言开发软件而不是采用汇编。相比之下，可以发现调用库接口开发与直接配置寄存器开发的关系，犹如 c 语言与汇编的关系。见表 4-1

特点	汇编语言	直接配置寄存器方式
更接近机器思维（直观）	汇编指令为机器码的助记符，能直接了解 cpu 的操作	直接针对寄存器的某些位进行置 1 或清 0 操作，能清晰地看到驱动代码使用了什么寄存器
运行效率	代码为 cpu 直接执行的指令，与编译器优化无关	没有库函数层，省去代码为分层而消耗的资源

和表 4-2。

据某无从考证的 IT 大师说过，*“一切计算机科学的问题都可以用分层来解决。”*从汇编到 c，从直接配置寄存器到使用库，从裸机到系统，从操作系统到应用层软件，无不体现着这样的分层思想。开发的软件多了，跨越的软件层次多了，会深刻地认同他这句话，分层思想在软件开发上体现得淋漓尽致，分层使得问题变得更简单，使得能够屏蔽下层实现方式的差异，使得软件开发变成简单的调用函数接口，而不用管它的实现，大大提高效率。

库就是建立了一个新的软件抽象层，库的优点，其实就是分层的优点，库的缺点，也是软件分层带来的缺点，而对于 STM32 这样高性能的芯片，我想我们会愿意承受分层带来的缺点的。

特点	C 语言	库开发方式
更接近人的思维（易读）	程序控制语句结构化。以函数作为程序单位便于模块化	1)用结构体封装寄存器参数 2)用宏表示参数，意义明确 3)用函数封装对寄存器的操作
移植性好	程序基本上不做修改就可应用于各种计算机上	代码的易读性使驱动代码的修改变得非常方便

表 4-1

特点	汇编语言	直接配置寄存器方式
更接近机器思维（直观）	汇编指令为机器码的助记符，能直接了解 cpu 的操作	直接针对寄存器的某些位进行置 1 或清 0 操作，能清晰地看到驱动代码使用了什么寄存器
运行效率	代码为 cpu 直接执行的指令，与编译器优化无关	没有库函数层，省去代码为分层而消耗的资源

表 4-2

4.2 STM32 结构及库层次关系

4.2.1 CMSIS 标准

我们知道由 ST 公司生产的 STM32 采用的是 Cortex-M3 *内核*，内核是整个微控制器的 CPU。该内核是 ARM 公司设计的一个处理器体系架构。ARM 公司并不生产芯片，而是出售其芯片技术授权。ST 公司或其它芯片生产厂商如 TI，负责设计的是在内核之外的部件，被称为*核外外设*或*片上外设、设备外设*。如芯片内部的模数转换外设 ADC、串口 UART、定时器 TIM 等。内核与外设，如同 PC 上的 CPU 与主板、内存、显卡、硬盘的关系。见**错误！未找到引用源。**。

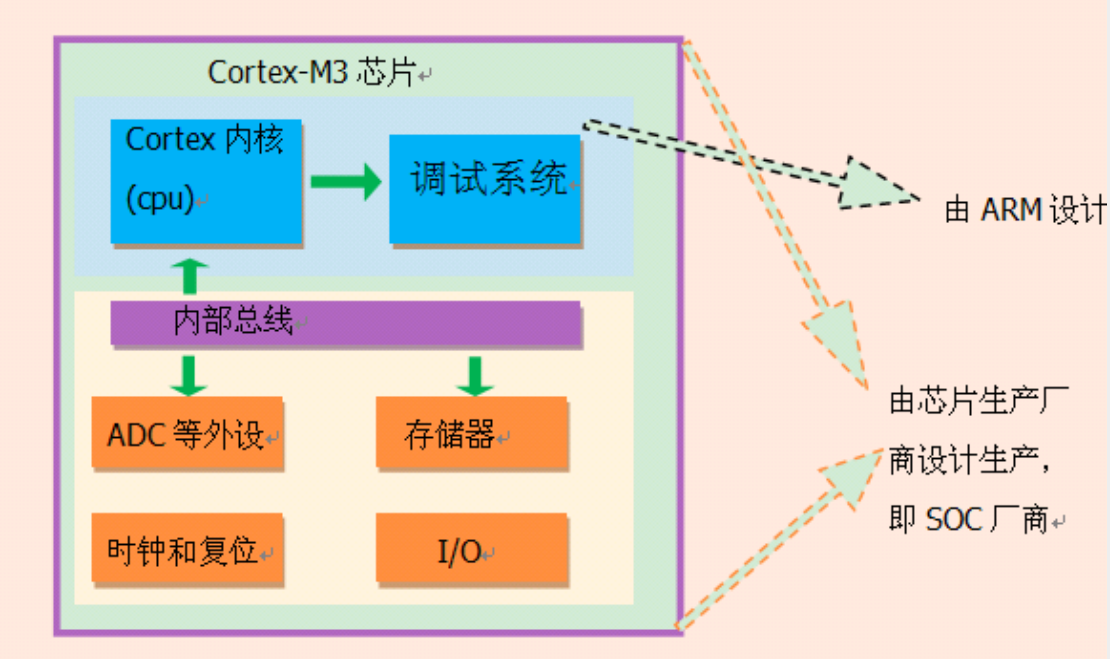
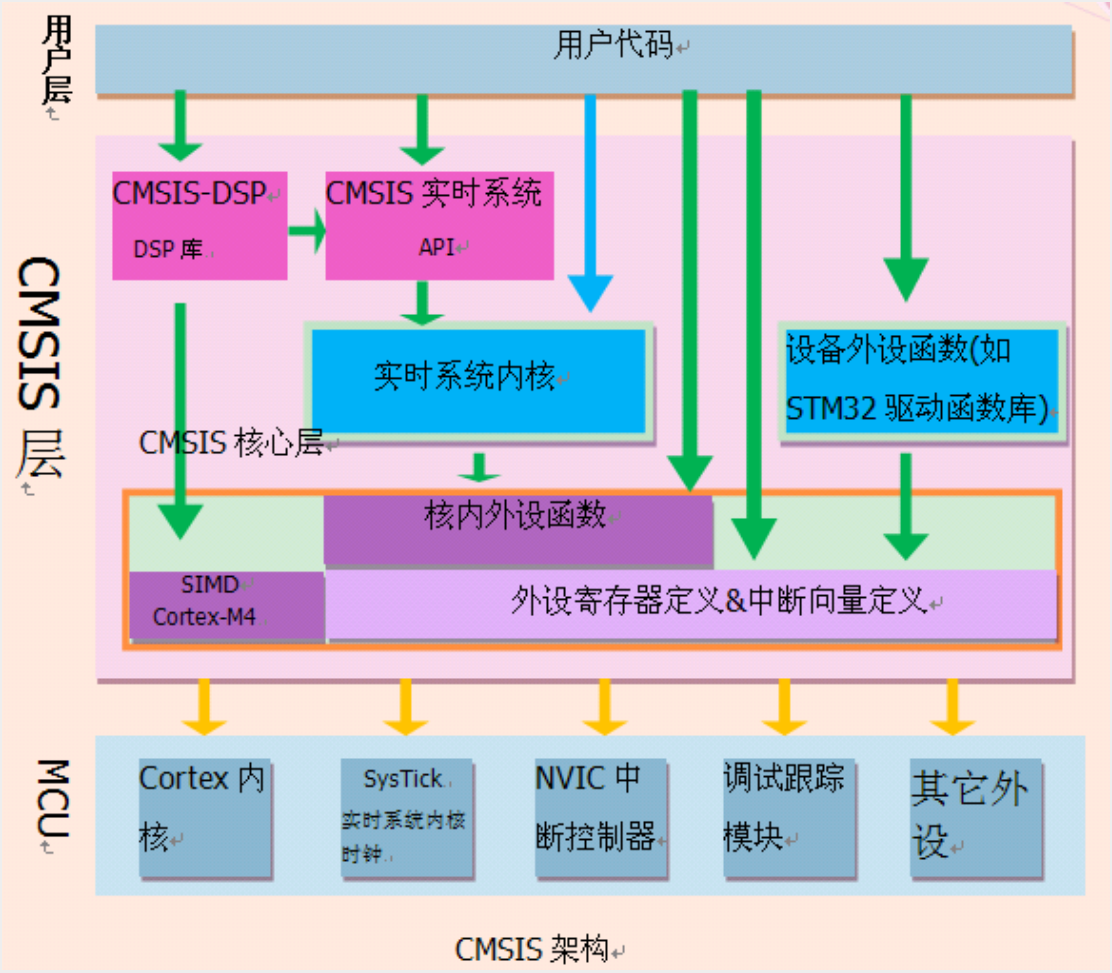


