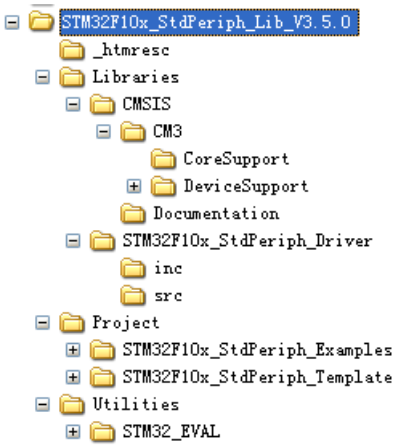
实例学习：

• 创建一个 32 的工程 •

和其他的单片机一样，我们要先搭建一个工程模板以实现我们软件部分的编辑。在创建之前，我们肯定要先下载好我们需要使用的 Keil MDK，以及 STM 官方提供的“固件库包”（可以从 ST 官网进行下载，如何下载百度“ST 官网 固件库”即可。不过我之前下载时，速度非常之慢，可挂 VPN 者可以尝试，或者百度下载也是可以的）。固件库包是 ST 官放提供给我们的使用文件，在 32 工程的配置中调用，这样我们在写程序的时候就可以直接调用固件库包给我们提供的子函数，不仅是为了方便，还有是因为要我们单独去写的话，就算是一行一行、没日没夜地抄，那也要抄上一两个月（我还没听说过有谁想要自己写，所以也许夸张了些）。考虑到手写的工作量过于巨大，加上 C 语言可以实现调用子文件中函数的功能，所以官方就提供给我们这样的工具。我们在之后的入门教学中将使用到的是 3.5.0 的固件库包。固件库包也在不断地升级，在使用更新的固件库包之前要知道有什么地方发生了更新即可（就和游戏打补丁是差不多的）。在下载好的固件库中我们会看到如下图所示的子目录，我们就来简单介绍一下各目录的作用。（参考正点原子提供的资料）



(P 2-4-22) 固件库说明

（1）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（即写在程序开头的库函数引用文件），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（C 语言程序文件）和一个相应的 .h 文件（库函数文件）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。

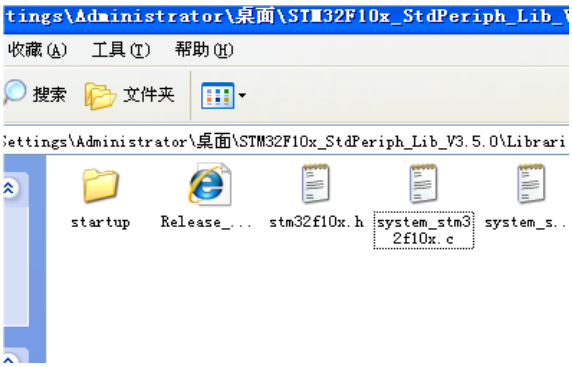
Libraries 文件夹里面的文件在我们建立工程的时候都会使用到。

（2）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代码，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。

（3）Utilities 文件下就是官方评估版的一些对应源码，可以忽略不看。

（4）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。

下面我们要着重介绍 Libraries 目录下面几个重要的文件（引用正点原子提供的资料）。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。永远都不需要修改这个文件。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。

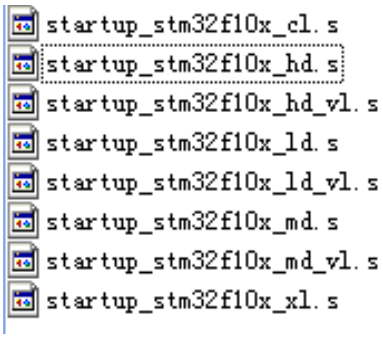


(P 2-4-23) 目录结构

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。几乎包含到平时使用的各类函数以及其他库函数的定义，可谓是编写程序的核心文件。

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。



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；

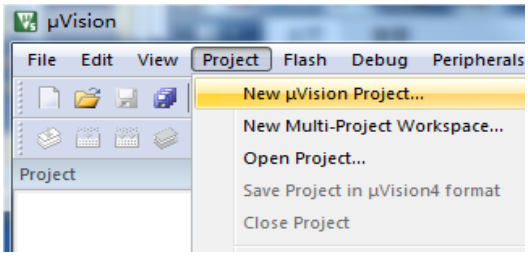
中容量：64K ≤ FLASH ≤ 128K；

大容量：256K ≤ FLASH；

介绍就到此为止。出于固件库的配置要求苛刻，且不可以出现半点儿的差错以杜绝之后的工作中出现基础的错误，所以我从商家附赠的教学资料里（正点原子提供的资料《原子教你学 32 • 库函数版》）截取配置工程的那一部分，并且提供在最后提供给大家所有 32 资料的下载方式，方便未开通睿思（走校园网内部流量）或者是网络流量不够用（其实 18 年上半年就有了校园无限流量卡）的同学。

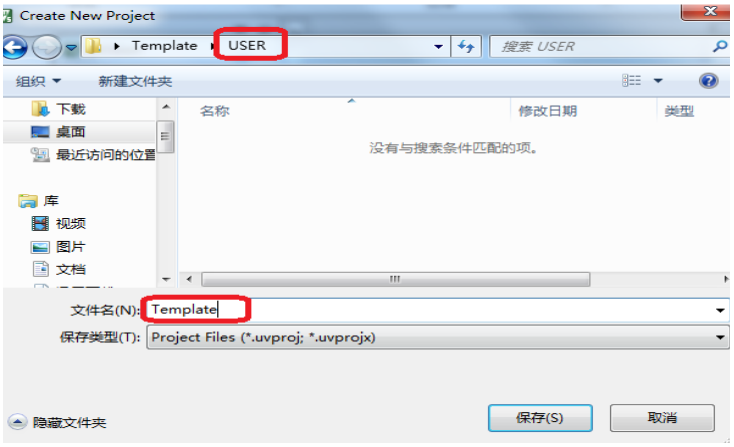
（1）在建立工程之前，我们建议大家在电脑的某个目录下面建立一个文件夹，后面所建立的工程都可以放在这个文件夹下面，这里我们建立一个文件夹为 Template（模板）。

（2）点击 MDK 的菜单：Project ->New



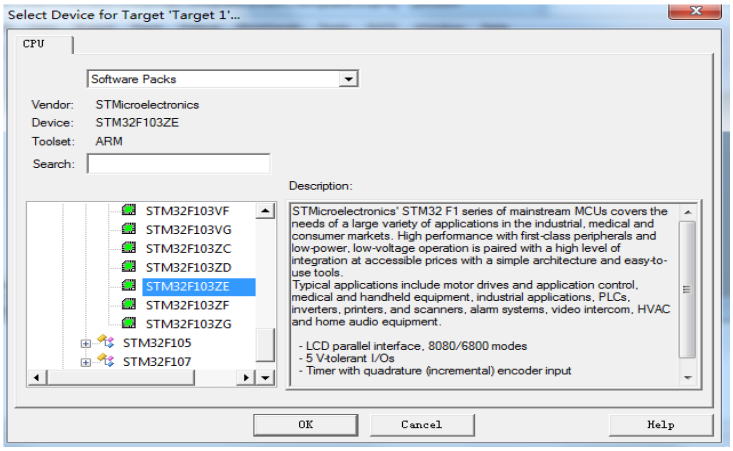
(P 2-4-25) 新建工程

然后将目录定位到刚才建立的文件夹 Template 之下，在这个目录下面建立子文件夹 USER（我们的代码工程文件都是放在 USER 目录，很多人喜欢新建“Project”目录放在下面，这也是可以的，这个就看个人喜好了），然后定位到 USER 目录下面，我们的工程文件就都保存到 USER 文件夹下面。工程命名为 Template，点击保存。



