STM32 核心板使用说明

第十七届电子设计大赛平台组
电子系科协　翁喆

一、写在前面

前两年的电设一直使用的是 TI 的单片机，比赛所给的支持也仅限于一场讲座以及告诉选手应当下载哪些文件。然而今年换了赞助商，所以重新选择单片机时采用了资料更多的 STM32，同时也为新人们准备一份稍微详细一些的文档。

对于**大神**及**有 STM32 开发经验的同学**，在这里提供你们所需的数据，无需向下继续浏览：

1、单片机：STM32F103RCT6

2、仿真器：JLINK-OB，是仅含 **SW 功能**的 JLINK，驱动与 JLINK 通用

（如果没有驱动，卖家也提供了一份驱动，请前往“**卖家提供**”目录获取）

3、连线方式应该也是一目了然的，板子的资料卖家也提供了一份。

注：本 PDF 创建了**书签**，方便使用。

二、所有提供的文件

1、<文件夹>卖家提供
- 1.1、JLINK OB 使用说明.rar
- 1.2、STM32F103RBT6-2.rar

2、<文件夹>电设组委会提供
- 2.1、<文件夹>Samples
  - 2.1.1、Sample Project Lib.zip
  - 2.1.2、Sample Project.zip
- 2.2、<文件夹>ST 官方文件
  - 2.2.1、固件库_V3.5.0.zip
  - 2.2.2、技术参考手册.pdf
  - 2.2.3、数据勘误手册.pdf
  - 2.2.4、数据手册.pdf
- 2.3、<文件夹>软件
  - 2.3.1、<文件夹>KEIL_MDK 开发环境
- 2.4、<文件夹>说明文档
  - 2.4.1、此文件

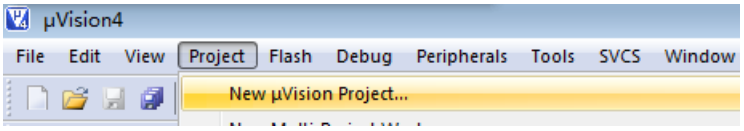
三、编程及烧写

A、准备阶段

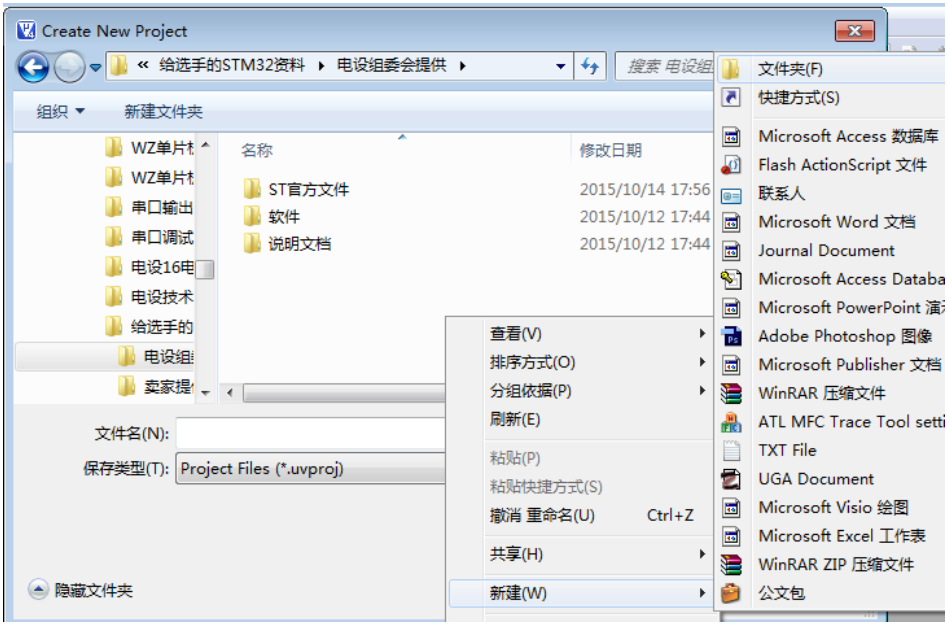
1、请先安装“软件”目录下的 **KEIL_MDK**；
2、安装“卖家提供”的“**JLINK OB** 使用说明.rar”内的**驱动**；（其驱动与 JLINK 一致）
3、软件及驱动的安装不提供支持，遇到问题请**自行上网解决**。