图 4-2

因为基于 Cortex 的某系列芯片采用的内核都是相同的，区别主要为核外的片上外设的差异，这些差异却导致软件在同内核，不同外设的芯片上移植困难。为了解决不同的芯片厂商生产的 Cortex 微控制器*软件*的兼容性问题，ARM 与芯片厂商建立了 CMSIS 标准(Cortex MicroController Software Interface Standard)。

所谓 CMSIS 标准，实际是新建了一个软件抽象层。见**错误！未找到引用源。**。



错误！未找到引用源。

CMSIS 标准中最主要的为 CMSIS 核心层，它包括了：

- 内核函数层：其中包含用于访问内核寄存器的名称、地址定义，主要由 ARM 公司提供。
- 设备外设访问层：提供了片上的核外外设的地址和中断定义，主要由芯片生产商提供。

可见 CMSIS 层位于硬件层与操作系统或用户层之间，提供了与芯片生产商无关的硬件抽象层，可以为接口外设、实时操作系统提供简单的处理器软件接口，屏蔽了硬件差异，这对软件的移植是有极大的好处的。STM32 的库，就是按照 CMSIS 标准建立的。

4.2.2 库目录、文件简介

STM32 的 3.5 版库可以从官网获得，也可以直接从本书的附录光盘得到。本书主要采用最新版的 3.5 库文件，在高级篇的章节有部分代码是采用 3.0 的库开发的，因为 3.5 与 3.0 的库文件兼容性很好，对于旧版的代码我们仍然使用用 3.0 版的。