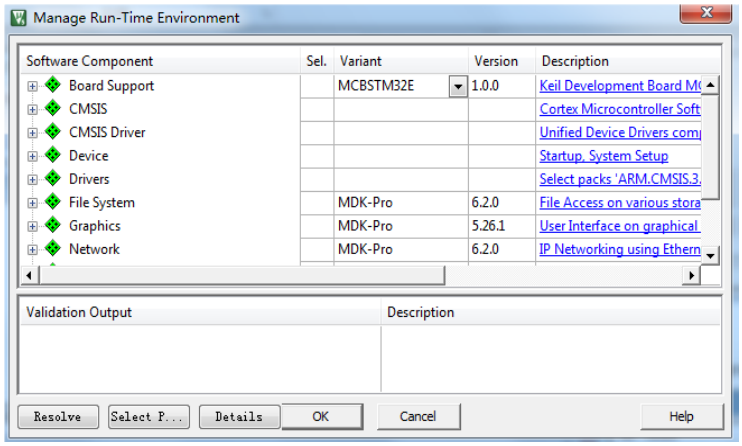
（P 2-4-26）定义工程名称

接下来会出现一个选择 CPU 的界面，就是选择我们的芯片型号。如图 3 所示，以 STM32 型号为 STM32F103ZET6 的板子为例，所以在这里我们选择 STMicroelectronics → STM32F1 Series → STM32F103 → STM32F103ZET6（如果使用的是其他系列的芯片，选择相应的型号就可以了，特别注意：一定要安装对应的器件包才会显示这些内容，就是我们之前提到的支持包 Keil. STM32F1xx_DFP. 1. 0. 5. pack。



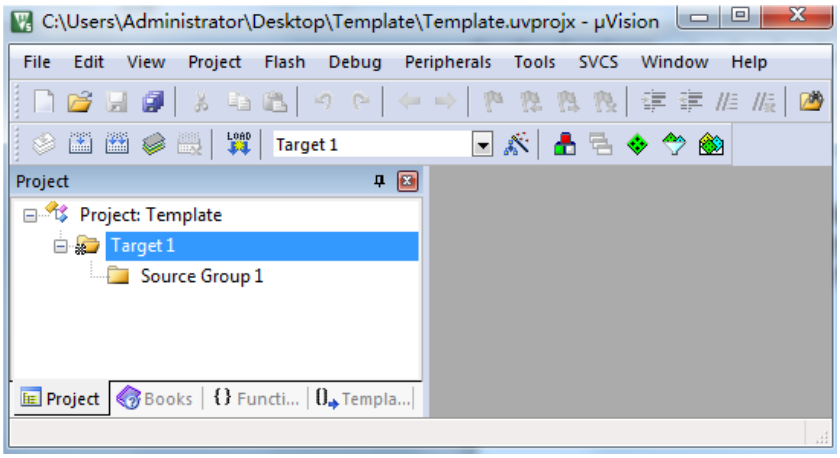
（P 2-4-27） 选择芯片型号

(3) 点击 OK，MDK 会弹出 Manage Run-Time Environment 对话框，如（P 2-4-28）所示：



（P 2-4-28）Manage Run-Time Environment 界面

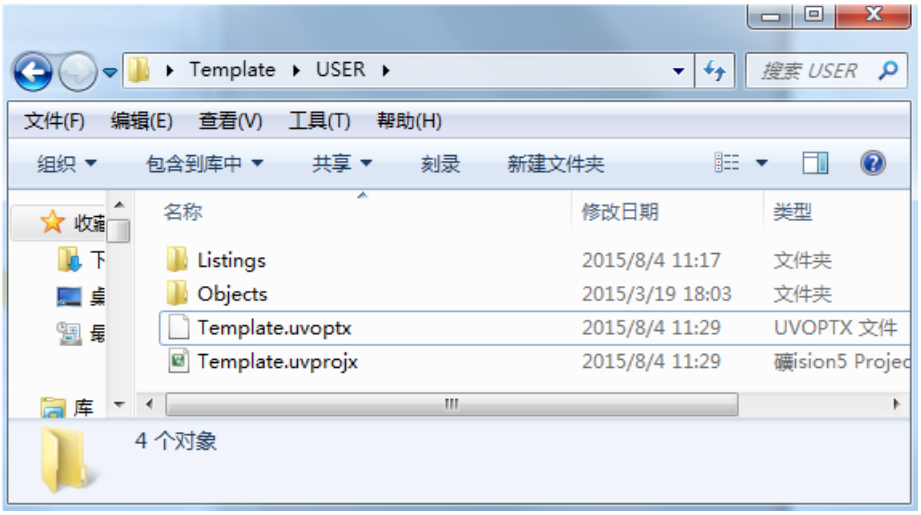
这是 MDK5 新增的一个功能，在这个界面，我们可以添加自己需要的组件，从而方便构建开发环境，不过这里我们不做介绍。所以在图 4 所示界面，我们直接点击 Cancel，即可，得到如图 5 所示界面：



（P 2-4-29）工程初步建立

到这里，我们还只是建了一个框架，还需要添加启动代码，以及 .c 文件等。

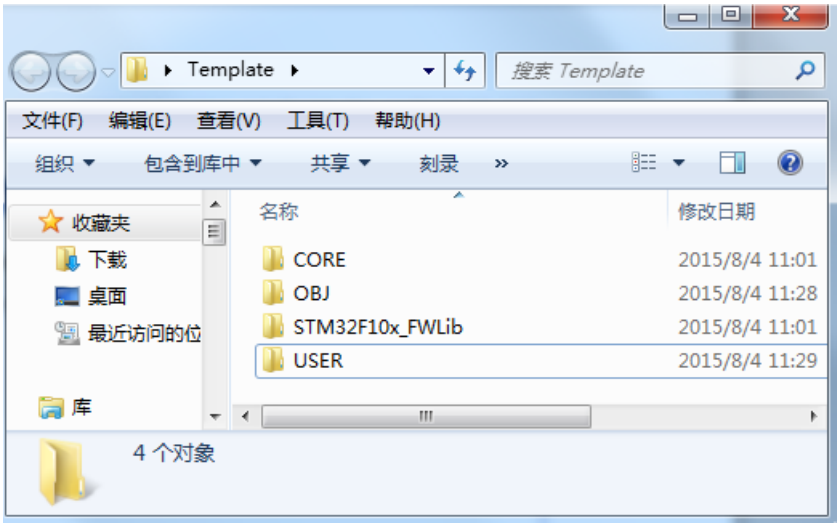
（4）现在我们看看 USER 目录下面包含 2 个文件夹和 2 个文件，如下图 6 所示：



（P 2-4-30）工程 USER 目录文件

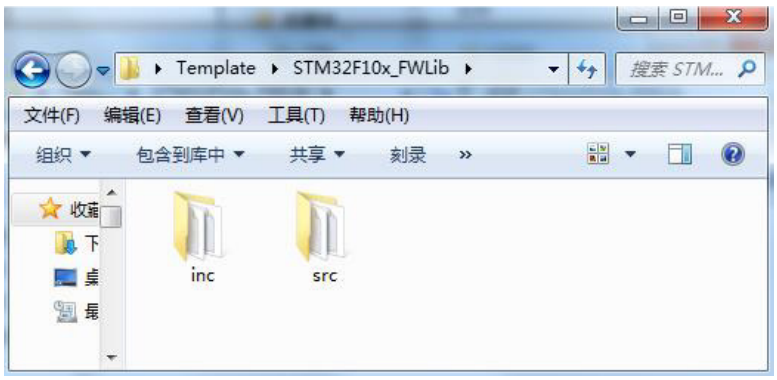
这里说明一下，Template.uvprojx 是工程文件（类似于 word 创建时会有个隐藏的备份一样），非常关键，不能轻易删除。Listings 和 Objects 文件夹是 MDK 自动生成的文件夹，用于存放编译过程产生的中间文件。这里，为了跟 MDK5.1 之前版本工程兼容，我们把两个文件夹删除，我们会在下一步骤中新建一个 OBJ 文件夹，用来存放编译中间文件。当然，我们不删除这两个文件夹也是没有关系的，只是我们不用它而已。

