#### 官方固件库说明（可以不看）

##### STM32官方库.c/.h文件功能（汇总）

<https://blog.csdn.net/Chrstian/article/details/123945627>

##### STM32 官方标准固件库简介

**先了解51单片机**

很多用户都是从学 51 单片机开发转而想进一步学习 STM32 开发，他们习惯了 51 单片机的寄存器开发方式，突然一个 ST 官方库摆在面前会一头雾水，不知道从何下手。下面我们将通过一个简单的例子来告诉 STM32 固件库到底是什么，和寄存器开发有什么关系？

其实一句话就可以概括：**固件库就是函数的集合，固件库函数的作用是向下负责与寄存器直接打交道，向上提供用户函数调用的接口（API）**。

在 51 的开发中我们常常的作法是直接操作寄存器，比如要控制某些 IO 口的状态，我们直接操作寄存器：

P0=0x11;

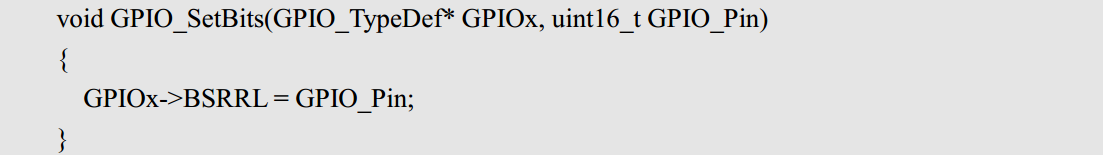
而在 STM32 的开发中，我们同样可以操作寄存器：

GPIOF->BSRRL=0x0001; //这里是针对 STM32F4 系列

这种方法当然可以，但是这种方法的劣势是你需要去掌握每个寄存器的用法，你才能正确使用STM32，而对于 STM32 这种级别的 MCU，数百个寄存器记起来又是谈何容易。

于是 ST(意法半导体)推出了官方固件库，固件库将这些寄存器底层操作都封装起来，提供一整套接口（API）供开发者调用，**大多数场合下，你不需要去知道操作的是哪个寄存器，你只需要知道调用哪些函数即可**。

比如上面的控制 BSRRL 寄存器实现电平控制，官方库封装了一个函数：



这个时候你不需要再直接去操作 BSRRL 寄存器了，你只需要知道怎么使用 GPIO\_SetBits ()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， 了解 STM32 各个外设的运行机制。只有了解了这些原理，你在进行固件库开发过程中才可能得心应手游刃有余。只有了解了原理，你才能做到“知其然知其所以然”，所以大家在学习库函数的同时，别忘了要了解一下寄存器大致配置过程。

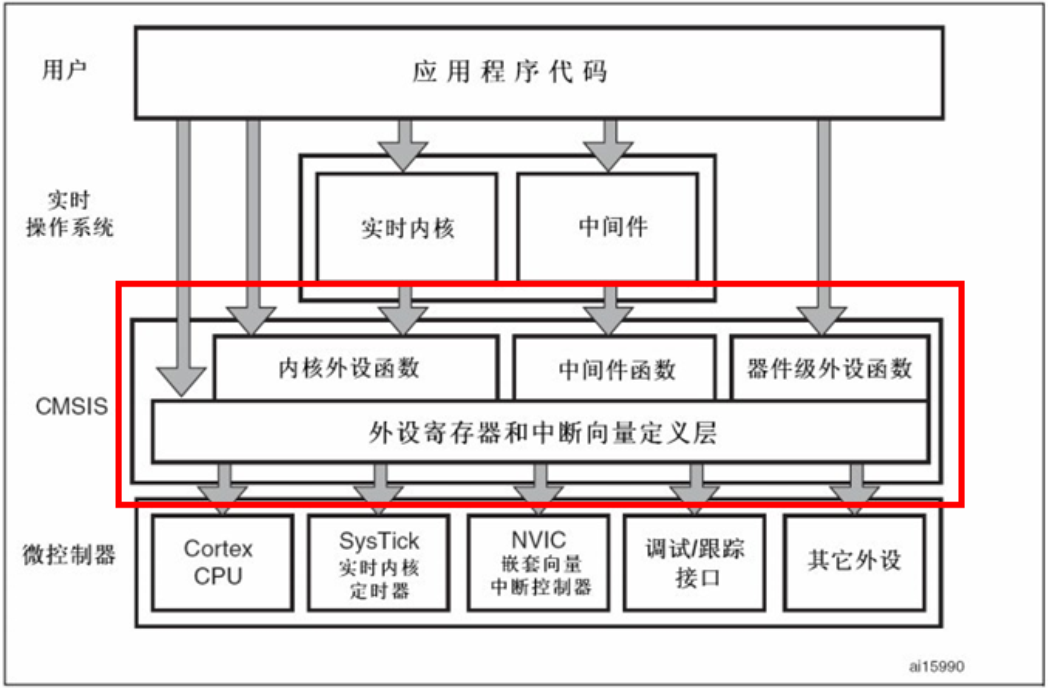
可以先学习51单片机那种简单的寄存器操作再来学习32的寄存器

##### STM32 固件库与 CMSIS 标准

前一节我们讲到， STM32F4 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。 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-M4 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-M4 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-M4 内核芯片的片上外设也会有很大的区别，比如 STM32F407 和 STM32F429，他们的片上外设就有很大的区别。

（所以Cortex-M4是一种芯片内核的架构，一种标准）

既然大家都使用的是 Cortex-M4 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-M4 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 应用程序基本结构如下图



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**。由ARM 提供，芯片厂商根据需要更新。  
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**。

**这些都是不同的芯片（相同的架构）通用的**

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。

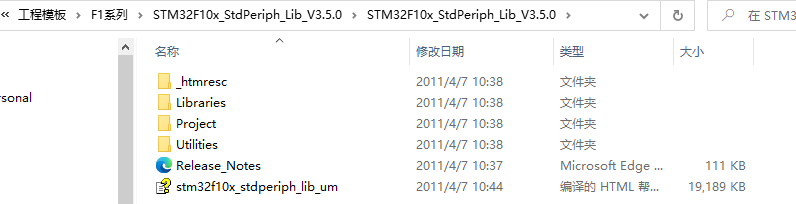
至于 CMSIS 的具体内容就不必多讲了，需要了解详细的朋友可以到网上搜索资料，相关资料可谓满天飞。

**所以懂了一种，其他架构相同的芯片就可以直接拿来用。**

**这就是标准化的好处。**

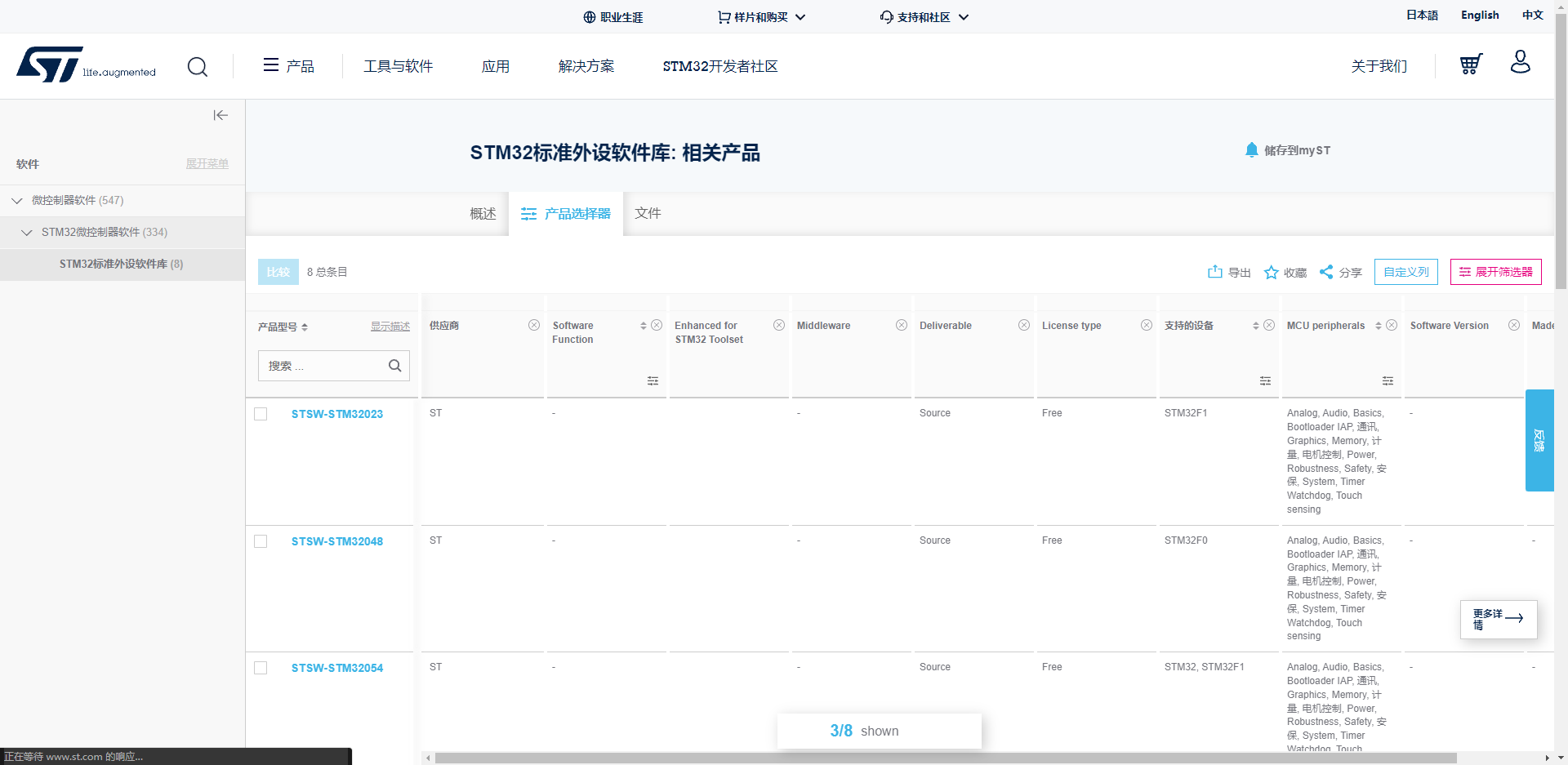
##### STM32F1 官方库包介绍

这一节内容主要讲解 ST 官方提供的 STM32F1 固件库包的结构。 ST 官方提供的固件库完整包可以在官方网站下载。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 V3.5.0版本的固件库



###### 官网标准库下载地址

<https://www.st.com/content/st_com/zh/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32-standard-peripheral-libraries/products.html>



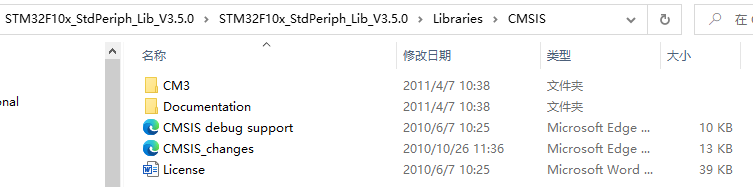
###### 文件夹介绍：

**Libraries**

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（库函数）**的所有子文件夹和文件。

* CMSIS 文件夹

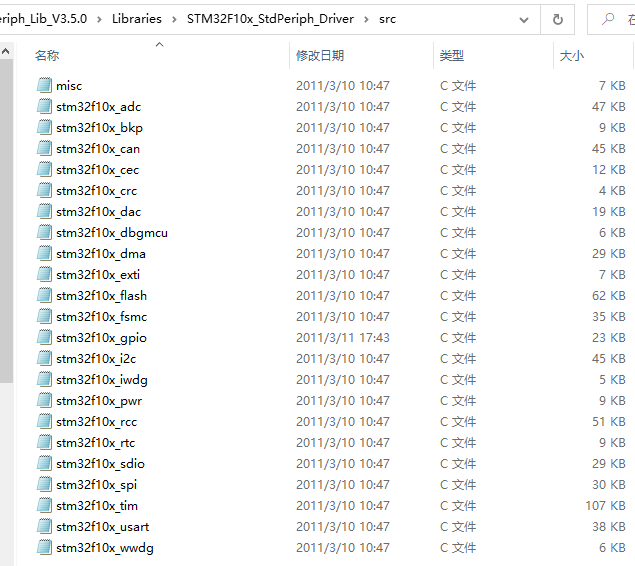
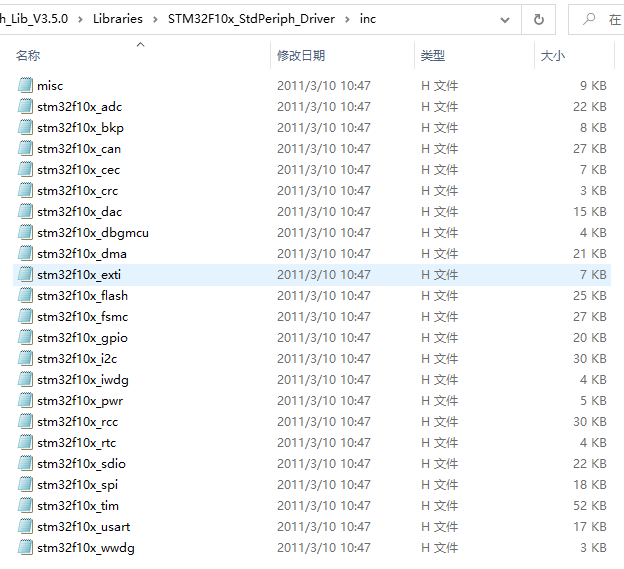
存放的是符合 CMSIS 规范的一些文件。包括 STM32F10x 核内外设访问层代码，， 以及 STM32F10x 片上外设访问层代码等。我们后面新建工程的时候会从这个文件夹复制一些文件到我们工程。



* STM32F10x\_StdPeriph\_Driver 文件夹

放的是 STM32F10x 标准外设固件库源码文件和对应的头文件。inc 目录存放的是 stm32f10x\_???.h 头文件,无需改动。 src 目录下面放的是 stm32f10x\_???.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外设的关键文件，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**