B、KEIL 的使用——新建工程

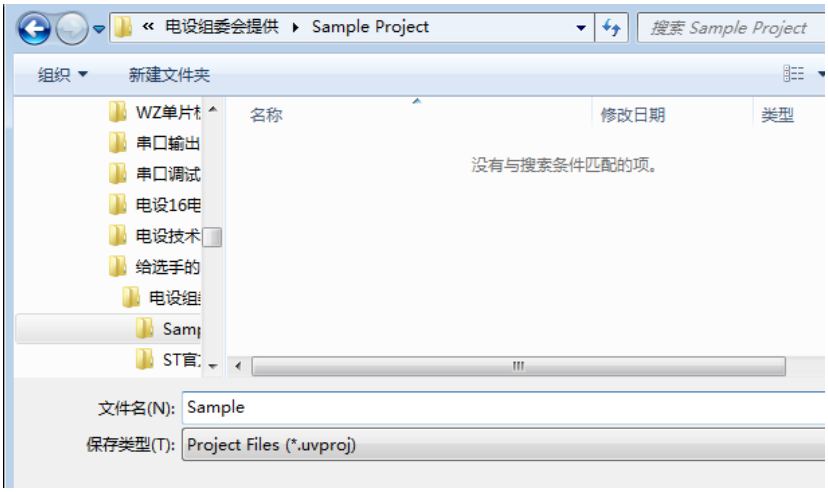
1、打开此软件（Keil uVision4）
2、Project—New µVision Project