（5）接下来，我们在 Template 工程目录下面，新建 3 个文件夹 CORE，OBJ 以及 STM32F10x_FWLib。CORE 用来存放核心文件和启动文件，OBJ 是用来存放编译过程文件以及生成的 hex 文件，STM32F10x_FWLib 文件夹顾名思义用来存放 ST 官方提供的库函数源码文件。已有的 USER 目录除了用来放工程文件外，还用来存放主函数文件 main.c，以及其他包括 system_stm32f10x.c 等等。



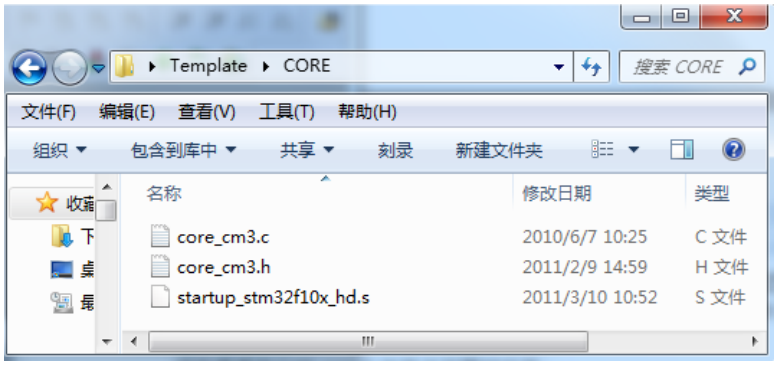
（P 2-4-31）工程目录预览

（6）下面我们要将官方的固件库包里的源码文件复制到我们的工程目录文件夹下面。打开官方固件库包，定位到我们之前准备好的固件库包的目录 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver 下面，将目录下面的 src，inc 文件夹复制到我们刚才建立的 STM32F10x_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，有兴趣者不妨打开一下里面的文件看一下，使用每个外设的代码对应的即是一个 .c 文件和一个 .h 头文件。



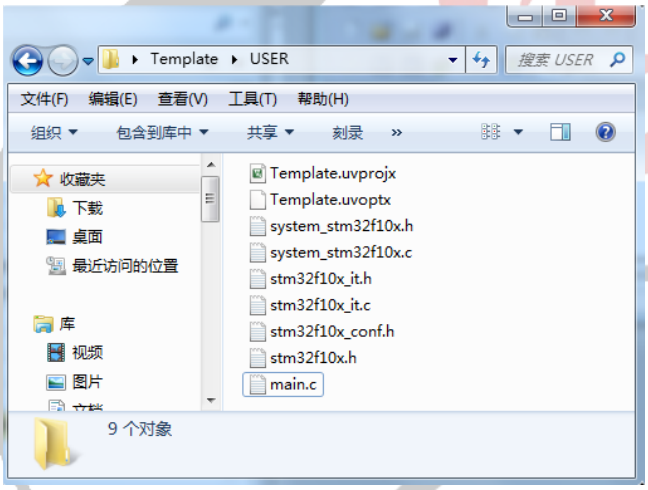
（P 2-4-32）官方库源码文件夹

（7）下面我们要将固件库包里面相关的启动文件复制到我们的工程目录 CORE 之下。打开官方固件库包，定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport 下面，将文件 core_cm3.c 和文件 core_cm3.h 复制到 CORE 下面去。然后定位到目录 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\Device Support\ST\ STM32F10x\startup\arm 下面，将里面 startup_stm32f10x_hd.s 文件复制到 CORE 下面。我们之前已经解释了不同容量的芯片需要使用不同的启动文件，我们要使用的芯片 STM32F103ZET6 是大容量芯片，所以选择这个启动文件。现在看看我们的 CORE 文件夹下面的文件：



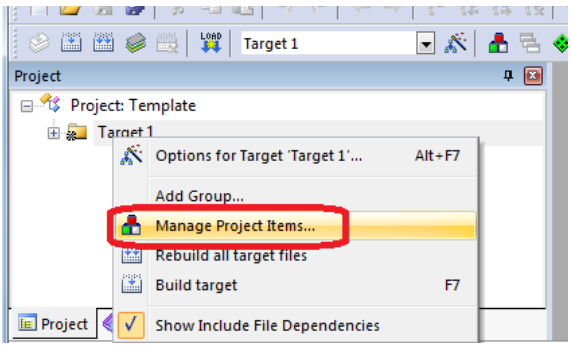
（P 2-4-33）CORE 文件夹内部分文件

（8）定位到目录：STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x 下面，将里面的三个文件 stm32f10x.h，system_stm32f10x.c，system_stm32f10x.h，复制到我们的 USER 目录之下。然后将 STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template 下面的 4 个文件 main.c，stm32f10x_conf.h，stm32f10x_it.c，stm32f10x_it.h 复制到 USER 目录下面。



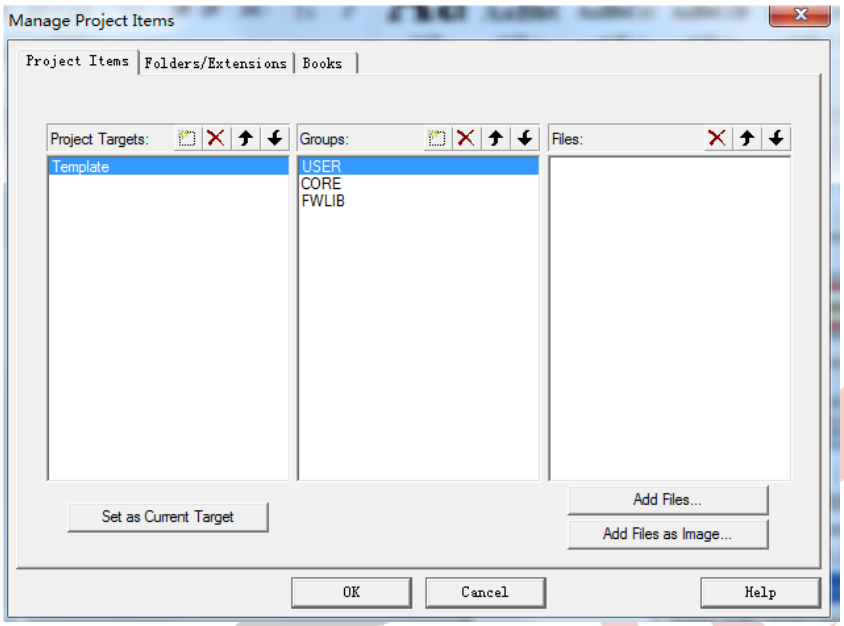
（P 2-4-34） USER 目录文件浏览

（9）前面 8 个步骤，我们将需要的固件库相关文件复制到了我们的工程目录下面，下面我们将这些文件加入我们的工程中去。右键点击 Target1，选择 Manage Project Items。