解压后进入库目录：

stm32f10x_stdperiph_lib\STM32F10x_StdPeriph_Lib_V3.5.0

各文件夹内容说明见图 4-1

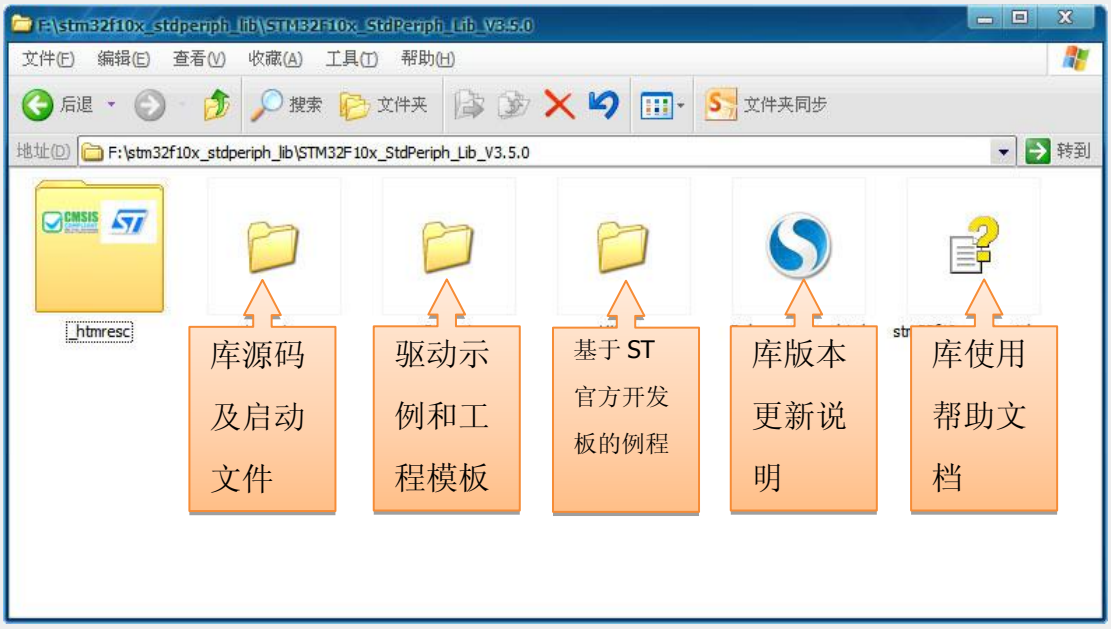


图 4-1

Libraries 文件夹下是驱动库的源代码及启动文件。

Project 文件夹下是用驱动库写的例子跟一个工程模板。

还有一个已经编译好的 ***HTML*** 文件，是*库帮助文档*，主要讲的是如何使用驱动库来编写自己的应用程序。说得形象一点，这个 ***HTML*** 就是告诉我们：ST 公司已经为你写好了每个外设的驱动了，想知道如何运用这些例子就来向我求救吧。不幸的是，这个帮助文档是英文的，这对很多英文不好的朋友来说是一个很大的障碍。但野火要告诉大家，英文仅仅是一种工具，绝对不能让它成为我们学习的障碍。其实这些英文还是很简单的，我们需要的是拿下它的勇气。

网上流传有一份中文版本的库帮助文档，但那个是 2.x 版本的，但 3.x 以上版本的目录结构和库函数接口跟 2.x 版本的区别还是比较大的，这点大家要注意下。

在使用库开发时，我们需要把 ***libraries*** 目录下的库函数文件添加到工程中，并查阅*库帮助文档*来了解 ST 提供的库函数，这个文档说明了每一个库函数的使用方法。

进入 ***Libraries*** 文件夹看到，关于内核与外设的库文件分别存放在 ***CMSIS*** 和 ***STM32F10x_StdPeriph_Driver*** 文件夹中。

Libraries|CMSIS|CM3 文件夹下又分为 ***CoreSupport*** 和 ***DeviceSupport*** 文件夹。

4.2.2.1 core_cm3.c 文件

在 ***CoreSupport*** 中的是位于 CMSIS 标准的*核内设备函数层* 的 M3 核通用的源文件 ***core_cm3.c*** 和头文件 ***core_cm3.h***，它们的作用是为那些采用 Cortex-M3 核设计 SOC 的芯片商设计的芯片外设提供一个进入 M3 内核的接口。这两个文件在其它公司的 M3 系列芯片也是相同的。至于这些功能是怎样用源码实现的，我们可以不用管它，我们只需把这个文件加进我们的工程文件即可，有兴趣的朋友可以深究。

core_cm3.c 文件还有一些与编译器相关条件编译语句，用于屏蔽不同编译器的差异，我们在开发时不用管这部分，有兴趣可以了解一下。里面包含了一些跟编译器相关的信息，如：RealView Compiler (RVMDK)，ICC Compiler (IAR)，GNU Compiler。

```
1.  /* define compiler specific symbols */
2.  #if defined ( __CC_ARM   )
3.    #define __ASM            __asm
4.    #define __INLINE         __inline
5.
6.  #elif defined ( __ICCARM__ )
7.    #define __ASM            __asm
8.    #define __INLINE         inline
9.  #elif defined   (  __GNUC__  )
10.   #define __ASM            __asm
      #define __INLINE         inline
11.
12. #elif defined   (  __TASKING__  )
13.   #define __ASM            __asm
```

使用 RVMDK 编译器时的嵌入汇编与内联函数的关键字形式

使用 IAR 编译器时的形式

```
14.  #define __INLINE         inline
15.  #endif
```

较重要的是在 *core_cm3.c* 文件中包含了 **stdin.h** 这个头文件，这是一个 ANSI C 文件，是独立于处理器之外的，就像我们熟知的 C 语言头文件 stdio.h 文件一样。位于 RVMDK 这个软件的安装目录下，主要作用是提供一些新类型定义，如：

```
1.  /* exact-width signed integer types */
2.  typedef   signed          char int8_t;
3.  typedef   signed short     int int16_t;
4.  typedef   signed           int int32_t;
5.  typedef   signed       __int64 int64_t;
6.
7.  /* exact-width unsigned integer types */
8.  typedef unsigned          char uint8_t;
9.  typedef unsigned short     int uint16_t;
10. typedef unsigned           int uint32_t;
11. typedef unsigned       __int64 uint64_t;
```

这些新类型定义屏蔽了在不同芯片平台时，出现的诸如 int 的大小是 16 位，还是 32 位的差异。所以在我们以后的程序中，都将使用新类型如 ***int8_t*** 、***int16_t*****……**

在稍旧版的程序中还可能会出现如 ***u8***、***u16***、***u32*** 这样的类型，请尽量避免这样使用，在这里提出来是因为初学时如果碰到这样的旧类型让人一头雾水，而且在以新的库建立的工程中是无法追踪到 ***u8***、***u16***、***u32*** 这些的定义的。

core_cm3.c 跟启动文件一样都是底层文件，都是由 ARM 公司提供的，遵守 CMSIS 标准，即所有 CM3 芯片的库都带有这个文件，这样软件在不同的 CM3 芯片的移植工作就得以简化。