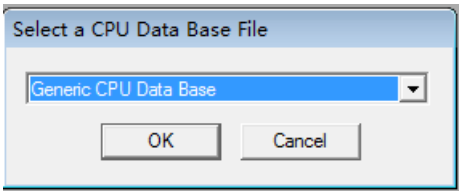
3、选择工程的保存位置，建议**每次**都使用一个**新的文件夹**来保存



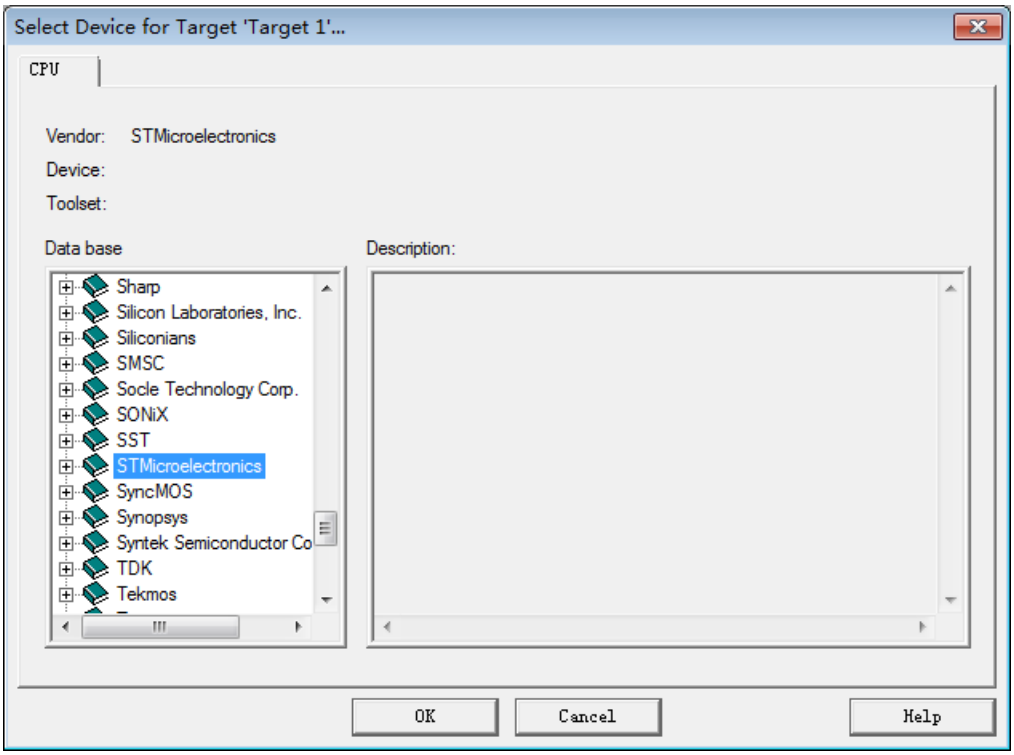
4、在新文件夹中保存工程文件



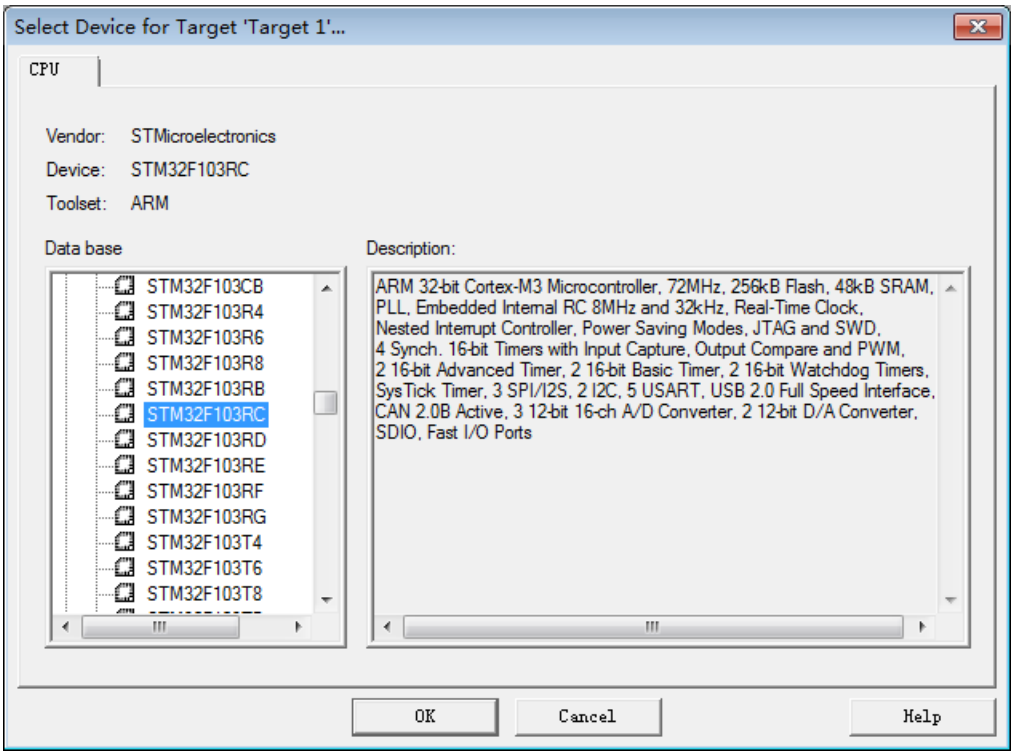
5、如果出现了这个选项请如图选择，否则**跳过此步**



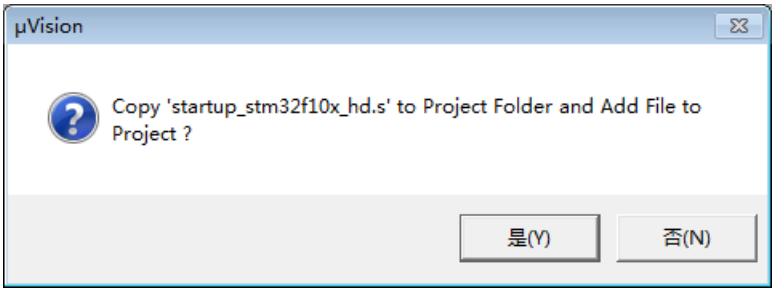