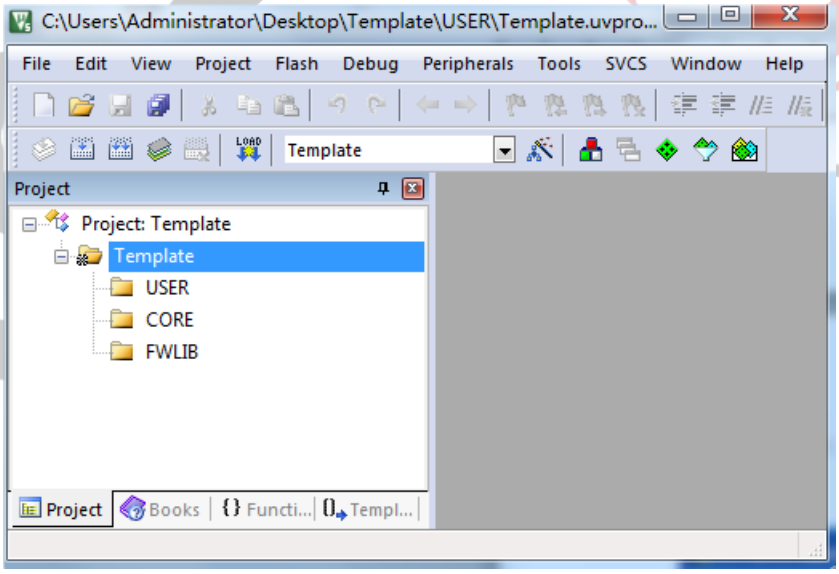


（P 2-4-35）点击 Management Project Itmes

(10)Project Targets 一栏，我们将 Target 名字修改为 Template，然后在 Groups 一栏删掉一个 SourceGroup1，建立三个 Groups：USER，CORE，FWLIB。然后点击 OK，可以看到我们的 Target 名字以及 Groups 情况。



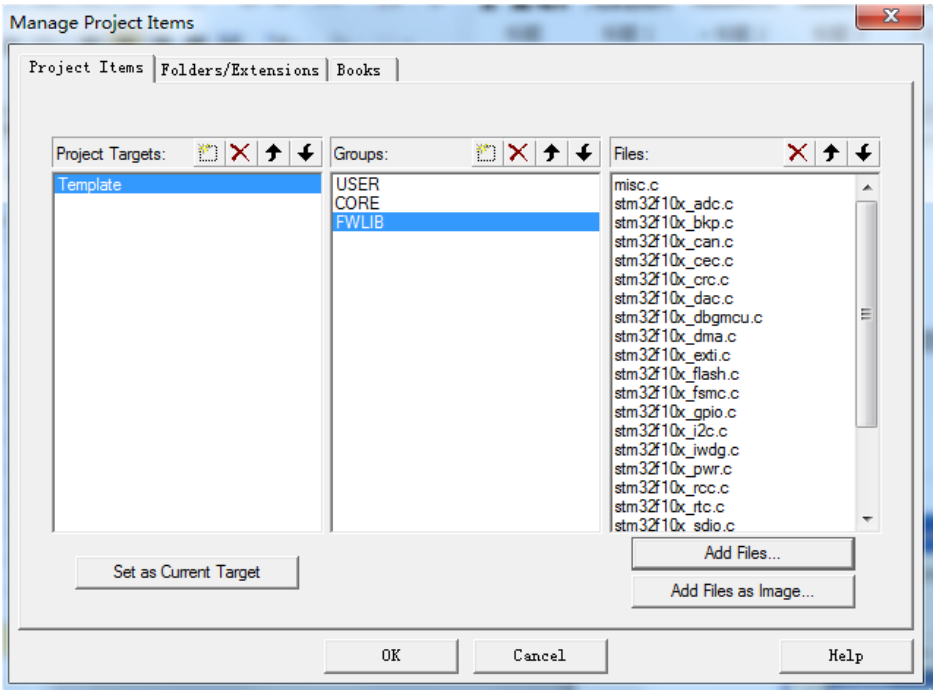
（P 2-4-36）新建分组



（P 2-4-370）工程主界面

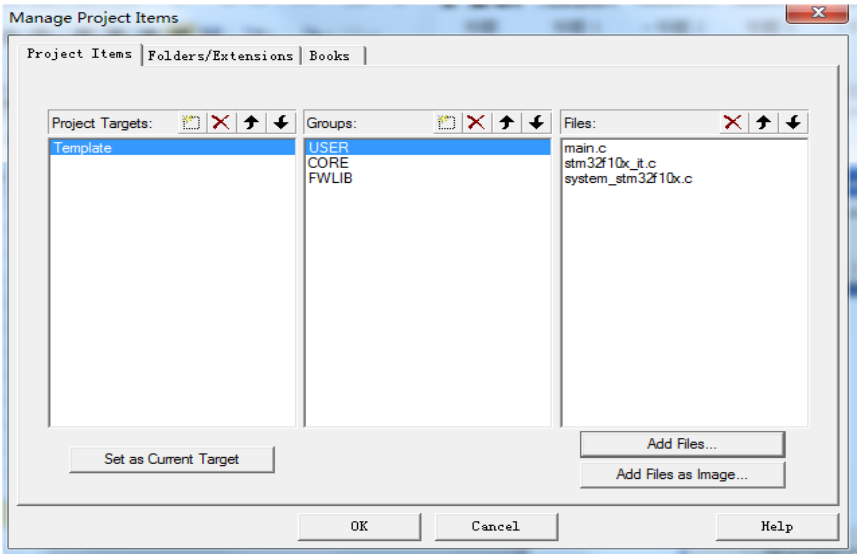
(11) 下面我们往 Group 里面添加我们需要的文件。我们按照步骤 10 的方法，右键点击点 Tempate，选择选择 Manage Project Itmes，然后选择需要添加文件的 Group，这里第一步我们选择 FWLIB，然后点击右边的 Add Files，定位到我们刚才建的目录 STM32F10x_FWLib/src 下面，将里面所有的文件选中（Ctrl+A），然后点击 Add，然后 点击 Close. 可以看到 Files 列表下面包含我们添加的文件。

这里需要说明一下，对于我们写代码，如果我们只用到了其中的某个外设，我们就可以不用添加没有用到的外设的库文件。例如我只用 GPIO，我可以只用添加 stm32f10x_gpio.c 而其他的可以不用添加。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，编译起来速度慢，各位同学可以自行选择。