4.2.2.2 system_stm32f10x.c 文件

在 ***DeviceSupport*** 文件夹下的是启动文件、*外设寄存器定义&中断向量定义层* 的一些文件，这是由 ST 公司提供的。见图 4-2

图 4-2

system_stm32f10x.c，是由 ST 公司提供的，遵守 CMSIS 标准。该文件的功能是设置系统时钟和总线时钟， M3 比 51 单片机复杂得多，并不是说我们外部给一个 8M 的晶振，M3 整个系统就以 8M 为时钟协调整个处理器的工作。我们还要通过 M3 核的核内寄存器来对 8M 的时钟进行倍频，分频，或者使用芯片内部的时钟。所有的外设都与时钟的频率有关，所以这个文件的时钟配置是很关键的。

system_stm32f10x.c 在实现系统时钟的时候要用到 PLL（锁相环），这就需要操作寄存器，寄存器都是以存储器映射的方式来访问的，所以该文件中包含了 ***stm32f10x.h*** 这个头文件。

4.2.2.3 stm32f10x.h 文件

stm32f10x.h 这个文件非常重要，是一个非常底层的文件。

所有处理器厂商都会将对内存的操作封装成一个宏，即我们通常说的寄存器，并且把这些实现封装成一个系统文件，包含在相应的开发环境中。这样，我们在开发自己的应用程序的时候只要将这个文件包含进来就可以了。

4.2.2.4 启动文件

Libraries|CMSIS|Core|CM3|startup|arm 文件夹下是由汇编编写的系统启动文件，不同的文件对应不同的芯片型号，在使用时要注意。见图 4-3

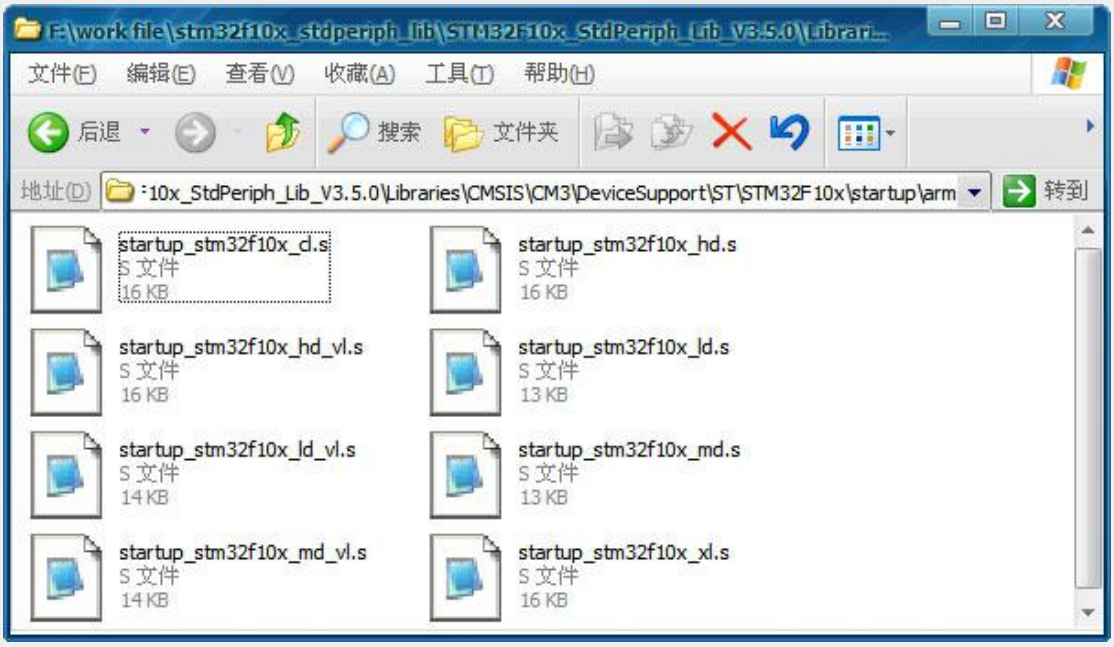


图 4-3

文件名的英文缩写的意义如下：

cl：互联型产品，stm32f105/107 系列
vl：超值型产品，stm32f100 系列
xl：超高密度（容量）产品，stm32f101/103 系列
ld：低密度产品，FLASH 小于 64K
md：中等密度产品，FLASH=64 or 128
hd：高密度产品，FLASH 大于 128

*野火 M3 开发板*中用的芯片是 ***STM32F103VET6，64KRAM，512KROM***，是属于高密度产品，所以启动文件要选择 ***startup_stm32f10x_hd.s。***

启动文件是任何处理器在上点复位之后最先运行的一段汇编程序。在我们编写的 c 语言代码运行之前，需要由汇编为 c 语言的运行建立一个合适的环境，接下来才能运行我们的程序。所以我们也要把启动文件添加进我们的的工程中去。

总的来说，启动文件的作用是：

1. 初始化堆栈指针 SP;
2. 初始化程序计数器指针 PC;
3. 设置堆、栈的大小;
4. 设置异常向量表的入口地址;
5. 配置外部 SRAM 作为数据存储器（这个由用户配置，一般的开发板可没有外部 SRAM）;
6. 设置 C 库的分支入口__main（最终用来调用 main 函数）;
7. 在 3.5 版的启动文件还调用了在 ***system_stm32f10x.c*** 文件中的 ***SystemIni()*** 函数配置系统时钟，在旧版本的工程中要用户进入 main 函数自己调用 ***SystemIni()*** 函数。

4.2.2.5 STM32F10x_StdPeriph_Driver 文件夹

Libraries\STM32F10x_StdPeriph_Driver 文件夹下有 ***inc***（include 的缩写）跟 ***src***（source 的简写）这两个文件夹，这都属于 CMSIS 的 *设备外设函数* 部分。***src*** 里面是每个设备外设的驱动程序，这些外设是芯片制造商在 Cortex-M3 核外加进去的。