6、在左侧**找到 STM** 并展开



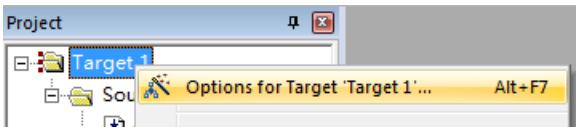
7、找到 **STM32F103RC** 并确认（OK）



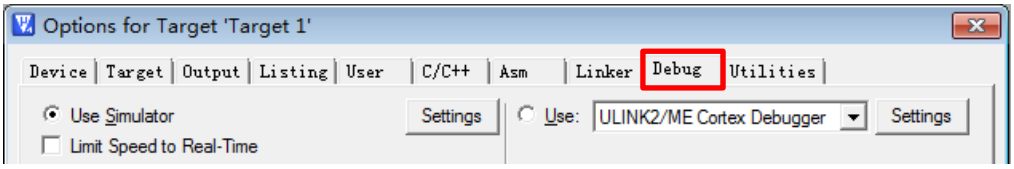
8、选择“是”



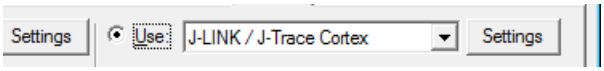
9、左侧“Target”上点右键，“Options...”或者直接 **Alt+F7**