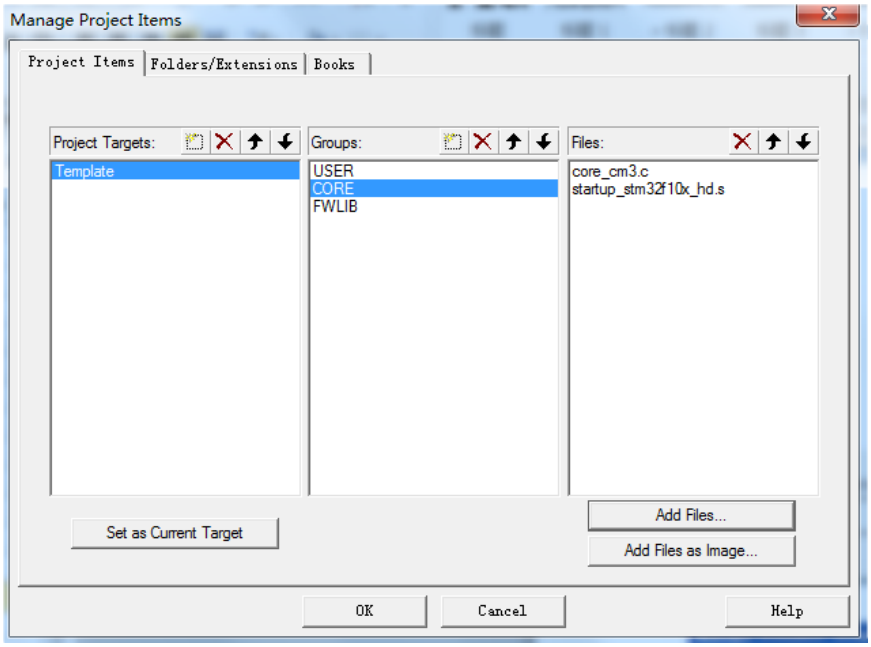


（P 2-4-37）添加文件到 FWLIB 分组

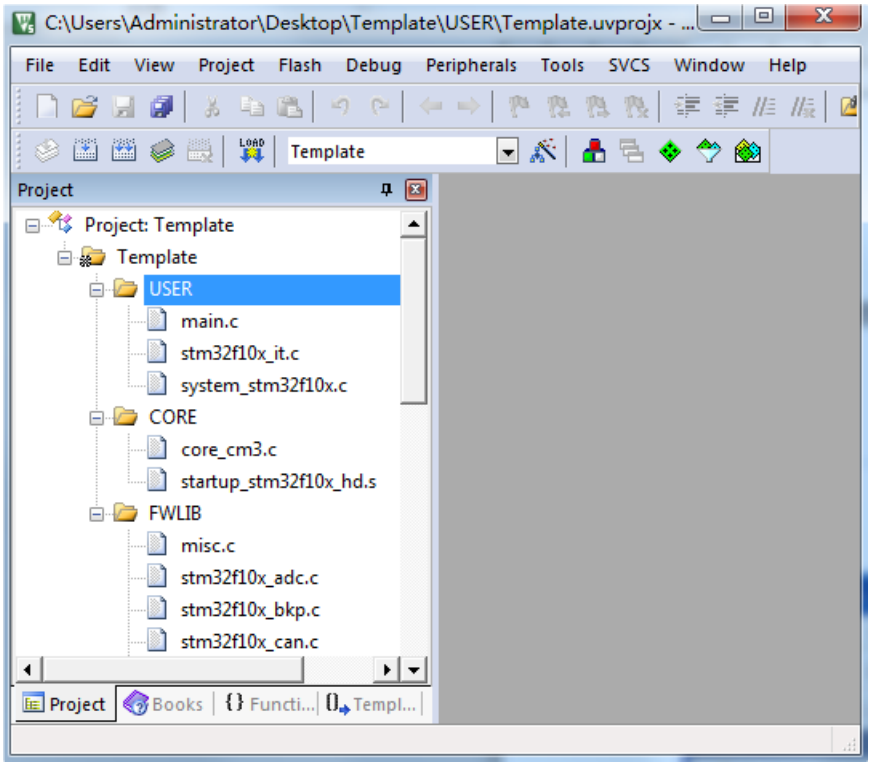
（12）用同样的方法，将 Groups 定位到 CORE 和 USER 下面，添加需要的文件。这里我们的 CORE 下面需要添加的文件为 core_cm3.c，startup_stm32f10x_hd.s（注意，默认添加的时候文件类型为.c，也就是添加 startup_stm32f10x_hd.s 启动文件的时候，你需要选择文件类型为“All files”才能看得到这个文件），USER 目录下面需要添加的文件为 main.c，stm32f10x_it.c，system_stm32f10x.c，这样我们需要添加的文件已经添加到我们的工程中了，最后点击 OK，回到工程主界面。