进入 ***libraries*** 目录下的 ***STM32F10x_StdPeriph_Driver*** 文件夹，见图 4-4。

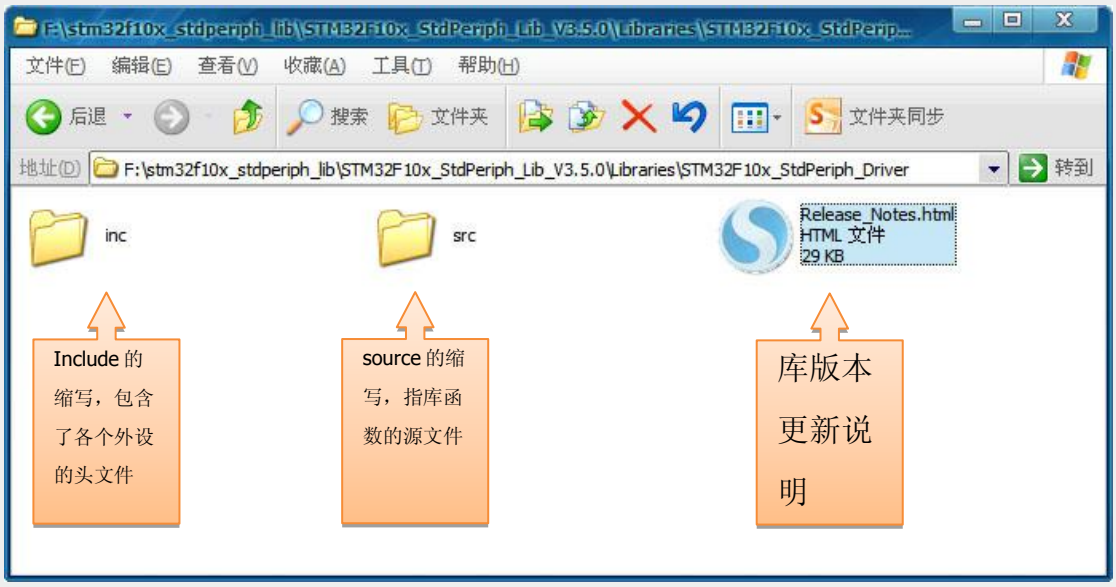


图 4-4

在 ***src*** 和 ***inc*** 文件夹里的就是 ST 公司针对每个 STM32 外设而编写的库函数文件，每个外设对应一个 ***.c*** 和 ***.h*** 后缀的文件。我们把这类外设文件统称为：***stm32f10x_ppp.c*** 或 ***stm32f10x_ppp.h*** 文件，PPP 表示外设名称。

如针对模数转换(ADC)外设，在 ***src*** 文件夹下有一个 ***stm32f10x_adc.c*** 源文件，在 ***inc*** 文件夹下有一个 ***stm32f10x_adc.h*** 头文件，若我们开发的工程中用到了 STM32 内部的 ADC，则至少要把这两个文件包含到工程里。见图 4-5。