10、点击 **“Debug”** 选项卡

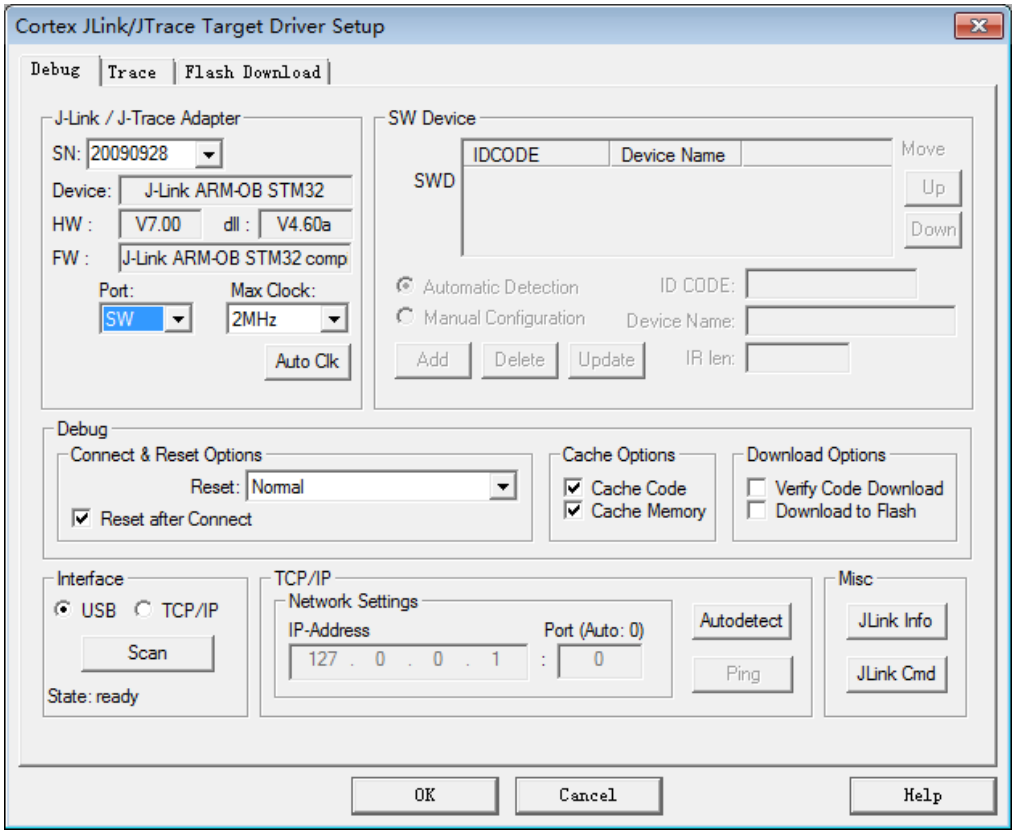


11、**选择 JLINK 并选中**此项（前面那个点）

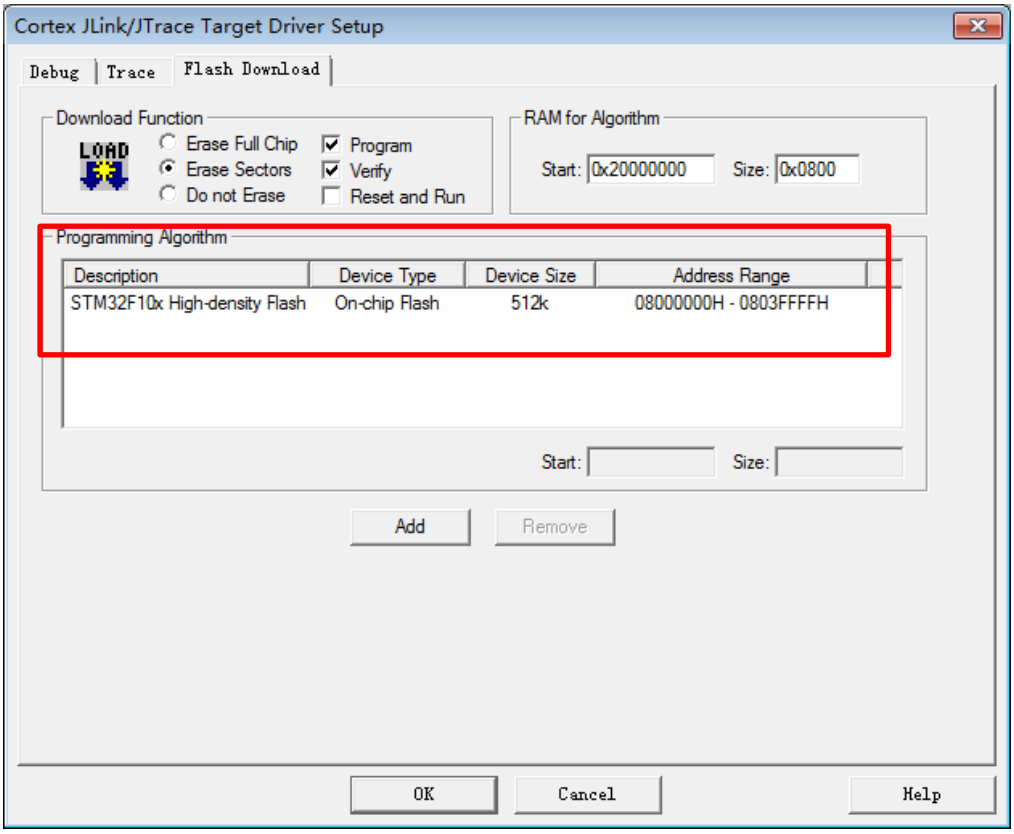


12、此时你要**确保已经插上了 JLINK-OB**，没有插上就现在插上它，然后点那个 Settings，（很显然这里指的是右边这个设置 JLINK 的 Settings），并**将 Port 设置成 SW**（下图被选中的蓝色的地方），先不急着关闭这个窗口，下一步还要用。

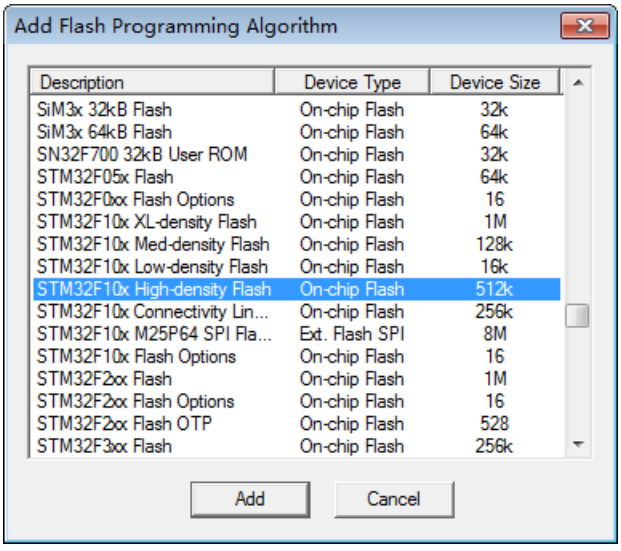
注：**如果**此处此选项**不可选**，请**检查驱动状况**及 JLINK **是否已经连接上**。