**Project 是官方提供的工程实例和模板**。下面有两个文件夹。

* STM32F10x\_StdPeriph\_Examples 文件夹

下面存放的的 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。（官方源码的注释较少）

**Utilities**

Utilities 文件下就是官方评估板的一些对应源码，这个对于本手册学习可以忽略不看。

**Release\_Notes.html**

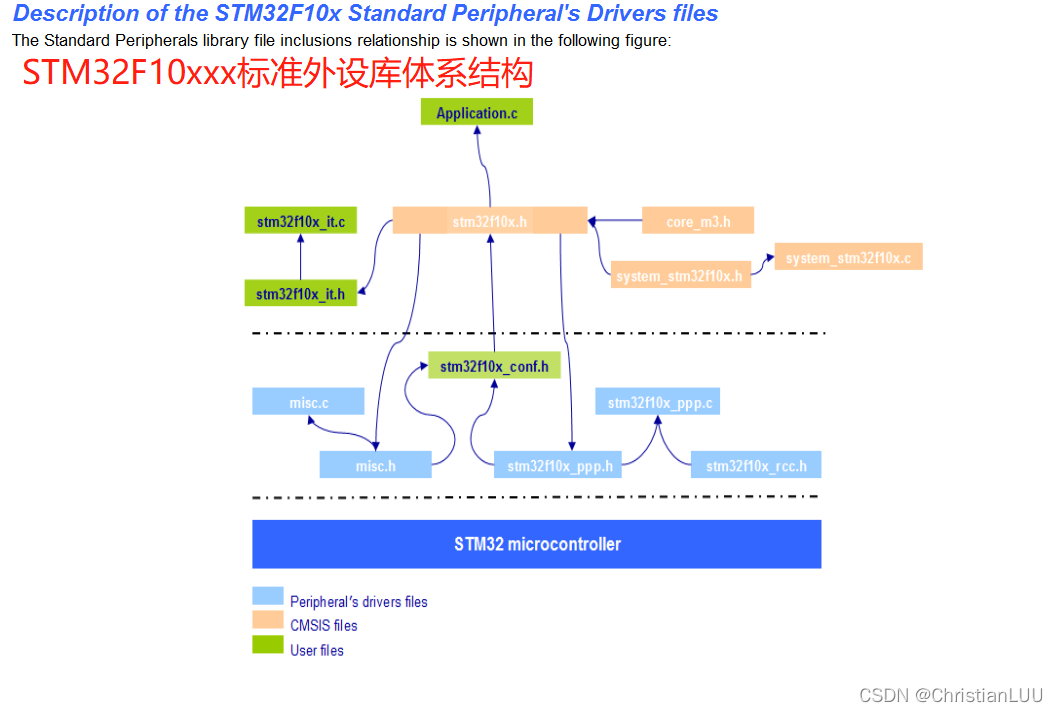
发布文档：版本说明

**stm32f10x\_stdperiph\_lib\_um.chm**

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。

###### 关键文件介绍：

在介绍一些关键文件之前，首先我们来看看一个基于固件库的 STM32F10x工程需要哪些关键文件，这些文件之间有哪些关联关系。其实这个可以从 ST 提供的英文版的 STM32F10x 固件库说明里面找到。这些文件它们之间的关系如下图



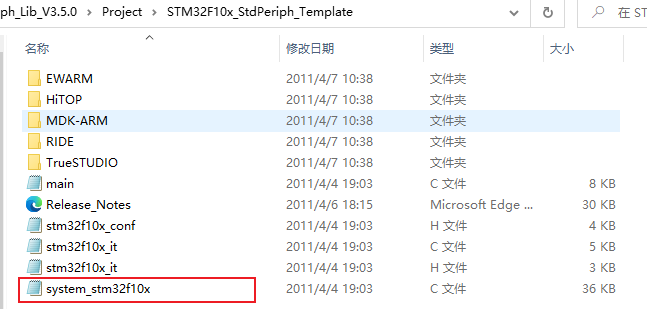
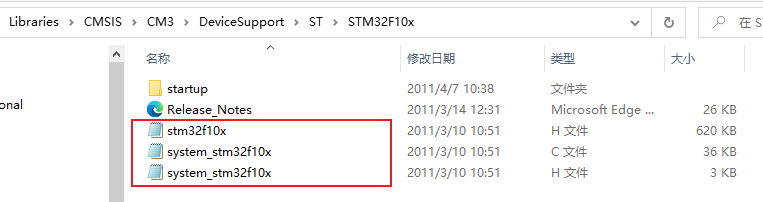
**stm32f10x.h**

**system\_stm32f10x.h ----- system\_stm32f10x.c 时钟系统**

stm32f10x.h和system\_stm32f10x.h、system\_stm32f10x.c文件存放在文件夹

\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x下面。

此外，system\_stm32f10x.c 在目录  
\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Project\STM32F10x\_StdPeriph\_Template中也可以找到。



system\_stm32f10x.h 是片上外设接入层系统头文件。主要是申明设置系统及总线时钟相关的函数。这个里面有一个非常重要的 SystemInit()函数申明，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系统和总线时钟。

stm32f10x.h 是 STM32F4 片上外设访问层头文件。 这个文件就相当重要了，只要你做  
STM32F10x 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“4.6 MDK 中寄存器地址名称映射分析”中会讲到。 同时该文件还包含了一些时钟相关的定义， FPU 和 MPU 单元开启定义，中断相关定义等等。

**core\_cm3.h**

core\_cm3.h 文件位于

STM32F10x\_StdPeriph\_Lib\_V3.5.0\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。

**stm32f10x\_it.c 外部中断服务函数 ----- stm32f10x\_it.h**

**stm32f10x\_conf.h---外设驱动配置文件**

stm32f10x\_it.c,stm32f10x\_it.h 以 及 stm32f10x\_conf.h 等 文 件 ， 我 们 可 以 从  
\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Project\STM32F10x\_StdPeriph\_Template文件夹中找到。这几个文件我们后面新建工程也有用到。 stm32f10x\_it.c 和 stm32f10x\_it.h 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，（正点原子觉得这个文件没太大意义）。

stm32f10x\_conf.h 是外设驱动配置文件。 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。

**misc.c -----misc.h**

**stm32f10x\_???.c --------- stm32f10x\_???.h**

**stm32f10x\_rcc.c ---------stm32f10x\_rcc.h**

对于上图中的 misc.c,misc.h,stm32f10x\_ppp.c,stm32f10x\_ppp.h 以及 stm32f10x\_rcc.c 和stm32f10x\_rcc.h 文件，这些文件存放在目录 \Libraries\STM32F10x\_StdPeriph\_Driver。这些文件是 STM32F10x标准的外设库文件。其中 misc.c 和 misc.h 是定义中断优先级分组以及 Systick 定时器相关的函数。 stm32f10x\_rcc.c 和 stm32f10x\_rcc.h 是与 RCC 相关的一些操作函数，作用主要是一些时钟的配置和使能。在任何一个 STM32 工程 RCC 相关的源文件和头文件是必须添加的。

对于文件 stm32f10x\_???.c 和 stm32f10x\_???.h，这就是 stm32F10x 标准外设固件库对应的源文件和头文件。包括一些常用外设 GPIO,ADC,USART 等。

**Application.c**

文件 Application.c 实际就是说是应用层代码。这个文件名称可以任意取了。 我们工程中，直接取名为 main.c。

\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Project\STM32F10x\_StdPeriph\_Template

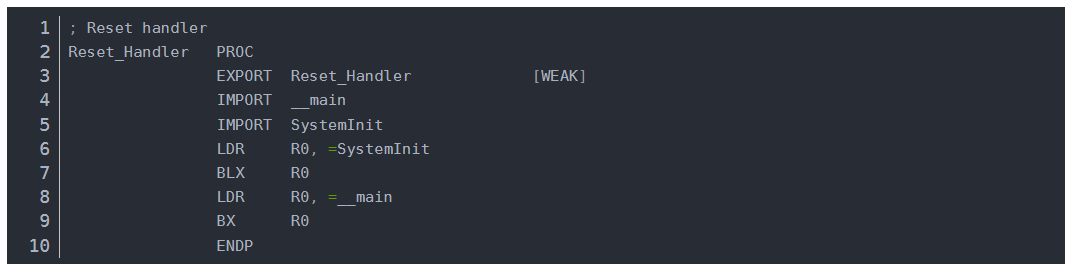
**补充**

实际上一个完整的 STM32F10x 的工程光有上面这些文件还是不够的。还缺少非常关键的启动文件。 STM32F10x 的启动文件存放在目录

\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm 下面。

对于不同型号的 STM32F10x 系列对应的启动文件也不一样。

我们的开发板是 STM32F10x 系列所以我们选择的启动文件为startup\_stm32f10x\_md.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：



这段代码的作用是在系统复位之后引导进入 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。  
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板。

#### 新建MDK5工程模板（基于STM32F10x固件库）

首先我们要准备如下资料：

V3.5.0 固件库包：STM32F10x\_StdPeriph\_Lib\_V3.5.0 这是 ST 官网下载的固件库完整版

##### Keil的工程配置选项说明

<https://blog.csdn.net/weixin_45992045/article/details/124431806>

##### （总结）新建模板过程

###### 1.新建工程结构

* 建立一个文件夹Template，作为工程的根目录文件夹。然后新建 7个子文件夹：

STARTUP , CMSIS, FWLIB, HARDWARE, SYSTEM, USER, OBJ。

文件夹名字可以任取的，这样取名是为了方便识别。

STARTUP----启动文件

startup\_stm32f10x\_md.s

CMSIS------CMSIS标准文件

core\_cm3.h 内核功能定义 .h文件

core\_cm3.c 内核功能定义 .c文件

stm32f10x.h 头文件

system\_stm32f1xx.h 系统 .h文件

system\_stm32f1xx.c 系统 .c文件

FWLIB------库文件

inc文件夹 库函数头文件 misc.h tm32f10x\_???.h

src 文件夹 库函数源文件misc.c tm32f10x\_???.c

HARDWARE---外设文件

~~LED文件夹 LED.h LED.c LED~~

~~KEY文件夹 Key.h Key.c 按键控制~~

~~OLED文件夹 OLED.h OLED\_Font.h OLED.c OLED显示屏~~

~~LightSensor 文件夹 LightSensor.h LightSensor.c 光敏传感器~~

~~Buzzer 文件夹 Buzzer.h Buzzer.c 蜂鸣器~~

~~CountSensor文件夹 CountSensor.h CountSensor.c对射式红外传感器计次~~

~~Encoder文件夹 Encoder.h Encoder.c 旋转编码器计次~~

~~PWM 文件夹 PWM.h PWM.c PWM驱动~~

~~Servo 文件夹 Servo.h Servo.c 舵机~~

~~Motor 文件夹 Motor.h Motor.c 直流电机~~

~~IC 文件夹 IC.h IC.c 定时器输入捕获（IC）~~

~~AD文件夹 AD.h AD.c ADC数模转换~~

~~Serial 文件夹 Serial.h Serial.c 串口驱动~~

~~MyI2C文件夹 MyI2C.h MyI2C.c 软件MyI2C驱动~~

~~MySPI 文件夹 MySPI.h MySPI.c 软件SPI驱动~~

~~MPU6050文件夹 MPU6050.h MPU6050\_Reg.h MPU6050.c~~

~~W25Q64文件夹 W25Q64.h W25Q64 \_Ins.h W25Q64.c~~

~~删除线表示，模板中没有，可以后续添加~~

SYSTEM-----系统文件，所有项目都会用到的代码

Delay 文件夹 Delay.h Delay.c 延时函数

~~Sys 文件夹 Sys.h Sys y.c 位带操作函数~~

~~Usart 文件夹 Usart.h Usart.c 串口操作函数~~

~~删除线表示，模板中没有，可以后续添加~~

USER-------用户文件

main.c 主函数源文件

stm32f1xx\_it.c 中断.c源文件

stm32f1xx\_it.h 中断.h头文件

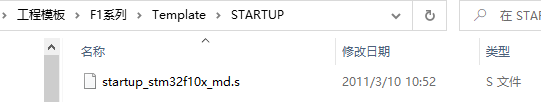
stm32f10x\_conf.h 外设驱动配置文件

OBJ-------用于存放编译过程产生的中间文件，以及hex 文件。



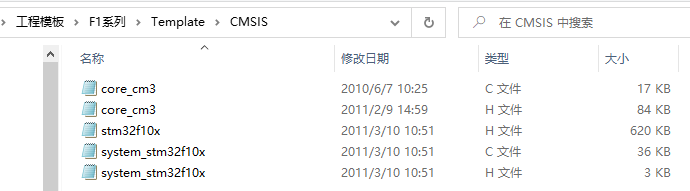
###### 2.复制官方固件库的一些文件

* STARTUP包含 1个文件



官方固件库路径：\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm

* CMSIS包含 5个文件

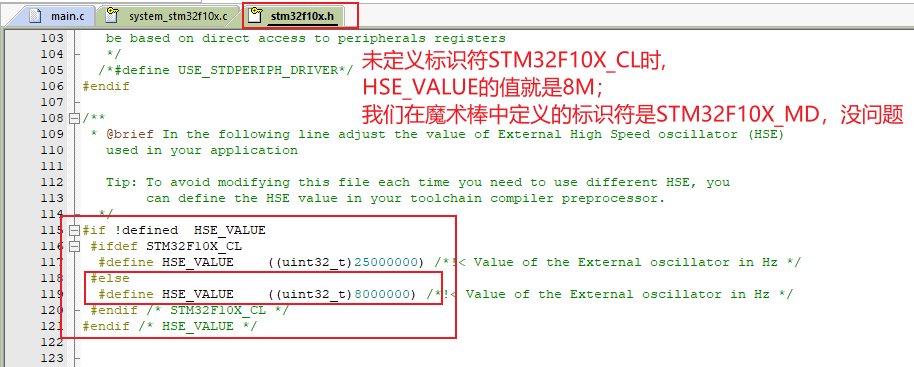


官方固件库路径1：\Libraries\CMSIS\CM3\ CoreSupport 2个文件

官方固件库路径2：\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x 3个文件

在stm32f10x.h 中

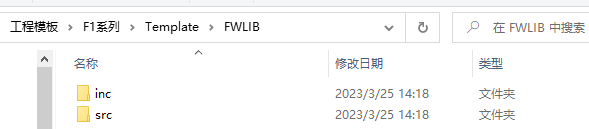
宏定义 HSE\_VALUE 的值为 8MHz，因为我们的外部高速时钟用的晶振为 8M