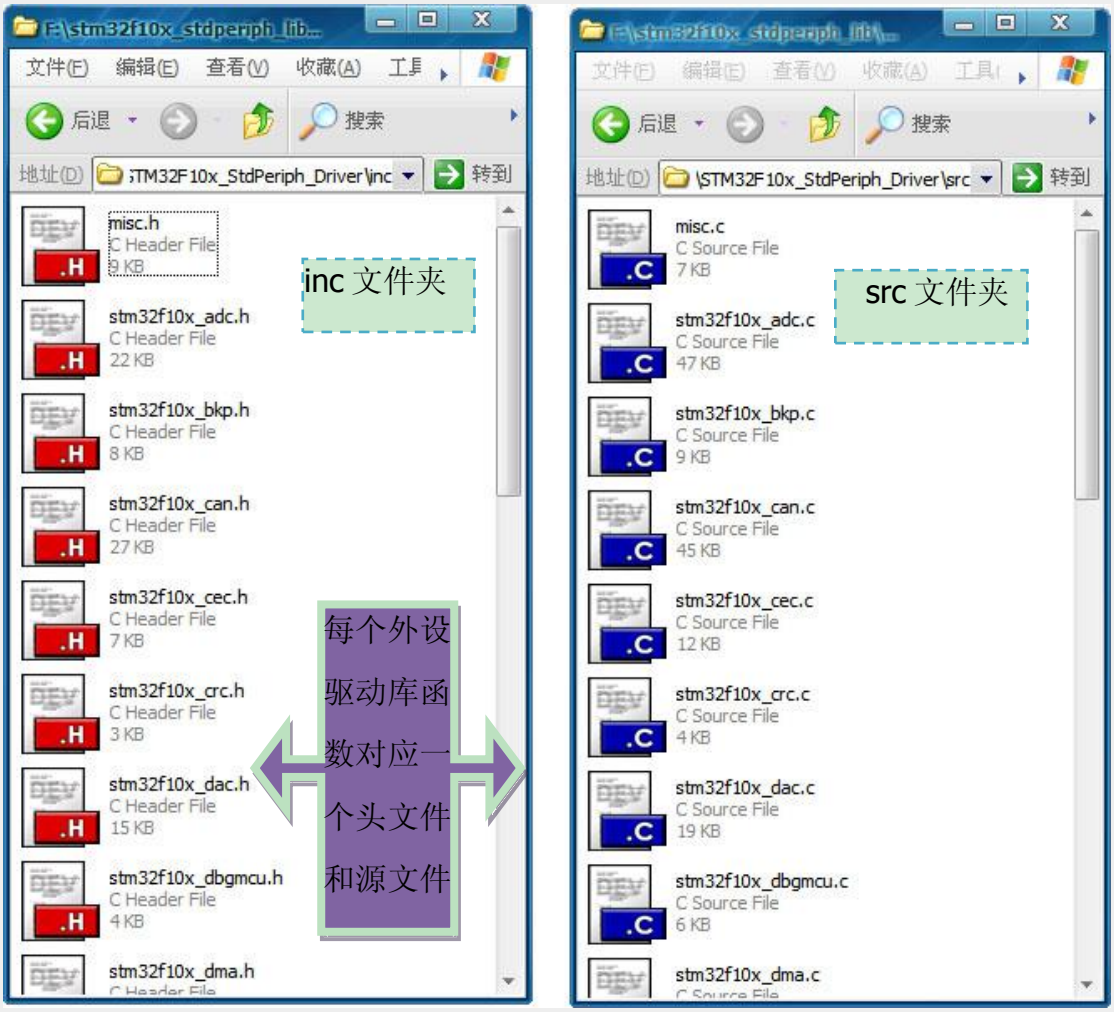


图 4-5

这两个文件夹中，还有一个很特别的 ***misc.c*** 文件，这个文件提供了外设对内核中的 NVIC(中断向量控制器)的访问函数，在配置中断时，我们必须把这个文件添加到工程中。

4.2.2.6 stm32f10x_it.c、 stm32f10x_conf.h 文件

在库目录的 ***|Project|STM32F10x_StdPeriph_Template*** 目录下，存放了官方的一个库工程模板，我们在用库建立一个完整的工程时，还需要添加这个目录下的 ***stm32f10x_it.c***、***stm32f10x_it.h***、***stm32f10x_conf.h*** 这三个文件。

stm32f10x_it.c，是专门用来编写中断服务函数的，在我们修改前，这个文件已经定义了一些*系统异常* 的接口，其它普通中断服务函数由我们自己添

加。但是我们怎么知道这些中断服务函数的接口如何写呢？是不是可以自定义呢？答案当然不是的，这些都有可以在汇编启动文件中找到，具体的大家自个看库的启动文件的源码去吧。

stm32f10x_conf.h，这个文件被包含进 ***stm32f10x.h*** 文件。是用来配置使用了什么外设的头文件，用这个头文件我们可以很方便地增加或删除上面 ***driver*** 目录下的外设驱动函数库。如下面的代码配置表示使用了 gpio、rcc、spi、usart 的外设库函数，其它的注释掉的部分，表示没有用到。

```
1. /* Includes ------------------------------------------------------------
   --------*/
2. /* Uncomment/Comment the line below to enable/disable peripheral heade
   r file inclusion */
3. //#include "stm32f10x_adc.h"
4. //#include "stm32f10x_bkp.h"
5. //#include "stm32f10x_can.h"
6. //#include "stm32f10x_cec.h"
7. //#include "stm32f10x_crc.h"
8. //#include "stm32f10x_dac.h"
9. //#include "stm32f10x_dbgmcu.h"
10. //#include "stm32f10x_dma.h"
11. //#include "stm32f10x_exti.h"
12. //#include "stm32f10x_flash.h"
13. //#include "stm32f10x_fsmc.h"
14. #include "stm32f10x_gpio.h"
15. //#include "stm32f10x_i2c.h"
16. //#include "stm32f10x_iwdg.h"
17. //#include "stm32f10x_pwr.h"
18. #include "stm32f10x_rcc.h"
19. //#include "stm32f10x_rtc.h"
20. //#include "stm32f10x_sdio.h"
21. #include "stm32f10x_spi.h"
22. //#include "stm32f10x_tim.h"
23. #include "stm32f10x_usart.h"
24. //#include "stm32f10x_wwdg.h"
25. //#include "misc.h" /* High level functions for NVIC and SysTick (add-
   on to CMSIS functions) */
```

stm32f10x_conf.h 这个文件还可配置是否使用 *“断言”* 编译选项，在开发时使用断言可由编译器检查库函数传入的参数是否正确，软件编写成功后，去掉 *“断言”* 编译选项可使程序全速运行。可通过定义 ***USE_FULL_ASSERT*** 或取消定义来配置是否使用断言。