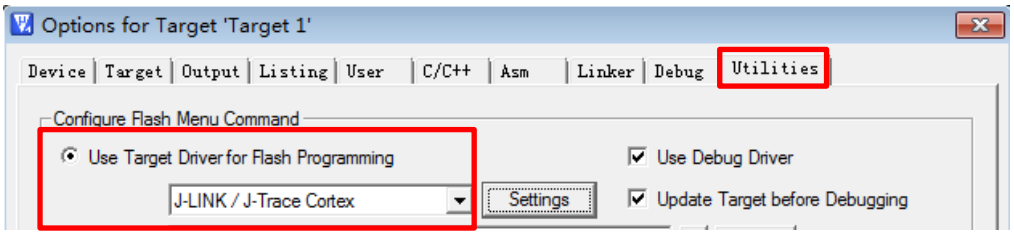
13、点开“Flash Download”选项卡，并确认“Programming Algorithm”内有条目。



14、如果里面没有条目，点击下面的“Add”，并找到 STM32F10x 开头的条目，然后添加进去；如果已经有一项内容了（如上图），请跳过此步。

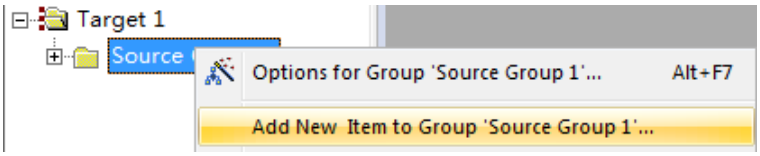


15、此时可以点击“OK”，回到之前的 Options 里，选择“Utilities”选项卡，并选中“Use Target Driver for …”以及选择 JLINK，然后 OK，设置完成。



C、KEIL 的使用——写个简单的代码

1、在工程中，默认会有一个“Source Group 1”，此处可以多建几个分组对代码进行分类管理。本次代码较简单，不进行改名、新建分组等操作。在此处右键，新建一个文件。