在system\_stm32f10x. h / c中

有一个非常重要的SystemInit( )函数申明，在系统启动时都会调用，以设置系统的整个系统和总线时钟。将STM32主频配置为72MHz。

* FWLIB包含2个文件夹

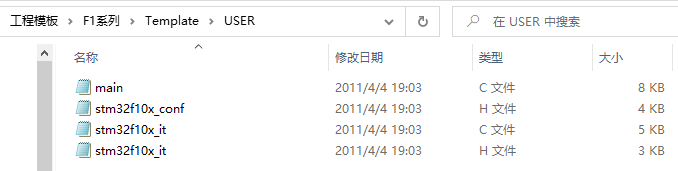


官方固件库路径：\Libraries\STM32F10x\_StdPeriph\_Driver

将目录下面的 src, inc 文件夹直接复制到我们刚才建立的 FWLib 文件夹下面。

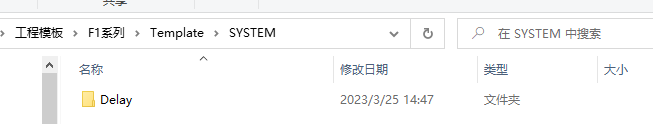
src 存放的是固件库的.c 文件，inc 存放的是对应的.h 文件，每个外设对应一个.c 文件和一个.h 头文件。

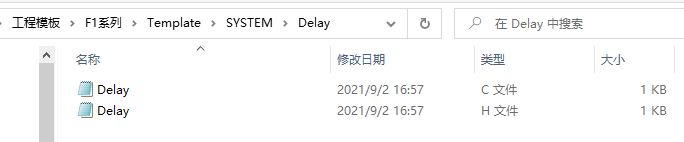
* USER包含4个文件



官方固件库路径：\Project\STM32F10x\_StdPeriph\_Template

* SYSTEM包含1~3个文件夹 Delay、Sys、Usart





来源路径：\江科大自化协\up主资料\STM32入门教程资料\程序源码\程序源码\STM32Project\1-3 Delay函数模块

Delay 文件夹 Delay.h Delay.c 延时函数

~~删除线表示，模板中没有，后续可以根据需要添加~~

~~Sys 文件夹 Sys.h Sys y.c 位带操作函数~~

~~Usart 文件夹 Usart.h Usart.c 串口操作函数~~

* HARDWARE包含？个文件夹。根据需要添加



来源路径：\江科大自化协\up主资料\STM32入门教程资料\程序源码\程序源码\STM32Project\ 里面有各种

~~删除线表示，模板中没有，后续可以根据需要添加~~

~~LED文件夹 LED.h LED.c LED~~

~~KEY文件夹 Key.h Key.c 按键控制~~

~~OLED文件夹 OLED.h OLED\_Font.h OLED.c OLED显示屏~~

~~LightSensor 文件夹 LightSensor.h LightSensor.c 光敏传感器~~

~~Buzzer 文件夹 Buzzer.h Buzzer.c 蜂鸣器~~

~~CountSensor文件夹 CountSensor.h CountSensor.c对射式红外传感器计次~~

~~Encoder文件夹 Encoder.h Encoder.c 旋转编码器计次~~

~~PWM 文件夹 PWM.h PWM.c PWM驱动~~

~~Servo 文件夹 Servo.h Servo.c 舵机~~

~~Motor 文件夹 Motor.h Motor.c 直流电机~~

~~IC 文件夹 IC.h IC.c 定时器输入捕获（IC）~~

~~AD文件夹 AD.h AD.c ADC数模转换~~

~~Serial 文件夹 Serial.h Serial.c 串口驱动~~

~~MyI2C文件夹 MyI2C.h MyI2C.c 软件MyI2C驱动~~

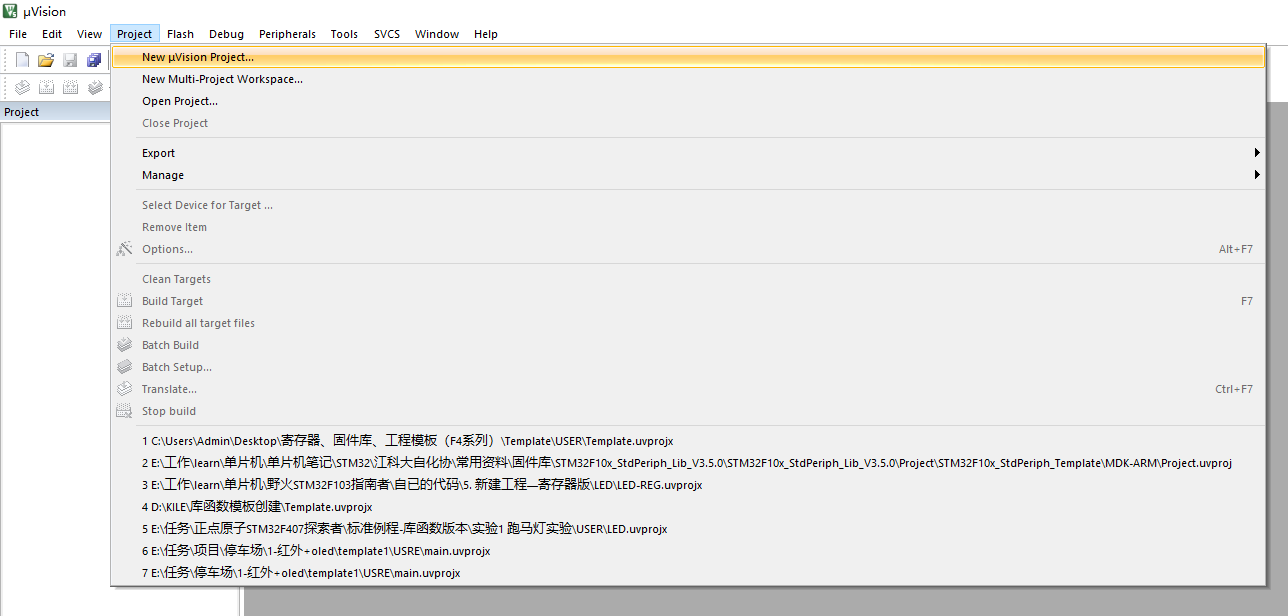
~~MySPI 文件夹 MySPI.h MySPI.c 软件SPI驱动~~

~~MPU6050文件夹 MPU6050.h MPU6050\_Reg.h MPU6050.c~~

~~W25Q64文件夹 W25Q64.h W25Q64 \_Ins.h W25Q64.c~~

###### 3.Keil建立工程

* 打开 Keil, Project–>New Uvision Project



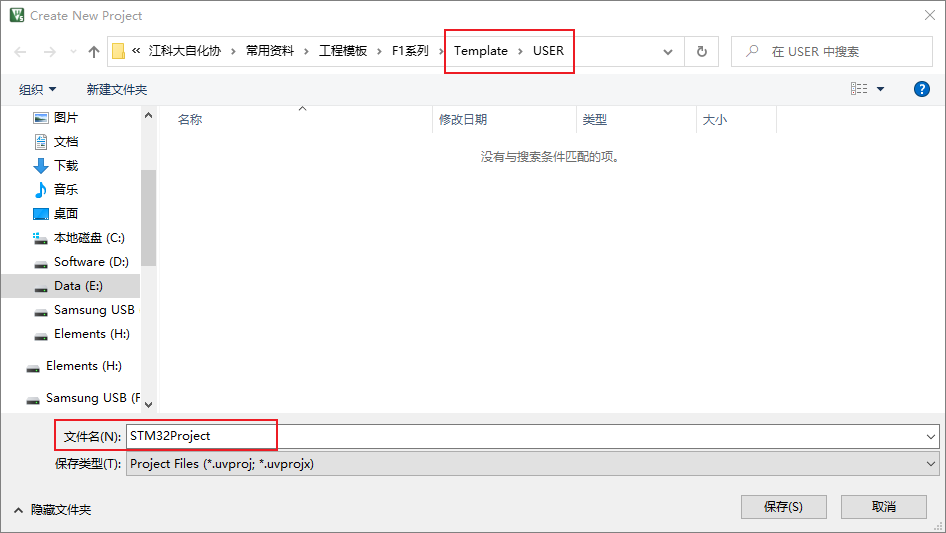
* 目录定位到工程\Template\USER文件夹，同时，工程取名之后点击保存，我们的工程文件就都保存到Template文件夹的根目录下面。 操作过程如下图

**注意：**

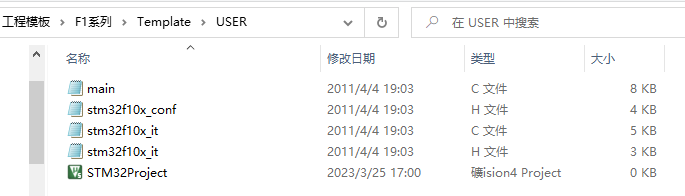
**我们需要取个通用的名称“STM32Project”。**

**因为文件夹的名称Template随便改，但是工程名称“STM32Project”难以修改。**

**以后开发时，直接复制Template文件夹，然后修改文件夹名字即可，工程名称“STM32Project”不变。**



点击保存

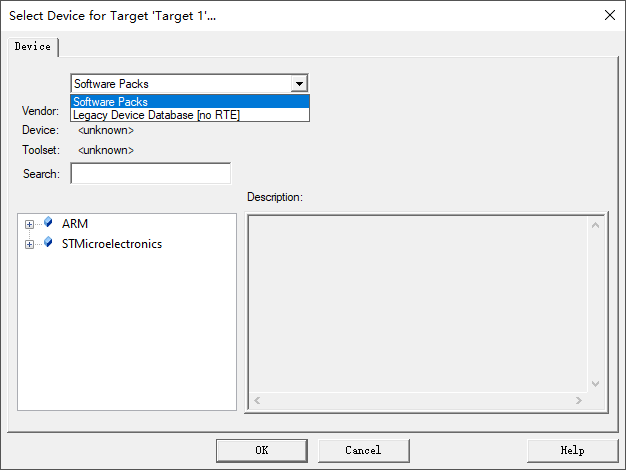


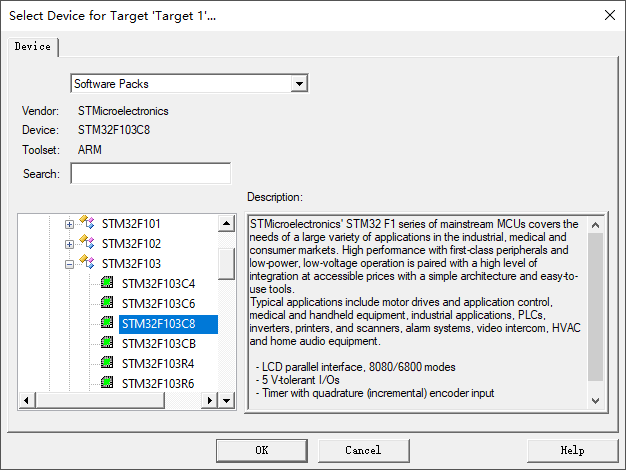
这里选USER，是为了将Template.uvprojx文件放在USER路径下

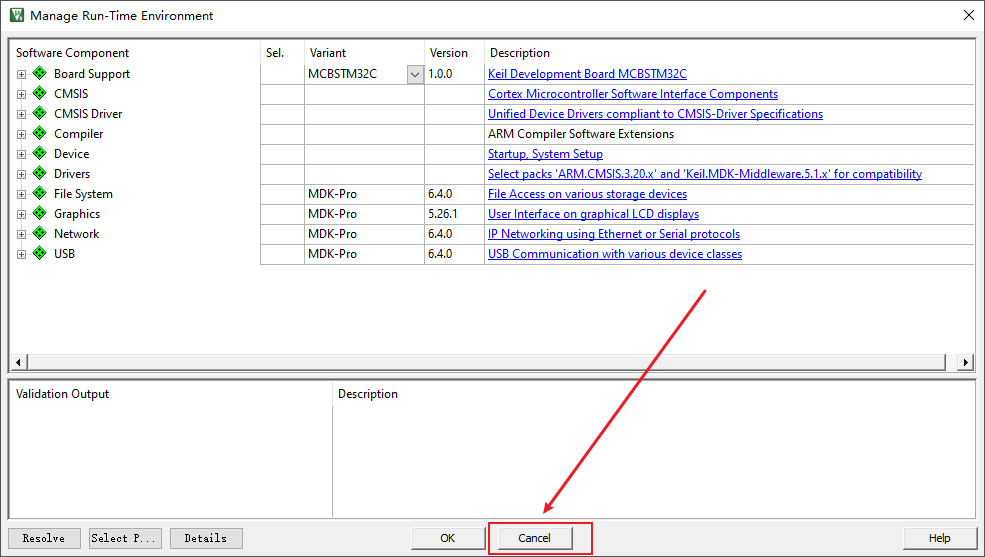
* 接下来出现选择 Device 的界面，

选择 STMicroelectronics→STM32F1 Series→STM32F103→STM32F407C8

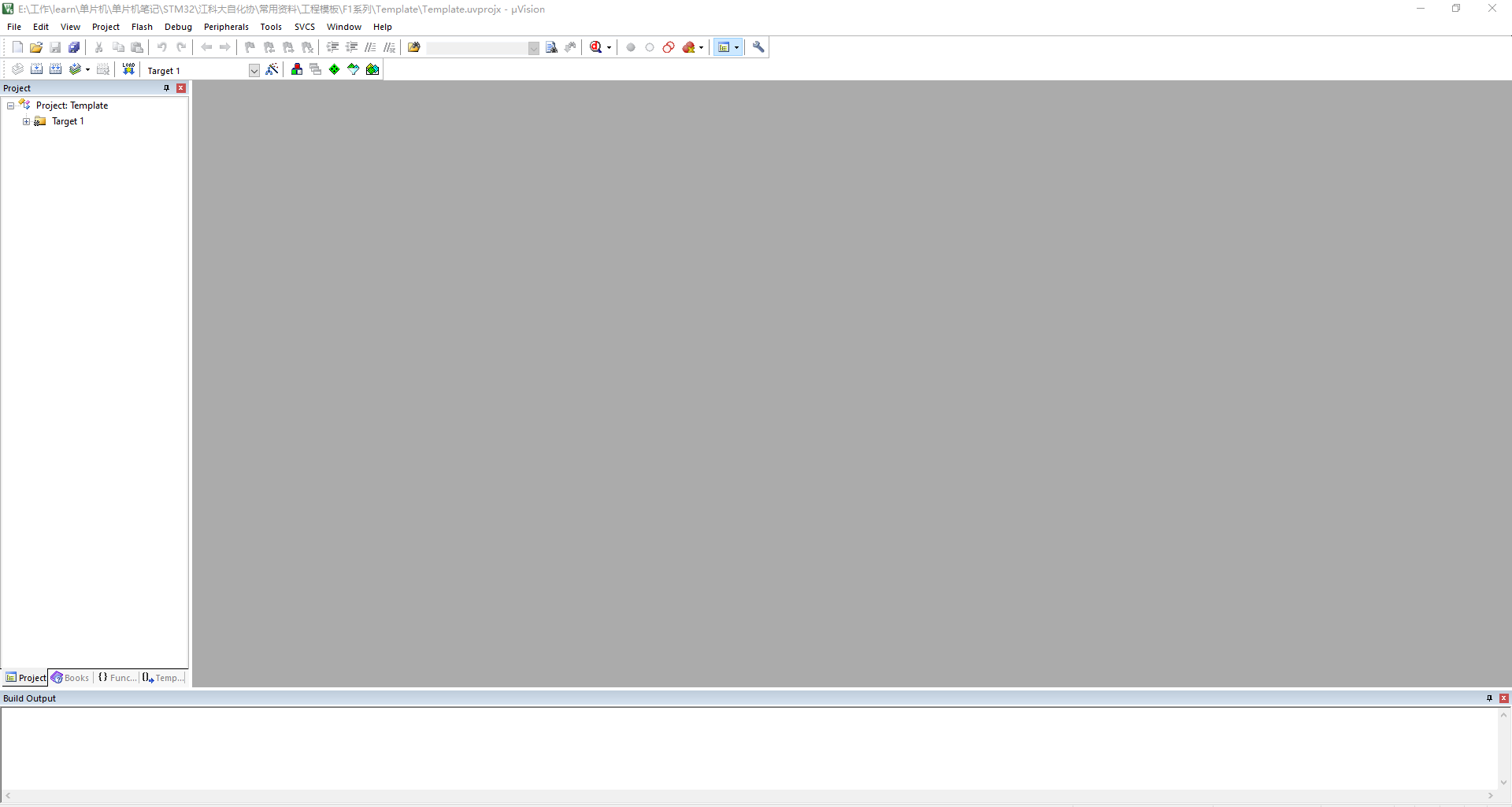
注意：如果要用51单片机，就得选择Legacy Device Database[no RTE]