（P 2-4-38）添加文件到 USER 分组

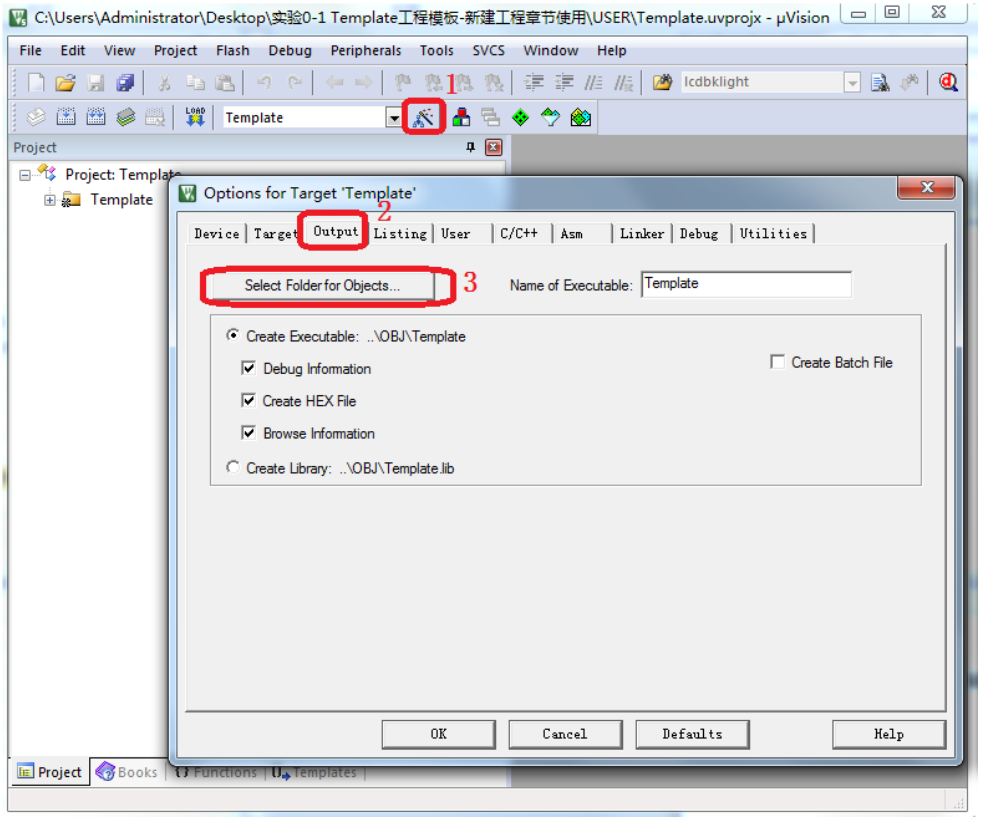


（P 2-4-39） 添加文件到 FWLIB 分组



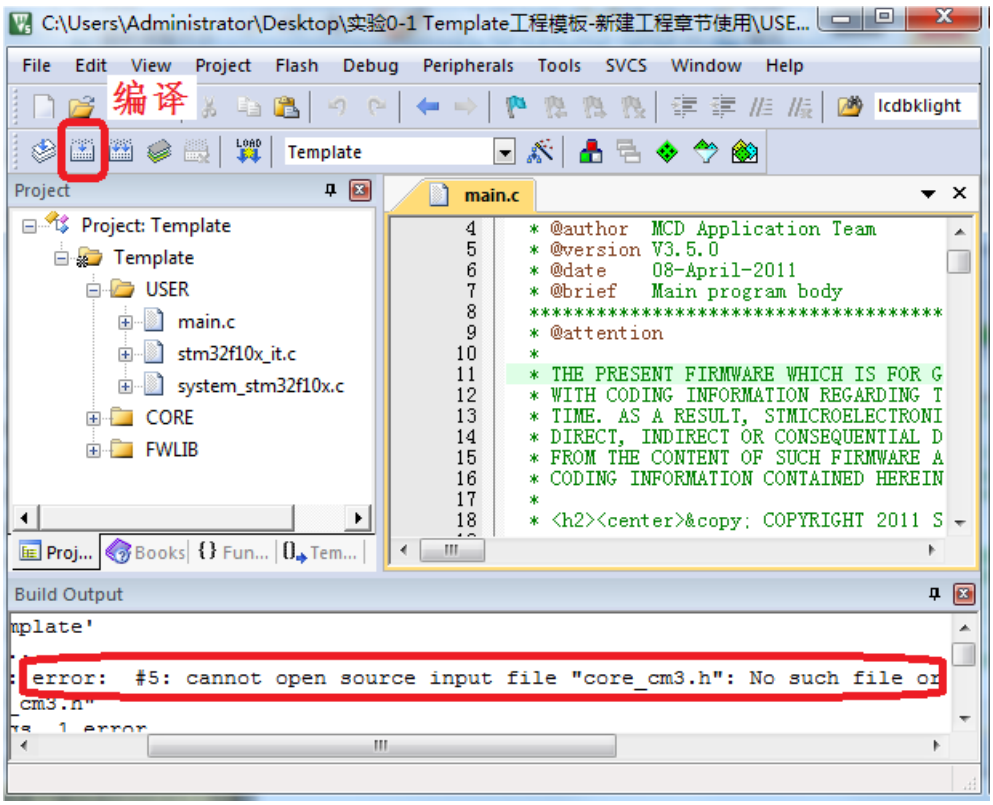
（P 2-4-40） 添加文件到 FWLIB 分组

（13）接下来我们要编译工程，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。方法是点击魔术棒（叫仙女棒也是没差的啦），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。这里大家注意，如果我们不设置 Output 路径，那么默认的编译中间文件存放目录就是 MDK 自动生成的 Objects 目录和 Listings 目录。



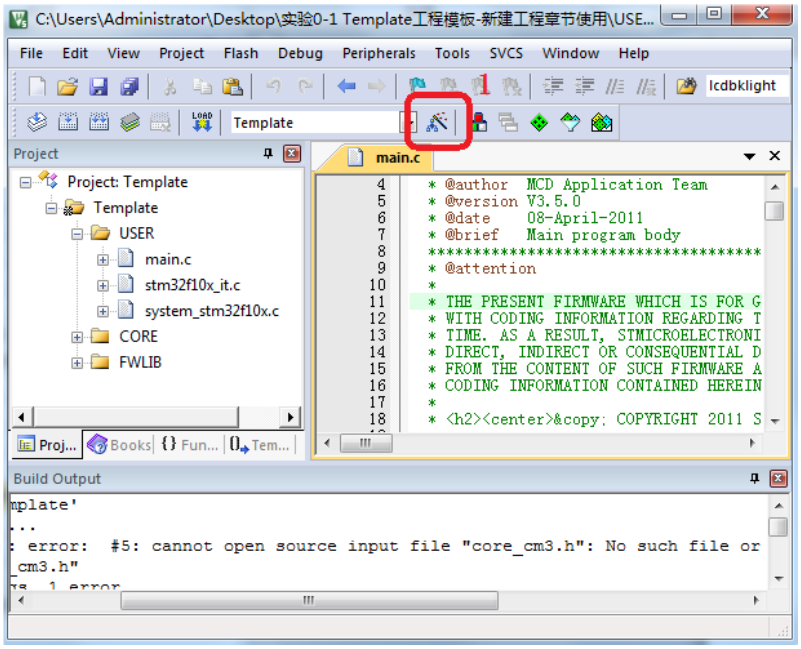
(P 2-4-41) 选择编译后的文件存放目录

(14) 下面我们点击编译按钮 编译工程，可以看到很多报错，因为找不到头文件。

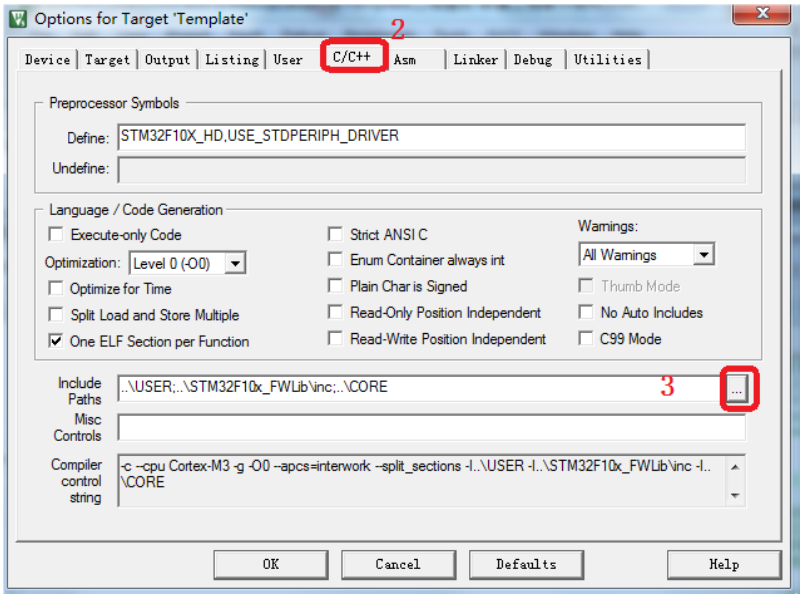


(P 2-4-42) 编译工程

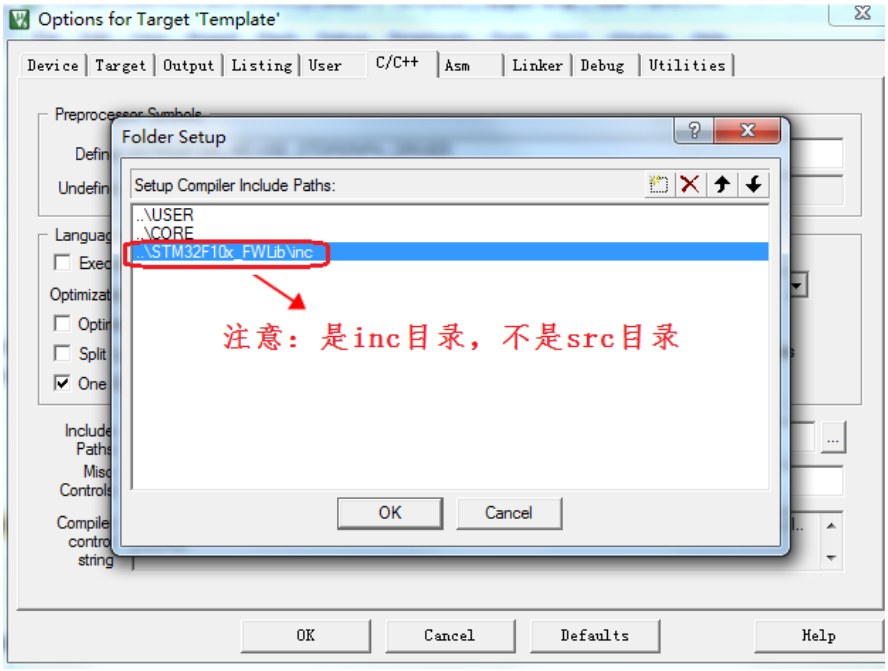
15）下面我们要告诉 MDK，通过哪些路径之下搜索编译工程时需要的头文件，也就是头文件目录。这里大家要注意，对于任何一个工程，我们都需要把工程中引用到的所有头文件的路径都包含到进来（如果没有添加的话，编译是无法通过的，我初学时，常因为这个问题而耽误时间）回到工程主菜单，点击魔术棒，出来一个菜单，然后点击 c/c++选项．然后点击 Include Paths 右边的按钮。弹出一个添加 path 的对话框，然后我们将图上面的 3 个目录添加进去。记住，keil 只会在一级目录查找，所以如果你的目录下面还有子目录，记得 path 一定要定位到最后一级子目录。然后点击 OK。