4.2.3 库各文件间的关系

前面向大家简单介绍了各个库文件的作用，库文件是直接包含进工程即可，丝毫不用修改，而有的文件就要我们在使用的时候根据具体的需要进行配

置。接下来从整体上把握一下各个文件在库工程中的层次或关系，这些文件对应到 CMSIS 标准架构上。见图 0-6

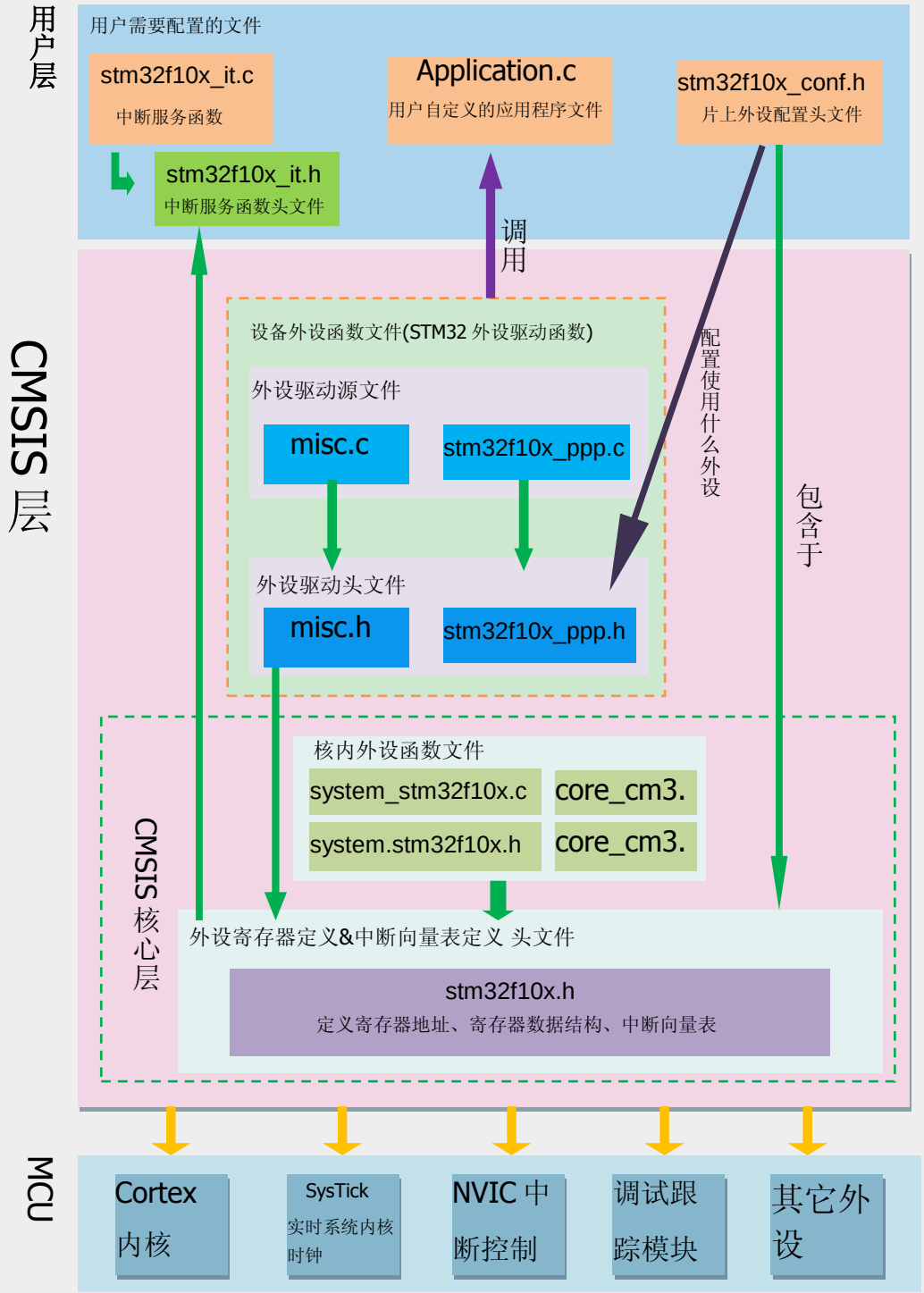


图 0-6 描述了 STM32 库各文件之间的调用关系，这个图省略了 DSP 核(Cortex-M3 没有 DSP 核)和实时系统层部分的文件关系。在实际的使用库开发工程的过程中，我们把位于 CMSIS 层的文件包含进工程，丝毫不用修改，也不建议修改。

图 0-6（STM32 库文件结构）

对于位于用户层的几个文件，就是我们在使用库的时候，针对不同的应用对库文件进行增删（用条件编译的方法增删）和改动的文件。

4.2.4 使用库帮助文档

野火坚信，授之以鱼不如授之以渔。官方资料是所有关于 STM32 知识的源头，所以在本小节介绍如何使用官方资料。官方的帮助手册，是最好的教程，几乎包含了所有在开发过程中遇到的问题。这些资料已整理到了附录光盘。

4.2.4.1 常用官方资料

1. 《stm32f10x_stdperiph_lib_um.chm》

   这个就是前面提到的库的帮助文档，在使用库函数时，我们最好通过查阅此文件来了解库函数原型，或库函数的*调用*的方法。也可以直接阅读源码里面的函数的函数说明。

2. 《STM32 参考手册.pdf 》

   这个文件相当于 STM32 的 datasheet，它把 STM32 的时钟、存储器架构、及各种外设、寄存器都描述得清清楚楚。当我们对 STM32 的库函数的*实现方式*感到困惑时，可查阅这个文件，以直接配置寄存器方式开发的话查阅这个文档的频率会更高。但你会累死。

3. 《Cortex-M3 权威指南》 宋岩译。

   该手册详细讲解了 Cortex 内核的架构和特性，要深入了解 Cortex-M3 内核，这是首选，经典中的经典呀。

4. 当然还有其他很有用的官方文档，这里就不再赘述……

4.4.2.2 初识库函数

所谓库函数，就是 STM32 的库文件中为我们编写好的函数接口，我们只要调用这些库函数，就可以对 STM32 进行配置，达到控制目的。我们可以不知道库

函数是如何实现的，但我们调用函数必须要知道*函数的功能*、*可传入的参数及其意义*、和*函数的返回值*。

于是，有读者就问那么多函数我怎么记呀？野火的回答是：会查就行了，哪个人记得了那么多。所以我们学会查阅*库帮助文档*是很有必要的。

打开库帮助文档 ***stm32f10x_stdperiph_lib_um.chm*** 见图 0-7

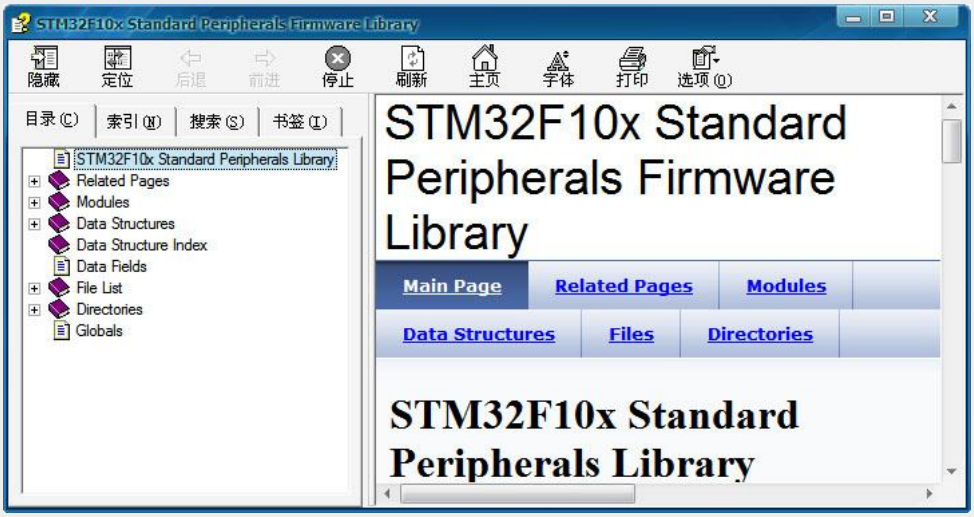


图 0-7

层层打开文档的目录标签 Modules\STM32F10x_StdPeriph_Driver，可看到

STM32F10x_StdPeriph_Driver 标签下有很多外设驱动文件的名字 MISC、ADC、BKP、CAN 等标签。我们试着查看 ADC 的*初始化库函数(ADC_Init)*看看，继续打开标签\ADC\ADC_Exported_Functions\Functions\ADC_Init 见图