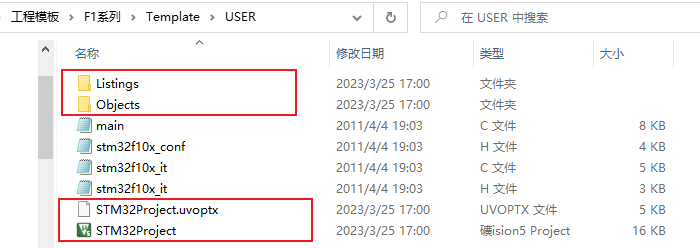


点击 OK，MDK 会弹出 Manage Run-Time Environment 对话框我们直接点击 Cancel

即可得到下图所示界面



* 现在我们看看根目录

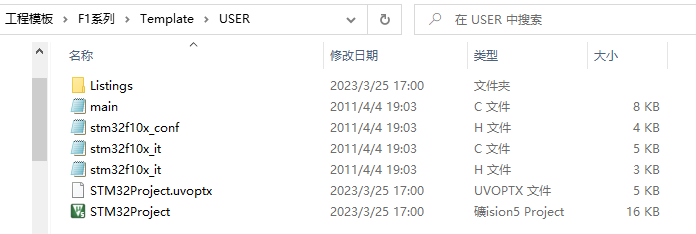


STM32Project.uvprojx 是工程文件，非常关键，不能删除。

Listings 和 Objects ：文件夹是 MDK 自动生成的文件夹，用于存放编译过程产生的中间文件。

把Objects文件夹删除。我们之前新建的OBJ 文件夹，用来存放编译中间文件。当然，我们不删除这个文件夹也是没有关系的，只是我们不用它而已。

就算我们删除Listings，在编译Build之后，Listings还会再次出现

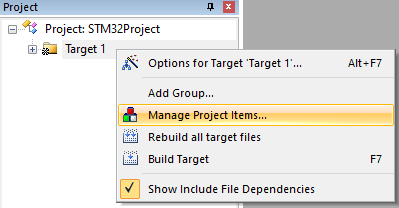


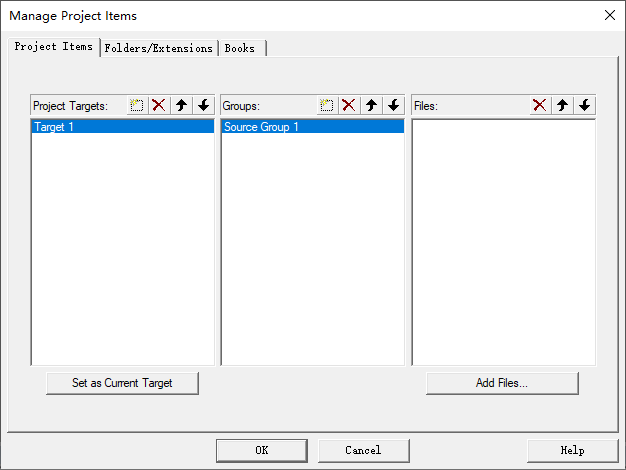
删除了Objects文件夹

###### 4.将“.c源文件”加入工程

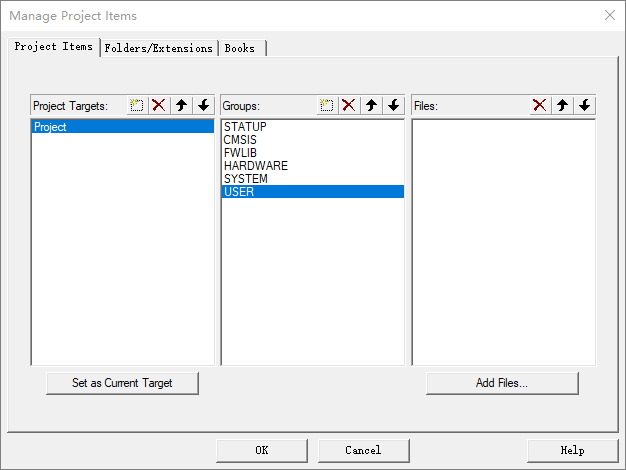
* 前面几个步骤只是将固件库相关文件复制到了工程目录下，现在需要将这四个目录下面的源文件加入到我们工程中去

右键点击 Target1，选择 Manage Project Items，如下图：

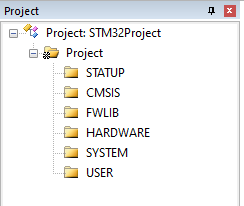




Project Targets 一栏，我们将 Target 1名字修改为Project, 然后在 Groups 一栏删掉一个 Source Group1，建立6个 Groups：STATUP、CMSIS、FWLIB、HARDWARE、SYSTEM、USER。然后点击 OK，可以看到我们的 Target名字以及 Groups 情况如下图:

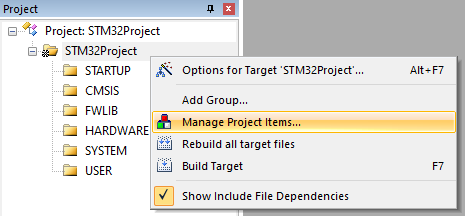


点击OK，



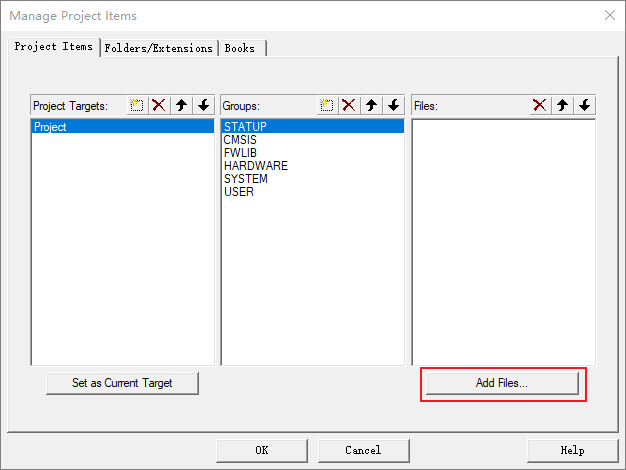
* 下面我们往 Group 里面添加我们需要的文件。

右键点击Project，选择 Manage Project Items。然后选择需要添加文件的 Group，



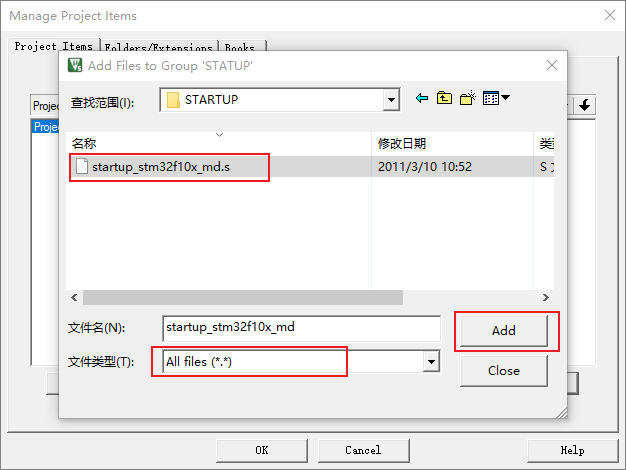
* STARTUP

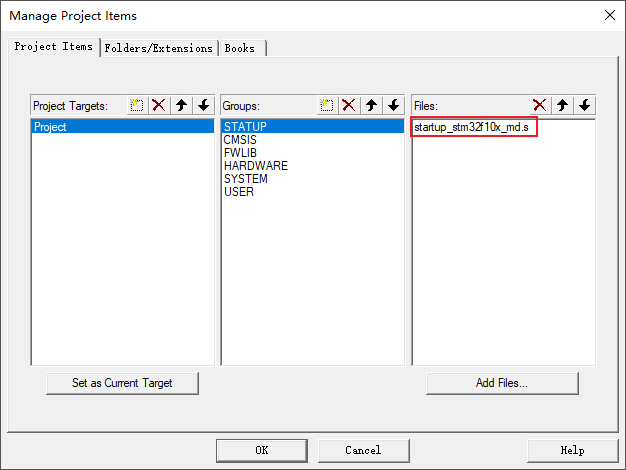
选择 STARTUP，然后点击右边的 Add Files，



定位到\STARTUP，添加文件 startup\_stm32f10x\_md.s

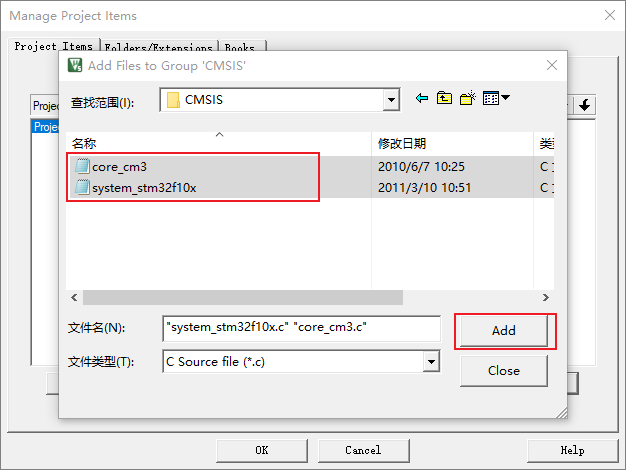
(注意，默认添加的时候文件类型为.c,也就是添加 startup\_stm32f40\_41xxx.s 启动文件的时候，你需要选择文件类型为 All files才能看得到.s文件)

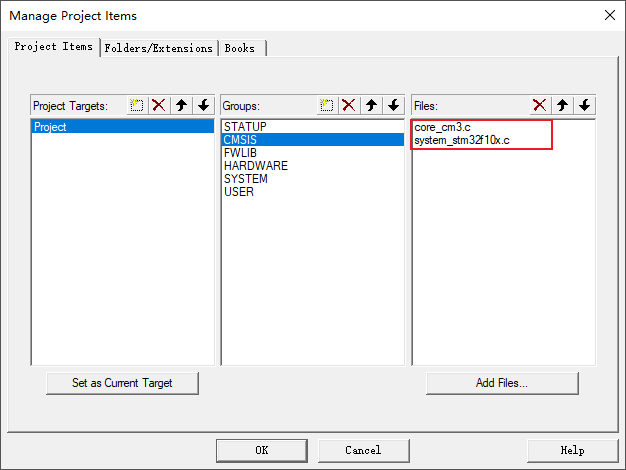




* CMSIS

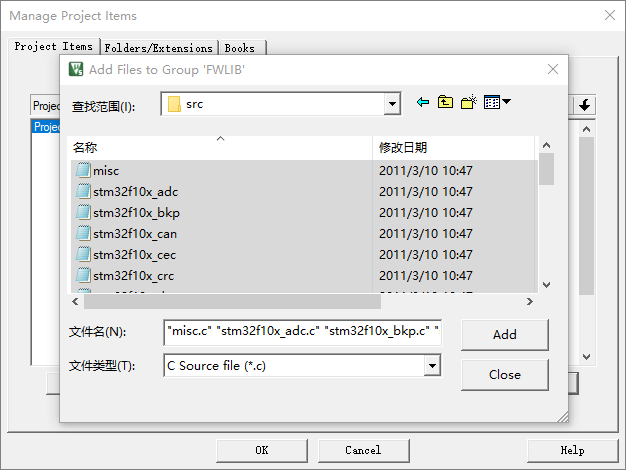
定位到\CMSIS，将里面所有的文件选中(Ctrl+A) 然后点击 Add，然后 Close.

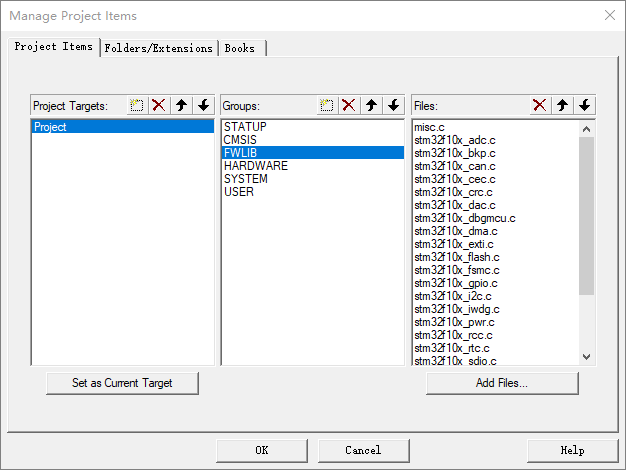




* FWLIB

定位到\FWLIB\src，将里面所有的文件选中(Ctrl+A)，然后点击 Add，然后 Close.