（P 2-4-43 ）点击魔术棒

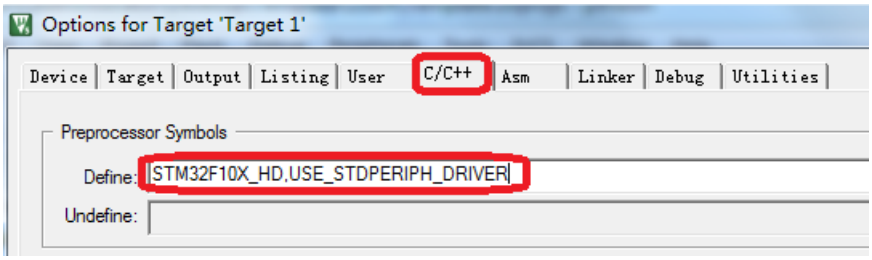


（P 2-4-44）C/C++ 选项卡



(P 2-4-45) 添加头文件路径到 PATH

(16) 接下来，我们再来编译工程，可以看到又报了很多同样的错误。为什么呢？这是因为 3.5 版本的库函数在配置和选择外设的时候通过宏定义来选择的，所以我们需要配置一个全局的宏定义变量。按照步骤 16，定位到 c/c++ 界面，然后填写“STM32F10X_HD,USE_STDPERIPH_DRIVER”到 Define 输入框里面（请注意，两个标识符中间是英文格式下的逗号而不是句号）。这里解释一下，如果以后用到的芯片是中容量的，那么 STM32F10X_HD 修改为 STM32F10X_MD，小容量的则修改为 STM32F10X_LD。然后点击 OK。

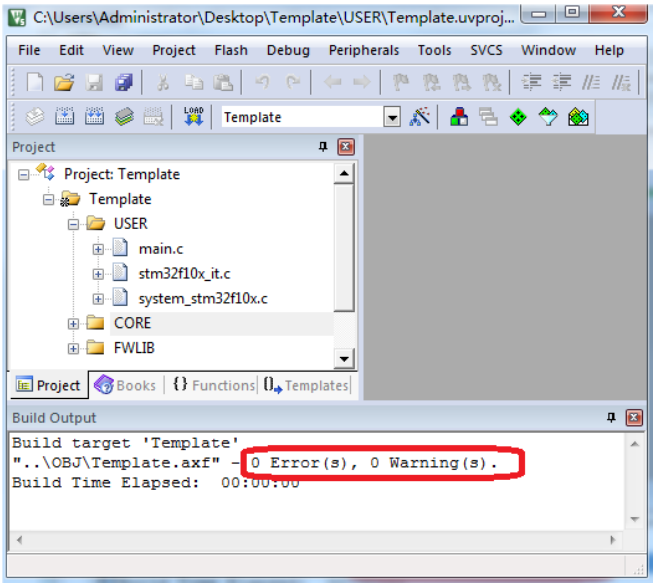


(P 2-4-46) 添加全局宏定义标识符到 Define 输入框

(17) 这次在编译之前，我们记得打开工程 USER 下面的 main.c 文件，复制下面代码到 main.c 覆盖已有代码(只是个试验是否编译通过的代码,使用别的代码也行的),然后进行编译。(记得在代码的最后面加上一个回车，否则会有警告“Warning”)，可以看到，这次编译已经成功了。

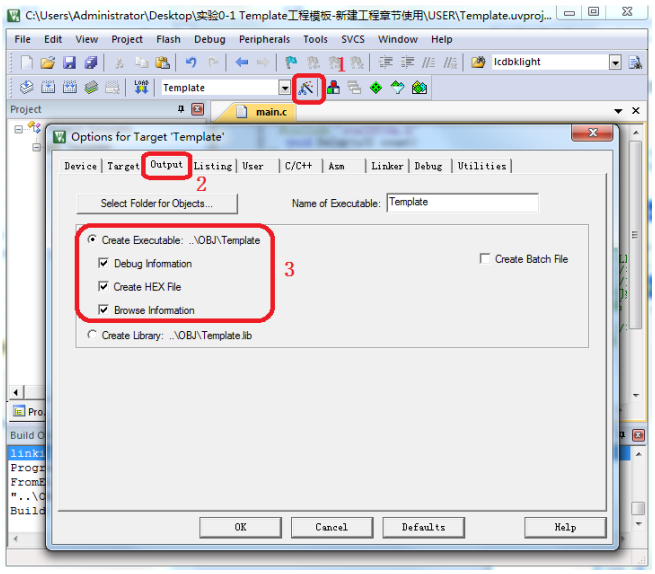
```
#include "stm32f10x.h" void Delay(u32 count)
{
  u32 i=0;
  for(;i<count;i++);
}
int main(void)
{
```

```
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|
    RCC_APB2Periph_GPIOE, ENABLE); // 使能 PB,PE 端口时钟
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; //LED0-->PB.5 端 口 配 置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // 推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //IO 口 速 度 为 50MHz
    GPIO_Init(GPIOB,&GPIO_InitStructure); // 初 始 化 GPIOB.5
    GPIO_SetBits(GPIOB,GPIO_Pin_5);  //PB.5 输 出 高
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; //LED1-->PE.5 推 挽 输 出
    GPIO_  Init(GPIOE, &GPIO_InitStructure); // 初 始 化
    GPIO GPIO_SetBits(GPIOE,GPIO_Pin_5); //PE.5 输 出 高
while(1)
    {
        GPIO_ResetBits(GPIOB,GPIO_Pin_5);
        GPIO_SetBits(GPIOE,GPIO_Pin_5);
        Delay(3000000);
        GPIO_SetBits(GPIOB,GPIO_Pin_5);
        GPIO_ResetBits(GPIOE,GPIO_Pin_5);
        Delay(3000000);
    }
```



（P 2-4-47）工程编译结果

（18）这样一个工程模版建立完毕。下面还需要配置，让编译之后能够生成 hex 文件（将程序下载到单片机内，就需要使用 hex 文件）。同样点击魔术棒，进入配置菜单，选择 Output。然后勾上下三个选项。 其中 Create HEX file 是编译生成 hex 文件，Browser Information 是可以查看变量和函数定义。

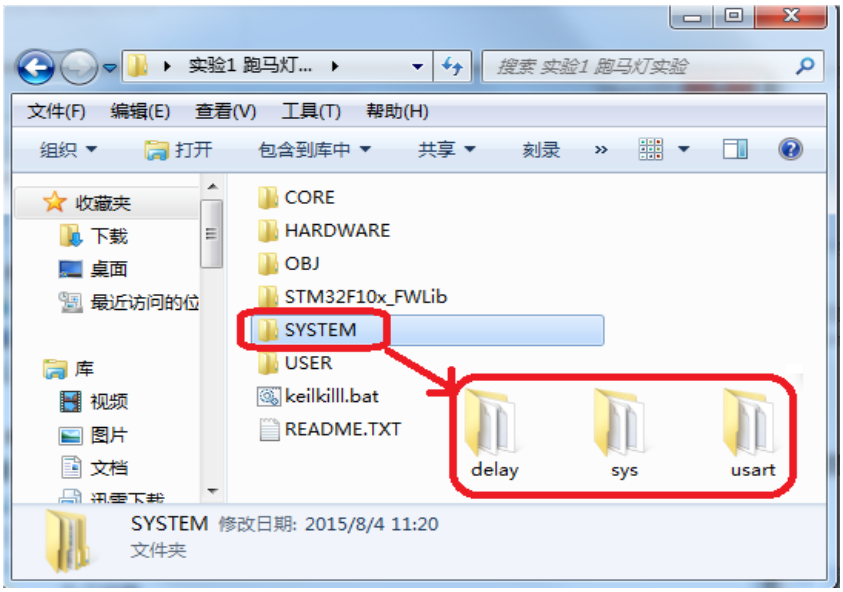


(P 2-4-48)Output 选项卡设置

(19)重新编译代码，可以看到生成了 hex 文件在 OBJ 目录下面，这个文件我们用“Flymcu”下载到 单片机内即可（我们后续会讲到如何下载程序）到这里，一个基于固件库 V3.5 的工程模板就建立了。（在之后的资料里，我们也会给出配置好的工程模板）