0-8

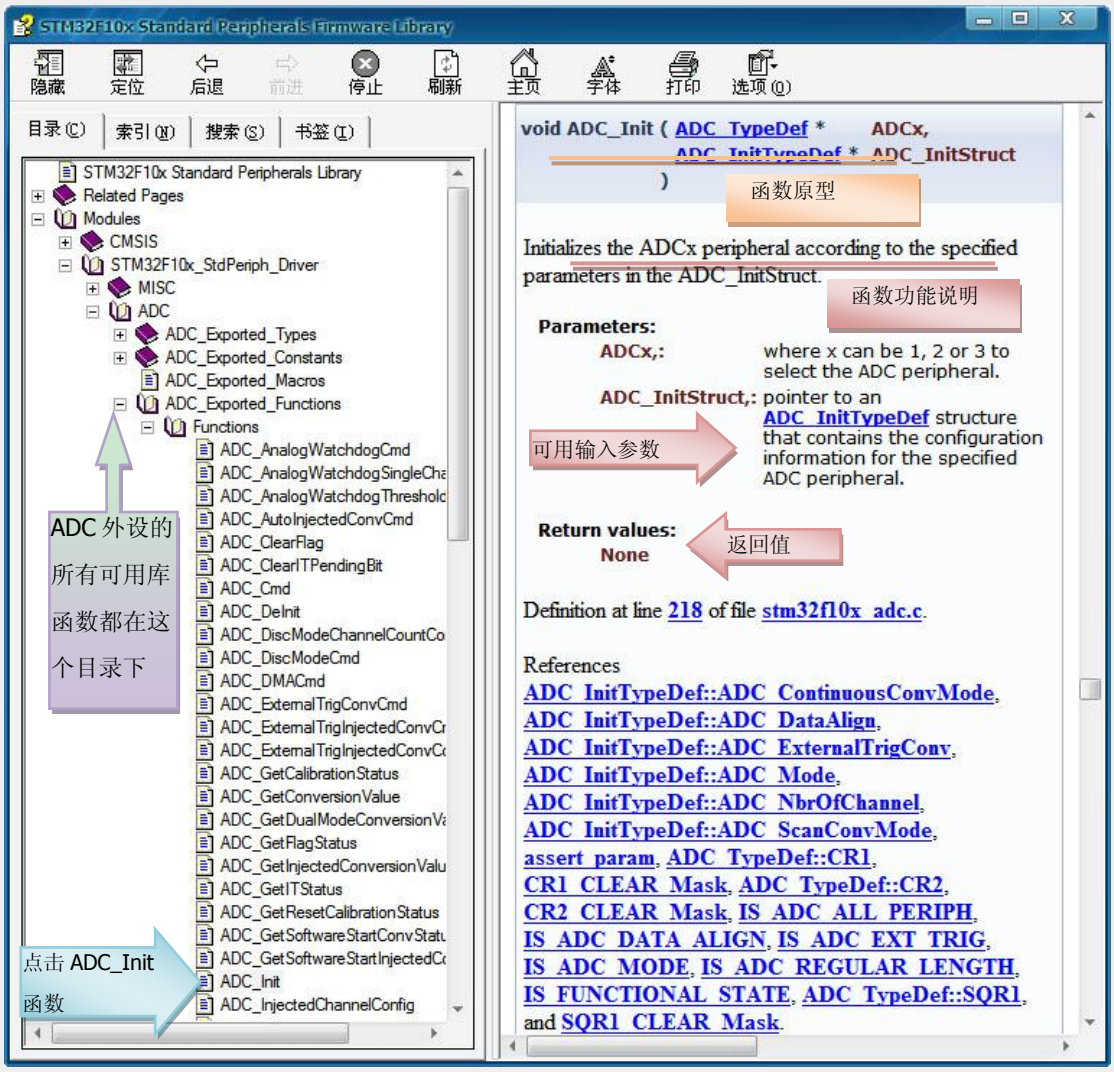


图 0-8

利用这个文档，我们即使没有去看它的具体代码，也知道要怎么利用它了。

如它的功能是：以 ADC_InitStruct 参数配置 ADC，进行初始化。原型为 void ADC_Init(***ADC_TypeDef*** * ADCx , ***ADC_Init_TypeDef*** * ADC_InitStruct)

其中输入的参数 ADCx 和 ADC_InitSturct 均为库文档中定义的 *自定义数据类型*，这两个传入参数均为结构体指针。初学时，我们并不知道如 ***ADC_TypeDef*** 这样的类型是什么意思，可以点击函数原型中带下划线的 ***ADC_TypeDef*** 就可以查看这是什么类型了。

就这样初步了解了一下库函数，读者就可以发现 STM32 的库是写得很优美的。每个函数和数据类型都符合 *见名知义* 的原则，当然，这样的名称写起来特

别长，而且对于我们来说要输入这么长的英文，很容易出错，所以在开发软件的时候，在用到库函数的地方，直接把*库帮助文档*中函数名称复制粘贴到工程文件就可以了。