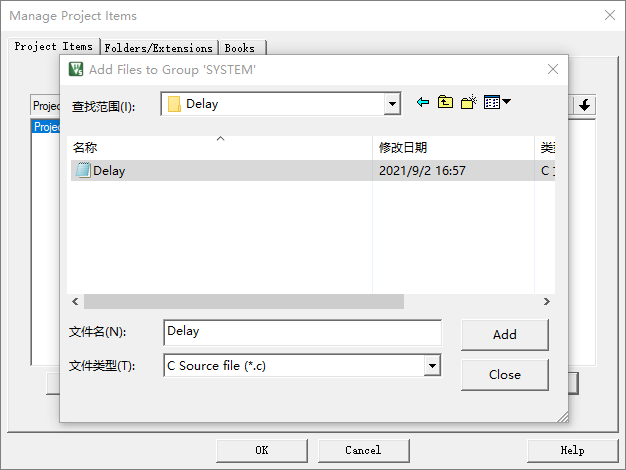
这里我们全部添加进来是为了后面方便，不用每次添加，当然这样的坏处是工程太大，当然编译起来速度慢，用户可以自行选择。

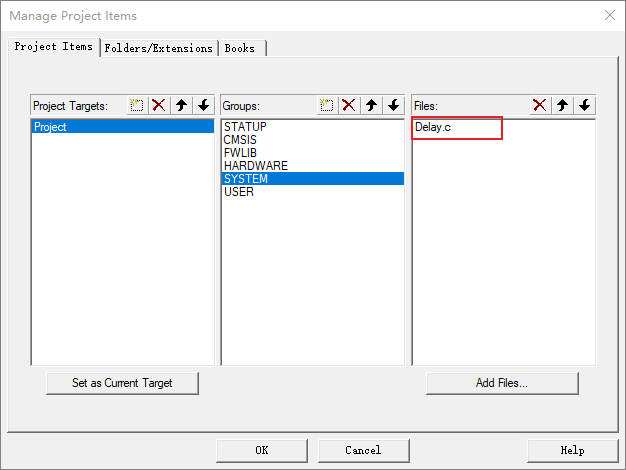
* HARDWARE

后续用到再添加，这里没有需要的

* SYSTEM

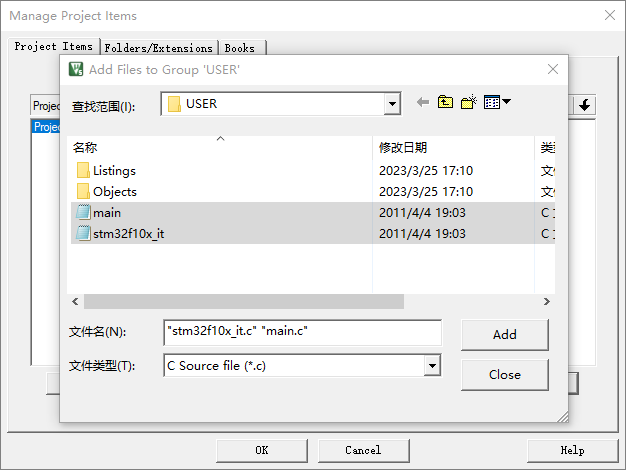
定位到\SYSTEM\Delay，将.c文件选中，然后点击 Add，然后 Close.

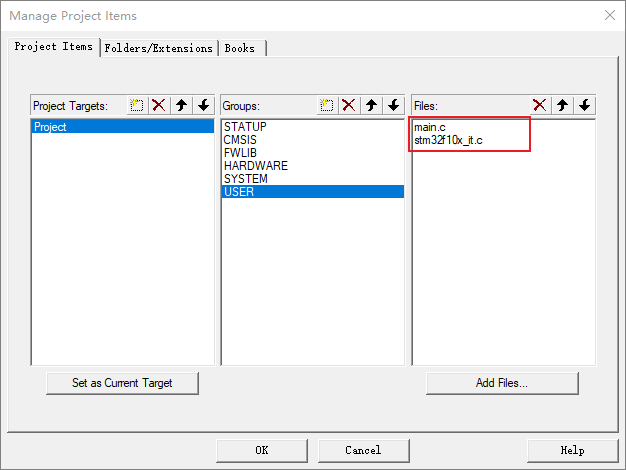




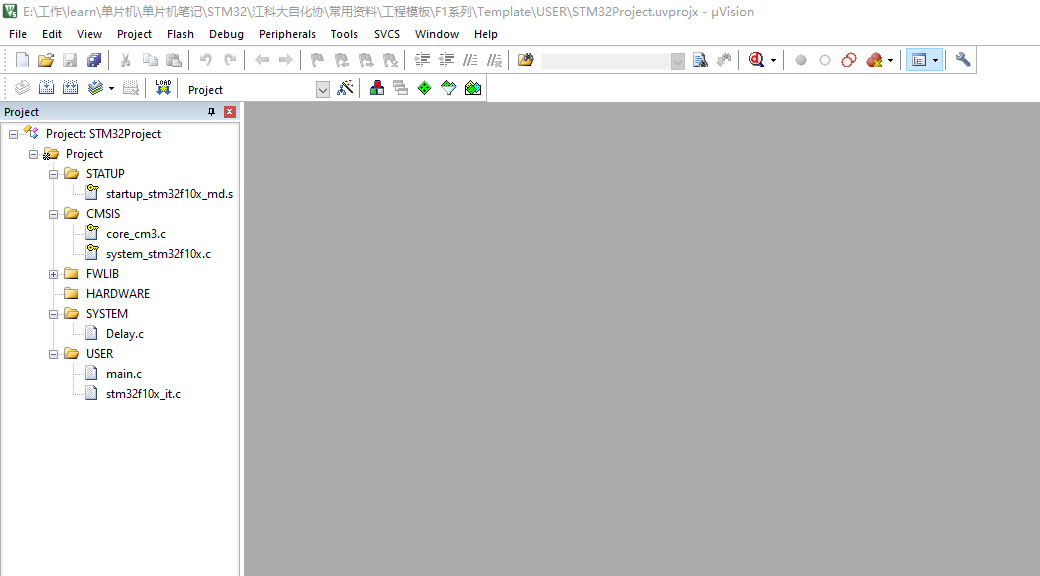
* USER

定位到\USER，将.c文件选中，然后点击 Add，然后 Close.





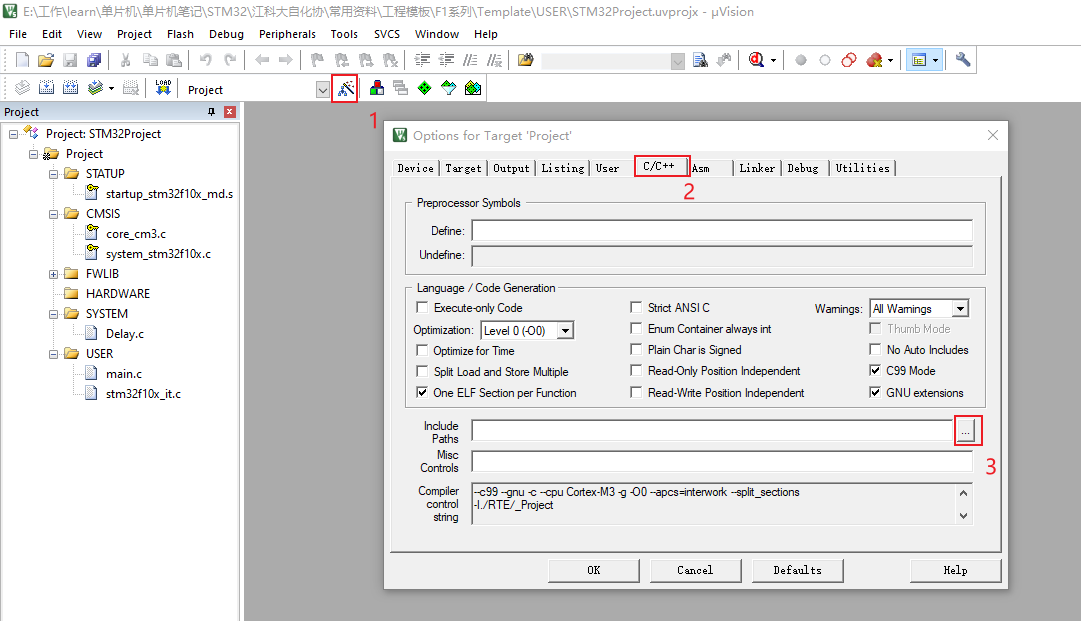
这样我们需要添加的文件已经添加到我们的工程中去了，最后点击 OK，回到工程主界面。

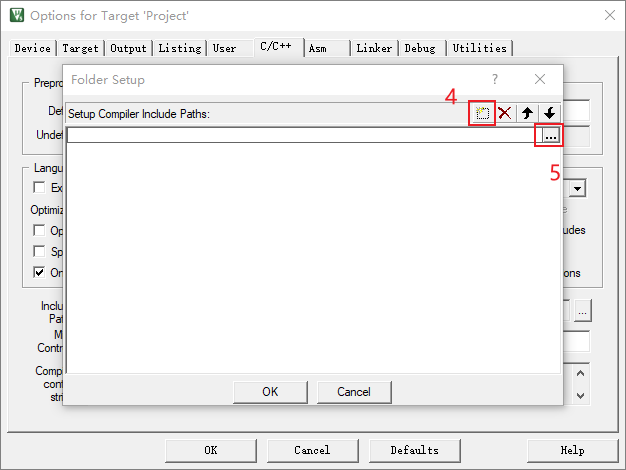


###### 5.将“.h头文件的路径”加入工程

* 上面是把**源文件**添加到了工程中，现在将**头文件**路径加入到 PATH 中，如下图所示：

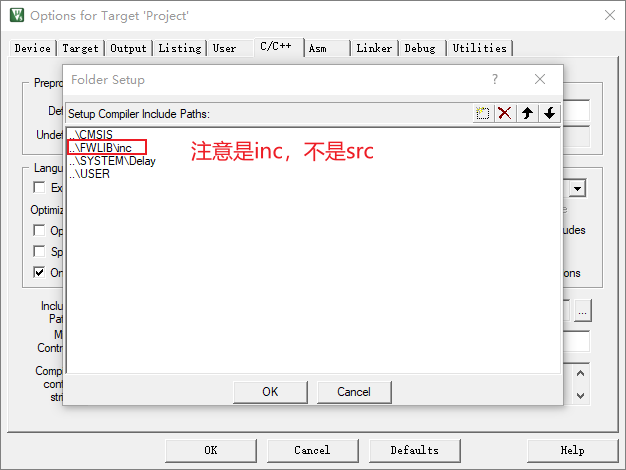
在 MDK 里面设置头文件存放路径。也就是告诉 MDK 到那些目录下面去寻找包含了的头文件。这一步骤非常重要。如果没有设置头文件路径，那么工程会出现报错头文件路径找不到。具体操作如下图，5 步之后添加相应的头文件路径。

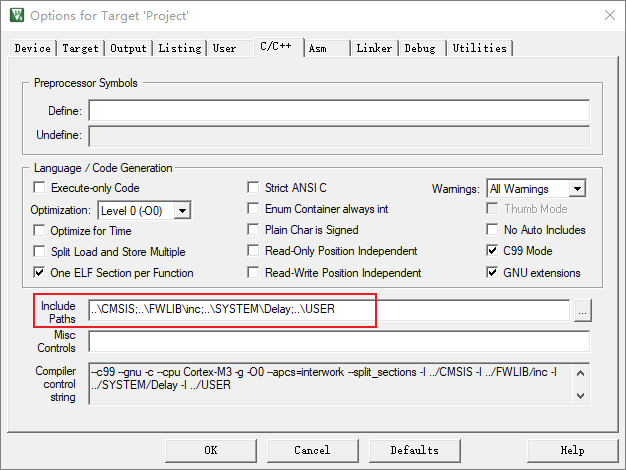




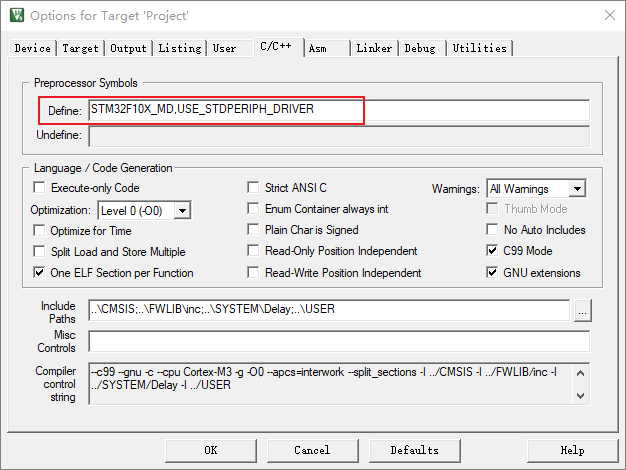
这里我们需要添加的头文件路径包括：\CMSIS、 \FWLIB\inc、\STSTEM\Delay、\USER。

我们将对应的4（模板中的6个子文件夹只有4个有.h文件）个目录加入到 PATH 中去，因为每个目录下面都有相应的.h 头文件。这里大家务必要仔细，固件库存放的头文件子目录是\FWLIB\inc，不是 FWLIB\src。很多朋友都是这里弄错导致报很多奇怪的错误。添加完成之后如下图