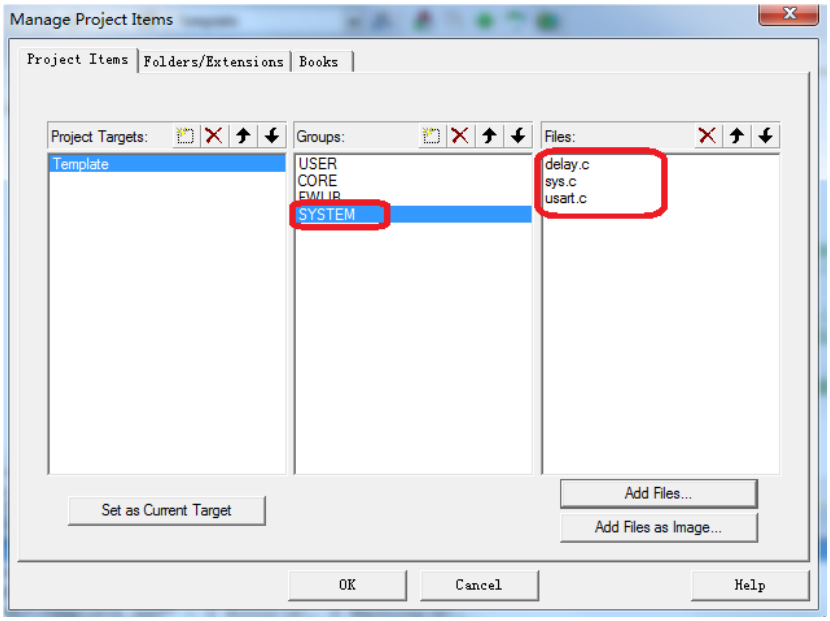
(20)实际上经过前面 19 个步骤，我们的工程模板已经建立完成。但是在 ALIENTEK 即正点原子提供的实验中，每个实验都有一个 SYSTEM 文件夹，下面有 3 个子目录分别为 sys,usart,delay,存放的是我们每个实验都要使用到的共用代码,该代码是由 ALIENTEK 商家编写。我们这里只是引入到工程中，方便后面的实验建立工程。

首先，打开资料中的任意一个工程模板，可以看到下面有一个 SYSTEM 文件夹，比如我们打开实验 1 的工程目录如下：



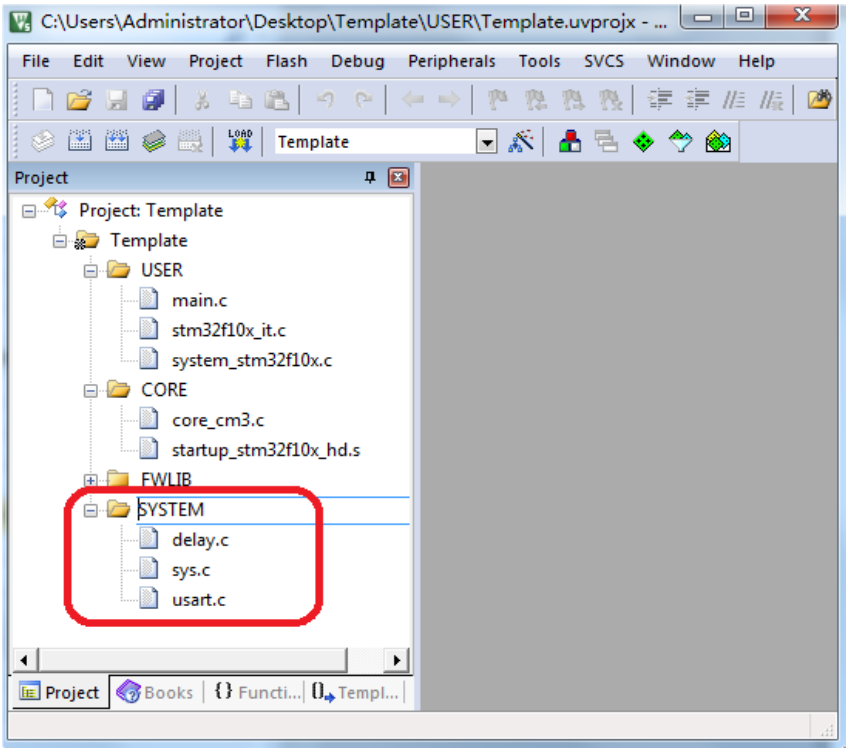
（P 2-4-48） USER 目录文件结构

可以看到有一个 SYSTEM 文件夹，进入 SYSTEM 文件夹，里面有三个子文件夹分别为 Delay（延时函数），sys（中断相关函数），usart（串口通讯函数），每个子文件夹下面都有相应的.c 文件和.h 文件。我们接下来要将这三个目录下面的代码加入到我们工程中去。用我们之前讲解步骤 13 的办法，在工程中新建一个组，命名为 SYSTEM，然后加入这三个文件夹下面的.c 文件分别为 sys.c，delay.c,usart.c，如下图：

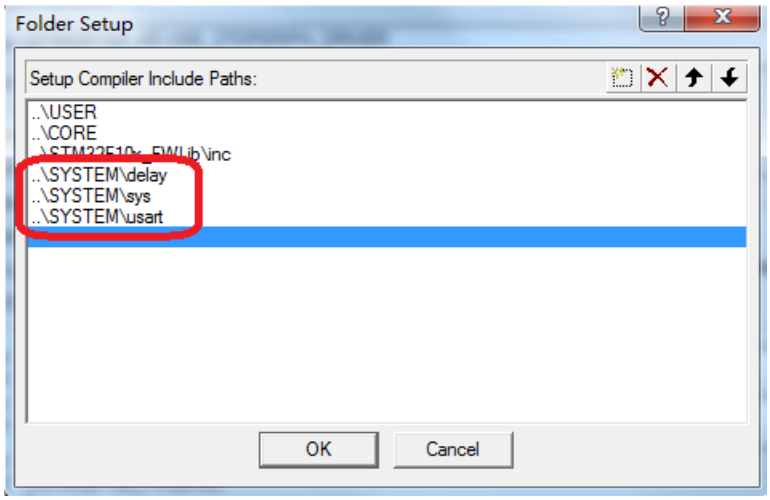


(P 2-4-49) 添加文件到 SYSTEM 分组

然后点击“OK”之后可以看到工程中多了一个 SYSTEM 组，下面有 3 个.c 文件。



（P 2-4-50）添加 SYSTEM 分组后的工程界面



（P 2-4-51）添加头文件路径到 PATH

最后点击 OK。这样我们的工程模板就彻底完成了，这样我们就可以调用 ALIENTEK 提供的 SYSTEM 文件夹里面的函数。我们建立好的工程模板在我们提供的资料里的实验目录里面有，名字为“实验 0-1 Template 工程模板－新建工程章节使用”。（感谢正点原子分享的配置教程。网上也能搜到很多相关教程。我们不用作商用，而是作为内部的资料分享。）

（P 2-4-52）点亮的 LED 灯