###### 6.添加全局宏定义标识符



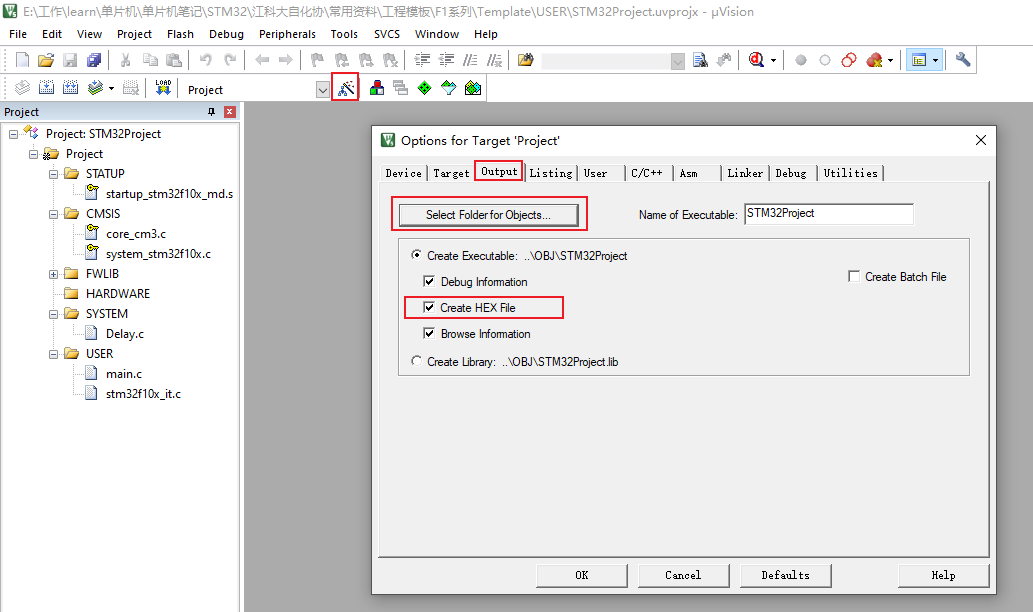
STM32F10X\_MD,USE\_STDPERIPH\_DRIVER

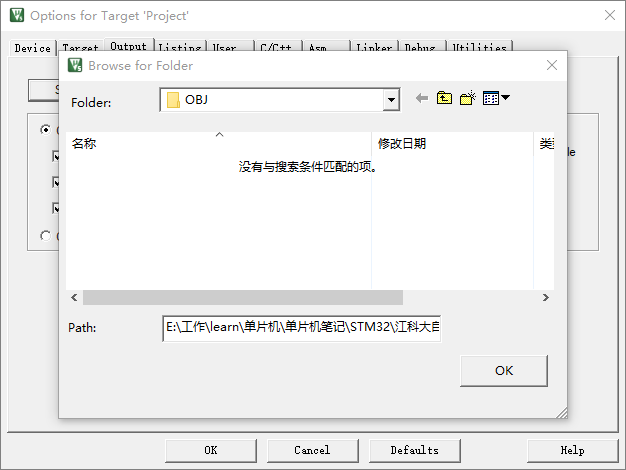
###### 7.设置编译中间文件存放目录

* 在编译之前我们首先要选择编译中间文件编译后存放目录。

点击魔术棒 ，然后选择“Output”选项下面的“Select folder for objects…”,

然后选择目录为我们上面新建的 OBJ 目录。同时将下方的三个选项框都勾上，操作过程如下图



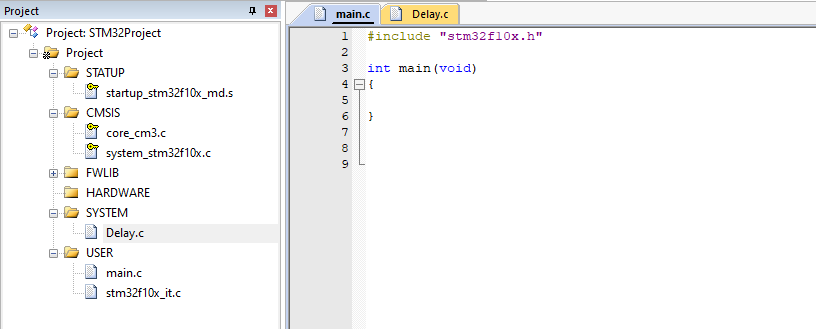


补充：

HEX文件一般是发给工程产间的人用的，他们只需要一个烧录文件就可以了，不需要源码。（通过USB转串口模块）串口下载烧录的也是HEX文件。

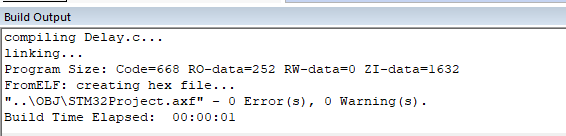
###### 8.测试main.c

这样我们的工程模板就彻底完成了。接下来我们修改主函数所在的文件 main.c的内容，

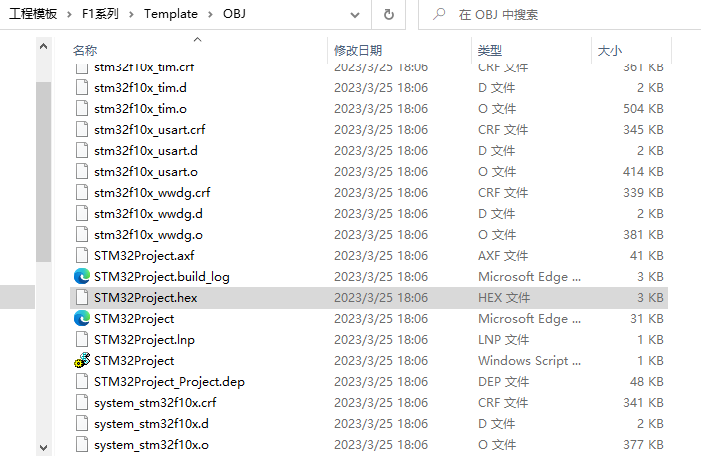


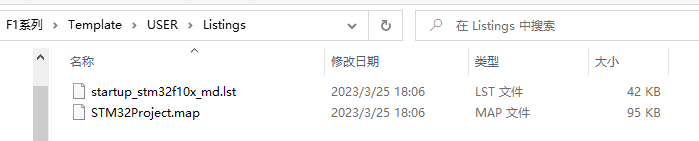
###### 9.编译工程

* 下面我们点击编译按钮编译工程，可以看到工程编译通过没有任何错误和警告

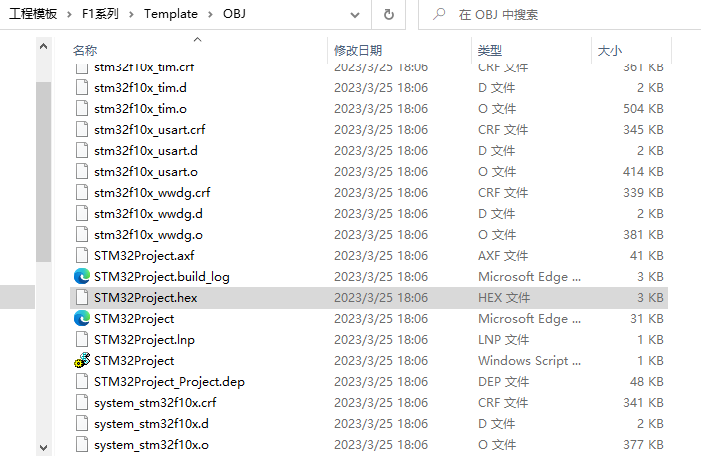


发现\Template\OBJ确实出现了编译中间文件、hex文件





###### 10.查看hex文件



###### 11.后续就可以进行烧录了

##### 补充讨论

###### stm32编译前为什么要配置keil中C/C++中的define 和include Paths？

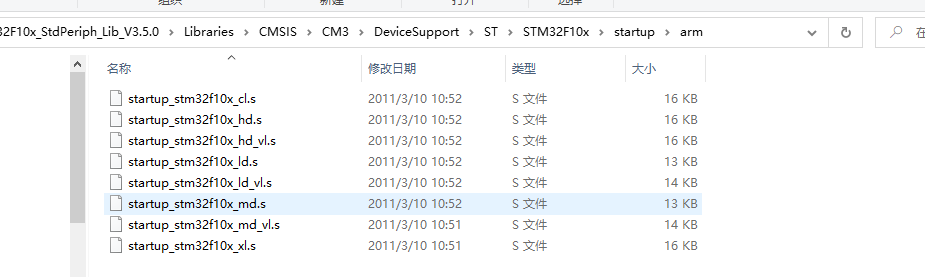
<https://www.jianshu.com/p/7f5549266e2a>



###### 启动文件那么多，选哪个？？？

STM32F10x 的启动文件存放在目录

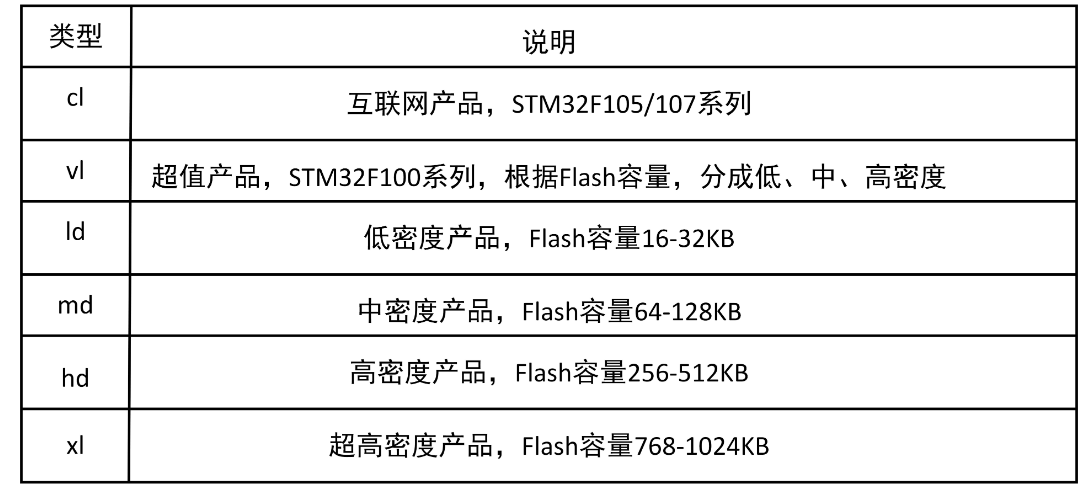
\STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm 下面。

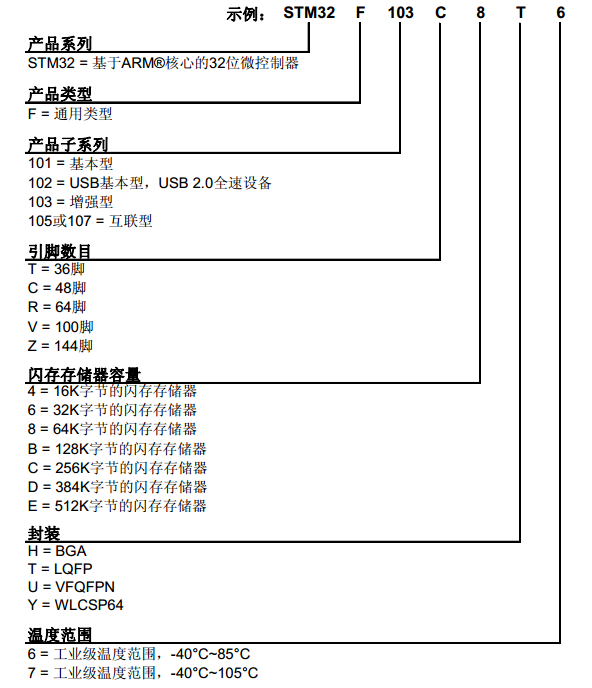


对于不同型号的 STM32F10x 系列对应的启动文件也不一样。

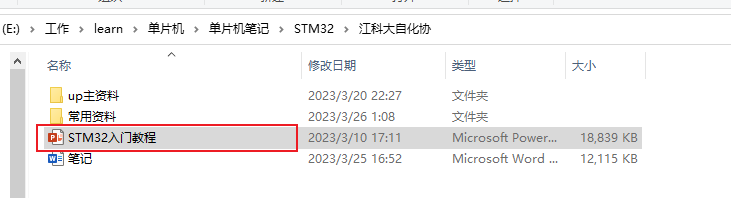
我们的开发板是 STM32F10x 系列所以我们选择的启动文件为startup\_stm32f10x\_md.s。

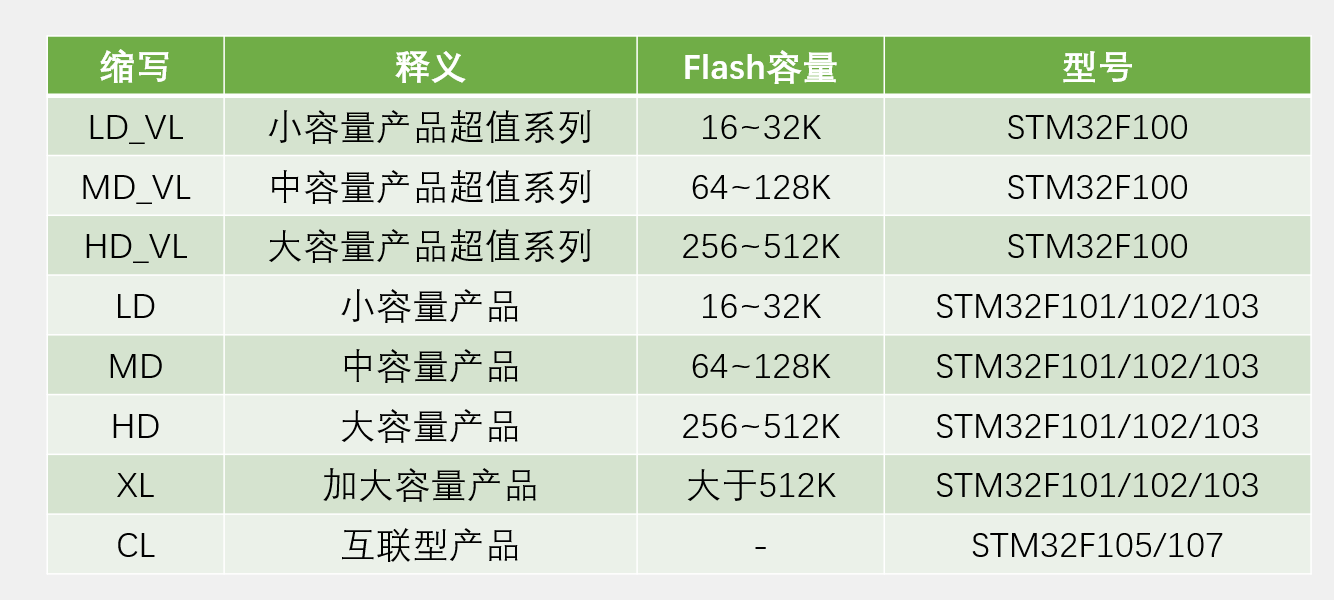
<https://blog.csdn.net/Chrstian/article/details/123945627>





STM32F103C8T6, 8意味着Flash容量是64K，应该是选择“md”





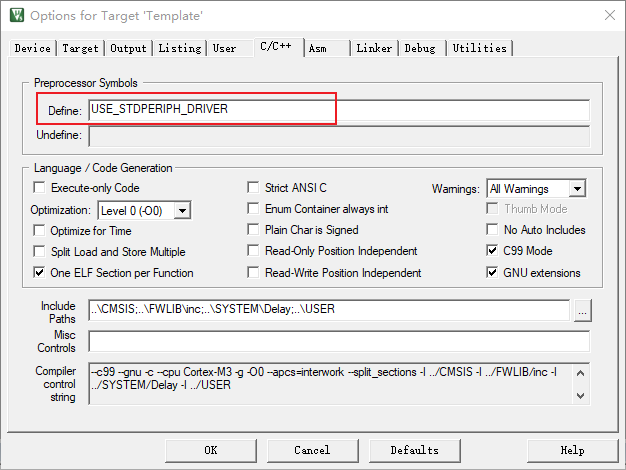
这里也可以看到

###### 全局宏定义标识符怎么填？

江科大自动化协只定义了一个

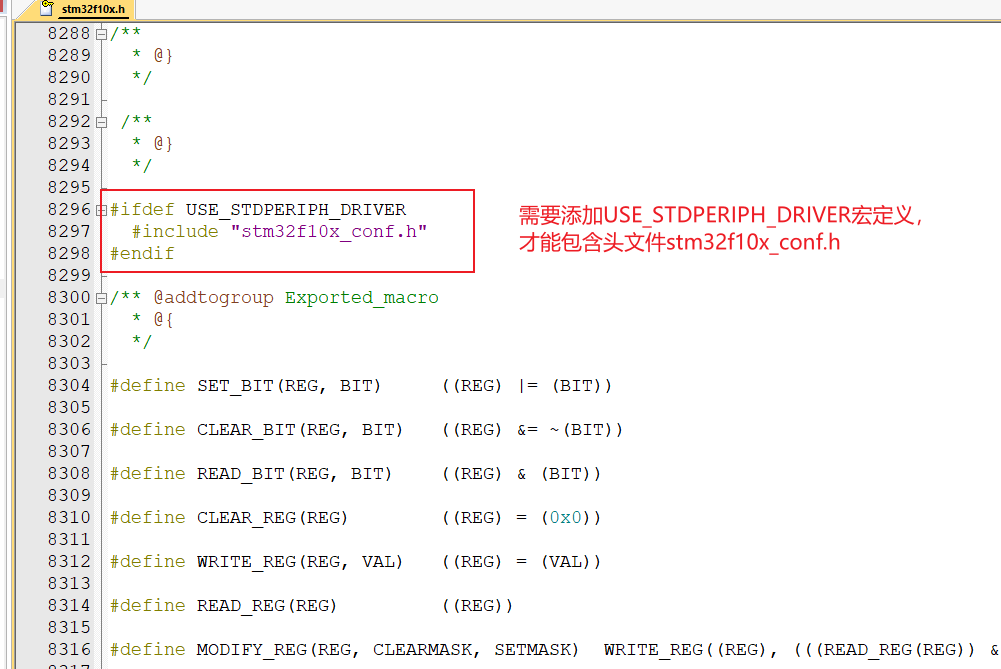
* 接下来对于 STM32F10x 系列的工程，还需要添加一个全局宏定义标识符。添加方法是点击魔术棒之后，进入 C/C++选项卡，然后在 Define 输入框连输入：

USE\_STDPERIPH\_DRIVER



<https://blog.csdn.net/weixin_45992045/article/details/124431806>

预定义功能，相当于在程序中进行#define xxx



另外一种普遍说法是：STM32F10X\_HD, USE\_STDPERIPH\_DRIVER

注意这里是两个标识符,他们之间是用逗号隔开的。

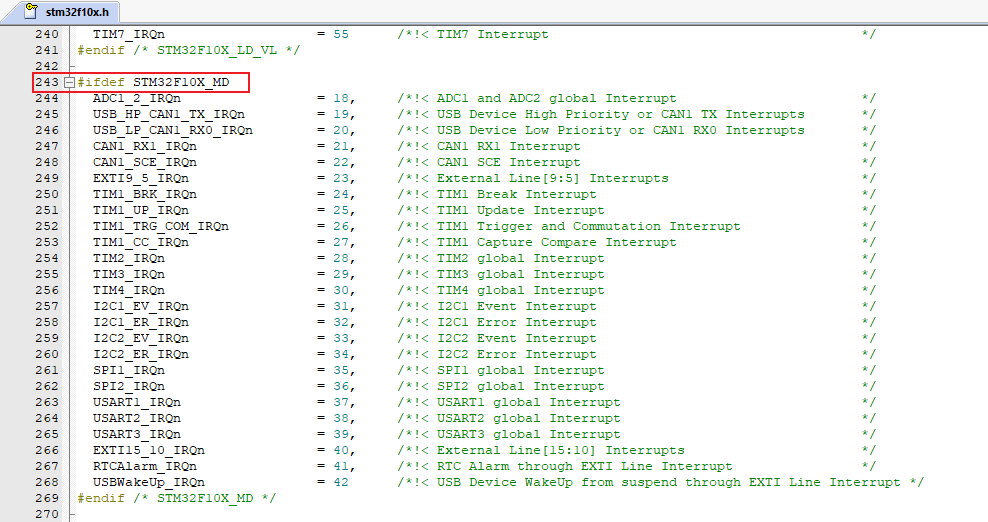
由于我们的启动文件是startup\_stm32f10x\_md.s。对于是STM32F10X\_MD

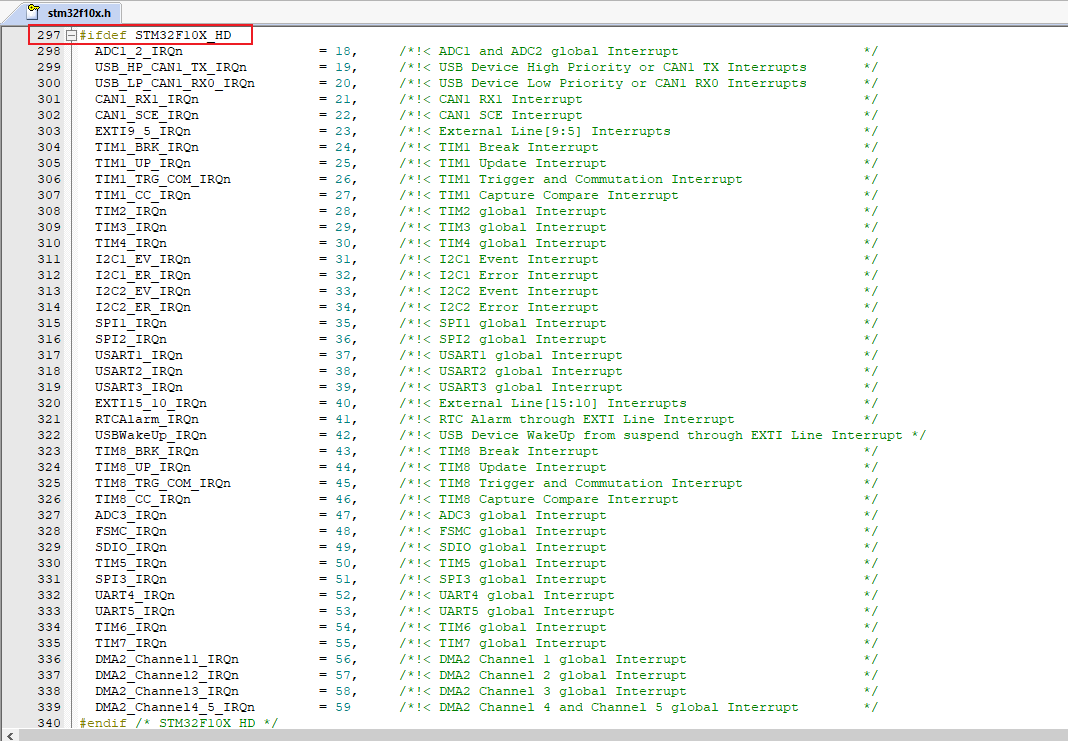
那为什么有的博客填的是STM32F10X\_HD呢？

讨论

<https://www.icxbk.com/ask/detail?tid=17935>







STM32F10X\_HD和STM32F10X\_MD对比来看，实际上是子集关系，即“MD”标识符包含的宏定义是“HD” 标识符包含的子集

HD大容量版本的STM32F10X系列，多了很多外设，比如TIM8、ADC3、TIM5、SPI3等，我们用不到，就算定义了也不影响。

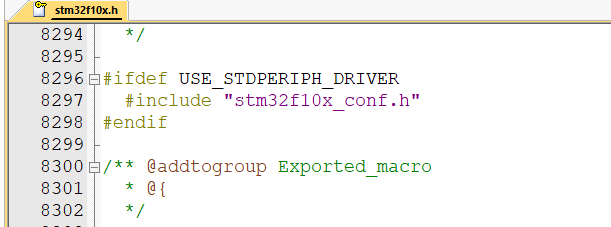
讨论总结：

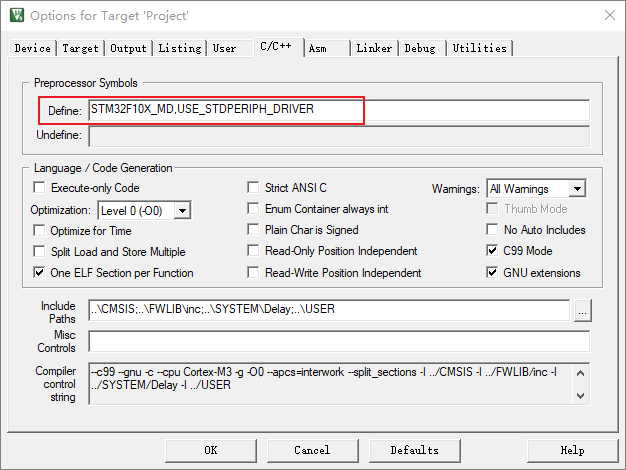
* 需要USE\_STDPERIPH\_DRIVER这个宏定义，才能包含头文件stm32f10x\_conf.h
* STM32F103C8T6, 8意味着Flash容量是64K，“md”
* STM32F10X\_HD和STM32F10X\_MD对比来看，实际上是子集关系。

即“MD”标识符包含的宏定义，是“HD” 标识符包含的宏定义的子集

* 对于STM32F103C8T6来说。C/C++选项卡的 Define 输入框中：STM32F10X\_HD和STM32F10X\_MD没有区别，不影响我们使用。
* 我们就用最准确的STM32F10X\_MD。但是用STM32F10X\_HD也不影响

（根据启动文件名startup\_stm32f10x\_md.s）



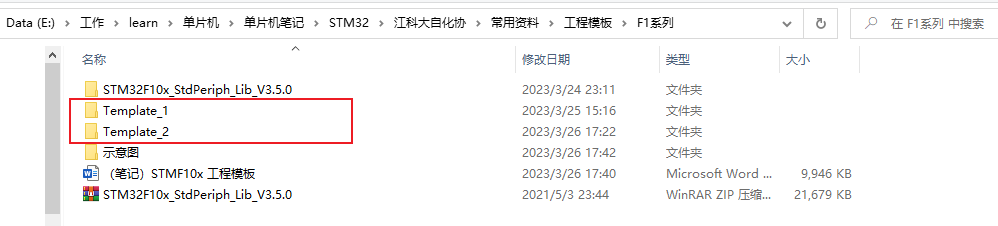


STM32F10X\_MD,USE\_STDPERIPH\_DRIVER

###### 为什么江科大自化协在group中添加了.h文件。按理说Group中只添加.c就够了，在魔术棒中配置头文件路径就行。

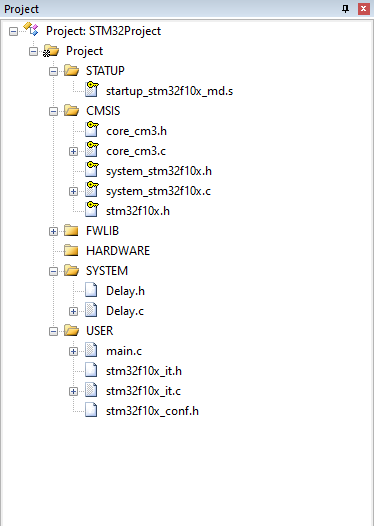
因为.h文件不参与编译，所以很多工程都不添加。

但是我们经常需要打开看.h文件，所以把一些.h文件添加进来方便打开看。



Template\_1：group中只添加了.c文件。

Template\_2：group中添加了.c和.h文件。即工程项目中的所有文件

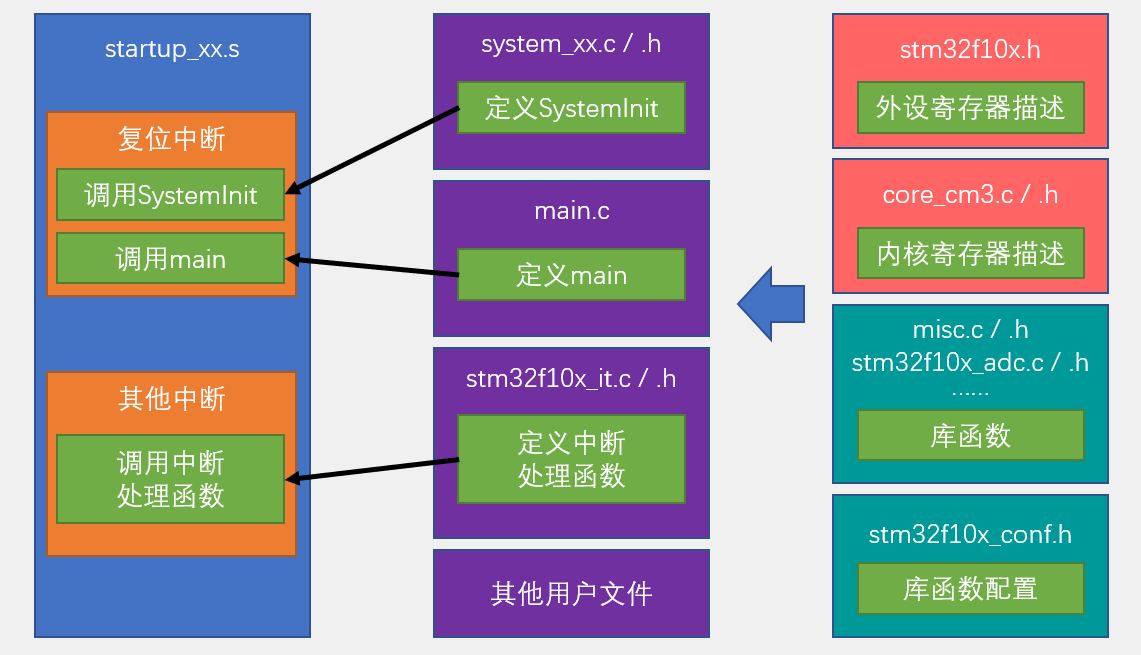


###### 为什么江科大自化协在全局宏定义中只声明了USE\_STDPERIPH\_DRIVER？

按照up的说法，keil5在建立工程时就已经声明了STM32F10X\_MD。

我们在魔术棒的C/C++选项卡中声明不是必要的。

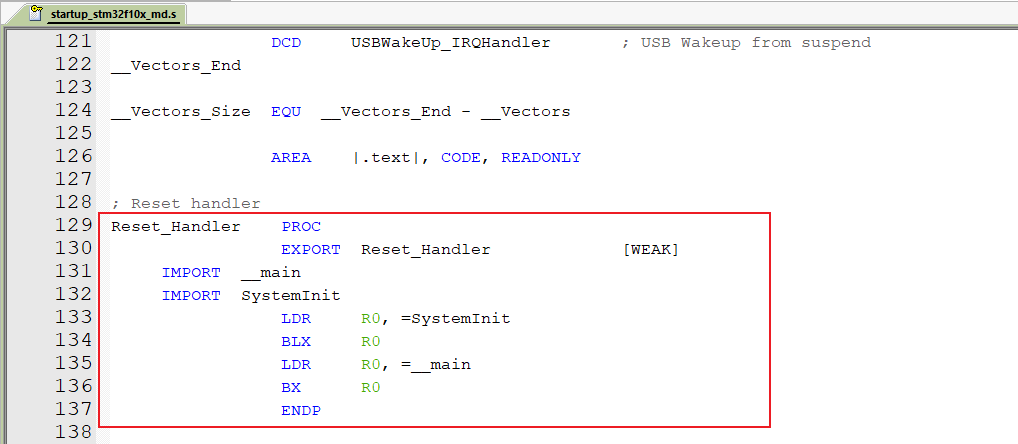
##### 工程架构说明



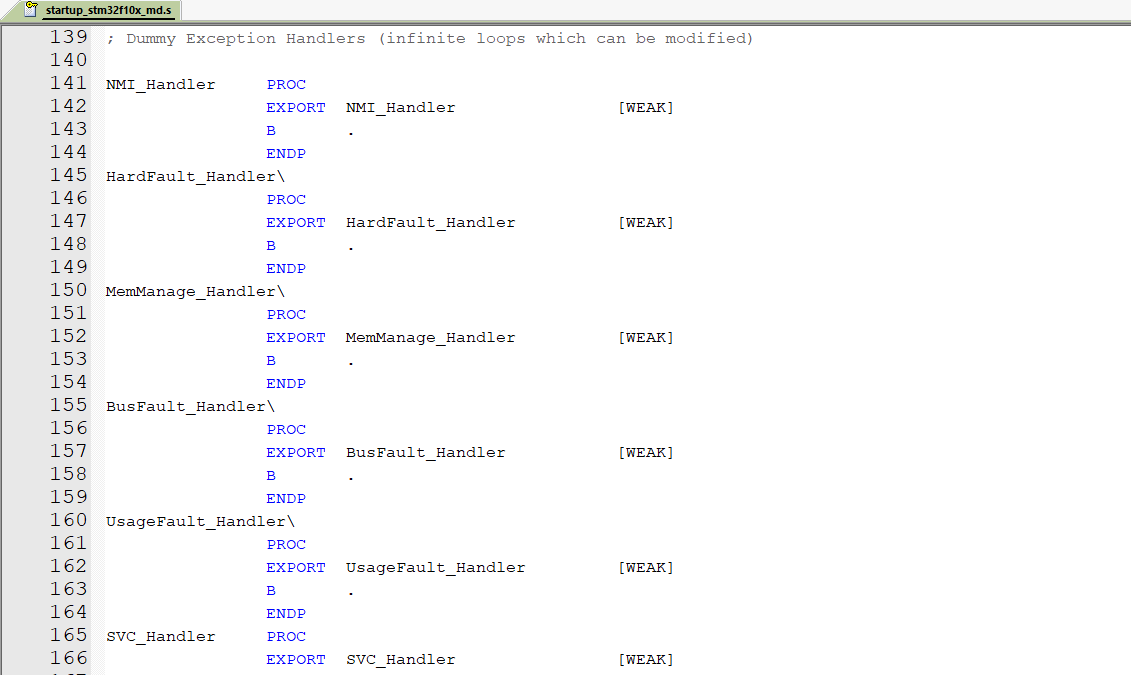
startup启动文件：

程序执行最基本的文件。这个是汇编语言写的。定义了中断向量表、中断服务函数等。中断服务函数中有一个“复位中断”，是整个程序的入口。当STM32上电复位或按下复位按键后，程序就会执行复位中断函数。

复位中断函数主要做两件事：调用SystemInit()函数、调用main()函数。

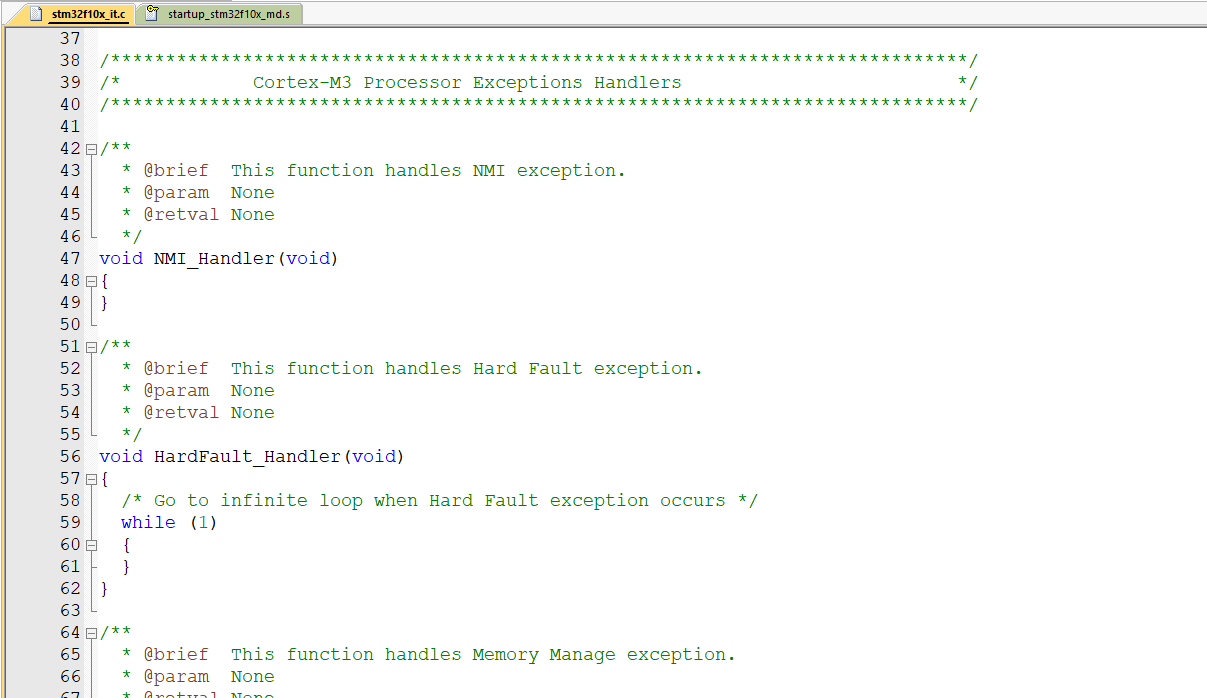


另外，启动文件后还定义了STM32所有的其他中断。这些中断达到触发条件后就会自动执行。



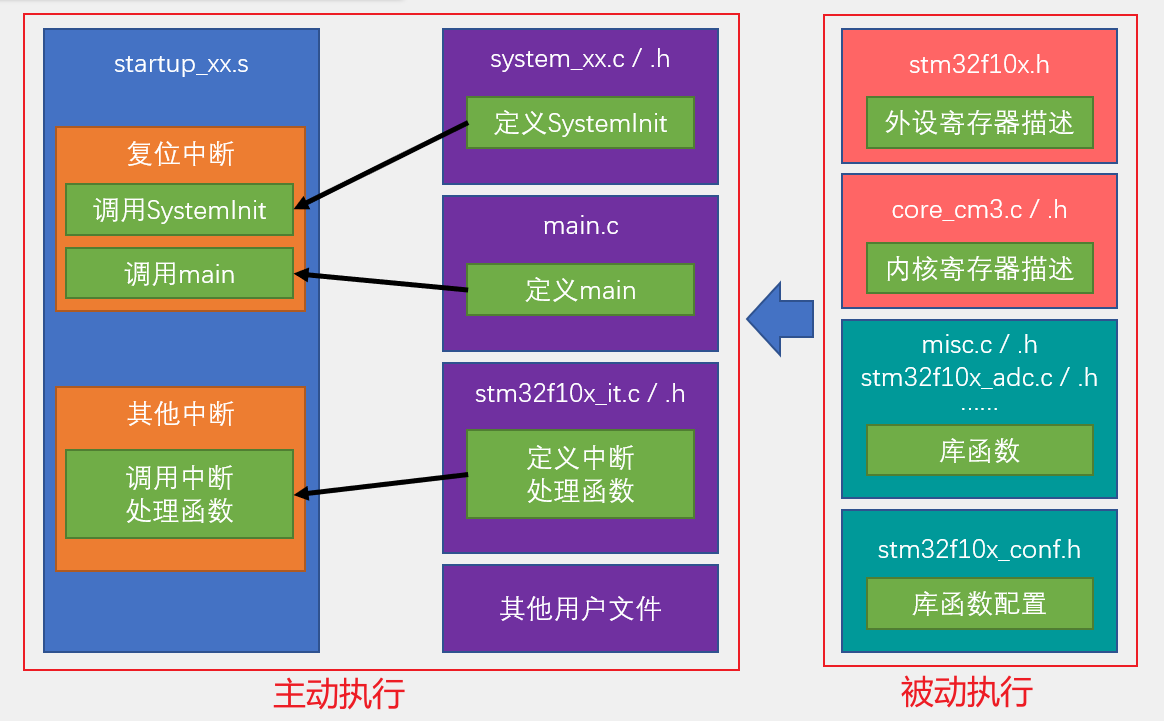
stm32f10x\_it：

上面这些中断函数的定义在stm32f10x\_it中。





当然，我们可以自己定义一些用户模块，封装好，供主函数和中断调用。有利于程序结构的模块化

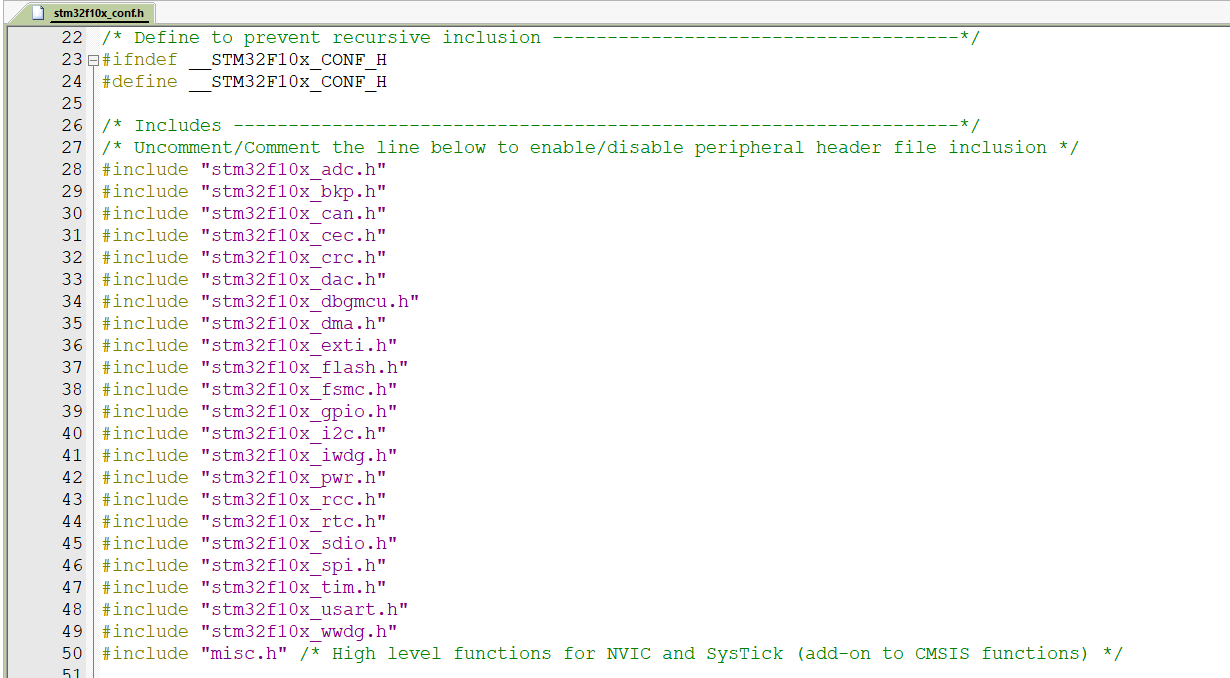


被动执行意思是STM32的资源，我们可以在主函数或中断函数里，可以调用这些资源。

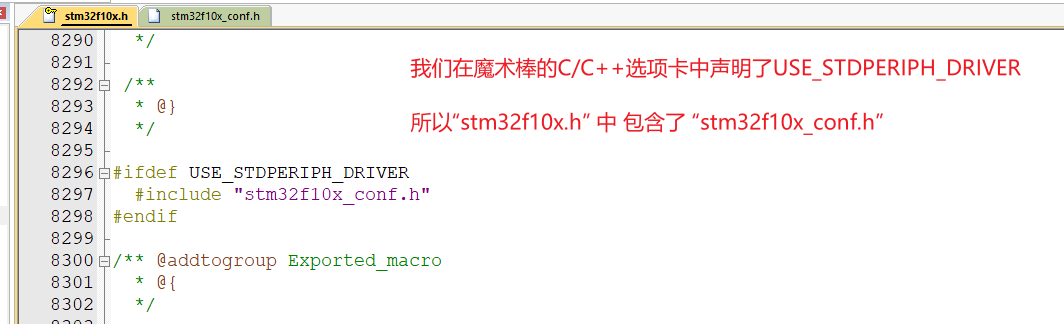
寄存器方式调用资源：stm32f10x.h、core\_cm3.c / .h。

库函数方式调用资源：stm32f10x\_conf.h、misc.c / .h、stm32f10x\_adc.c / .h、…..

stm32f10x\_conf.h 包含了所有的库函数头文件



而“stm32f10x.h”又包含了“ stm32f10x\_conf.h”



所以我们在main.c中只需要包含“stm32f10x.h”，就可以使用所有库函数了