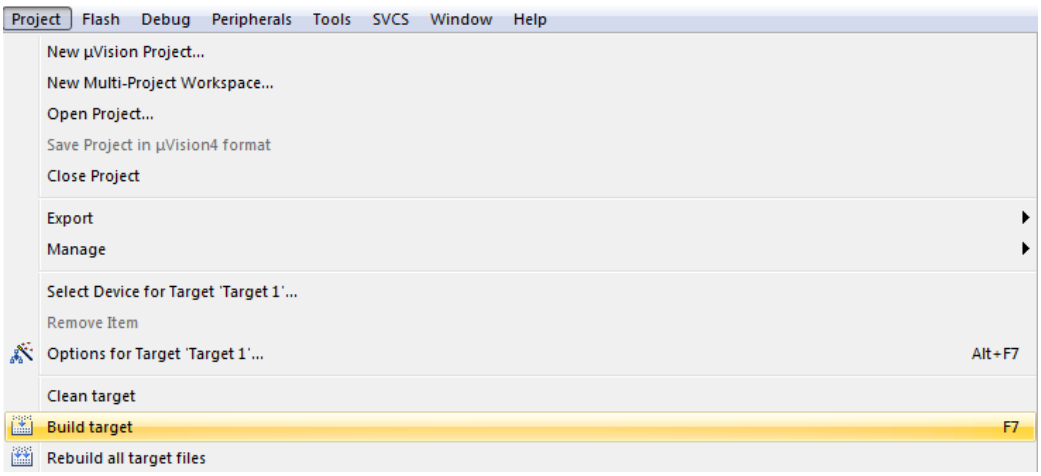


2、选择你需要的文件，此样例使用 C 文件(.c)演示。新建文件后敲入（复制）如下代码：

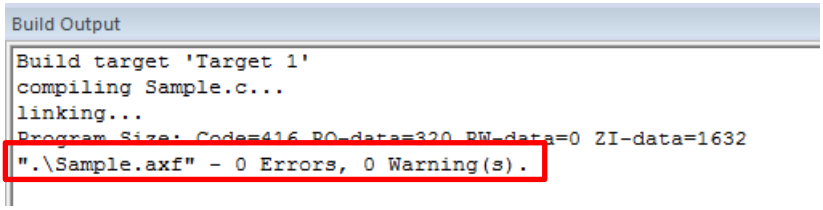
```
#define STM32F10X_HD
#include <stm32f10x.h>

int main()
{
    RCC->APB2ENR|=1<<5;          //使能 PORTD 时钟
    GPIOD->CRL&=0xFFFFF0FF;      //清掉 PD2 原来的设置
    GPIOD->CRL|=0x00000300;      //PD2  推挽输出
    GPIOD->ODR=0x0004;           //PD2=1
    while(1)
    {
        unsigned int t,i,k;
        for(t = 0; t < 20000; t++)
            for(i = 0; i < 100; i++);
        GPIOD->ODR=k ? 0x0000 : 0x0004;
        k = !k;
    }
}
void SystemInit()          //系统初始化函数
{
}
```

3、Project—Build target，或者直接按 F7 进行编译



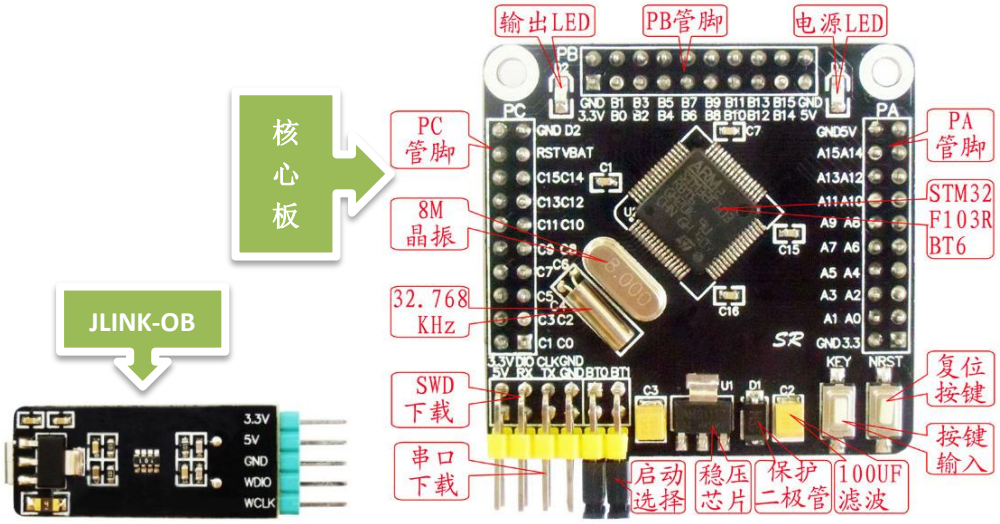
4、确认结果无误



5、至此，已经可以准备烧写程序了，下面是连接一下硬件。

D、硬件连接

1、先来看一下板子和仿真器：



JLINK-OB 左侧通过 micro USB 线连接电脑（推荐使用自己的手机数据线）；

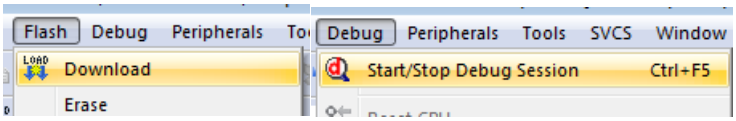
右侧依次有 3.3V,5V,GND,WDIO,WCLK 五根排针，我们不需使用 5V；

核心板左下有 SWD 下载，即 3.3V,DIO,CLK,GND，是不是感觉跟 JLINK 上的线很像？

没错，3.3V-3.3V，GND-GND，WDIO-DIO，WCLK-CLK，如此连接即可。

2、当两边都连接完成后，核心板上的电源 LED（右上）会亮起，可以烧写程序了。

3、在 KEIL 里面选择 Flash-Download 或者 Debug-Start/Stop Debug...(Ctrl+F5)烧写或者开始调试。



如果只是烧写了程序，完成后在板子上按一下复位按键（右下角）；

如果是调试程序，不要忘记按 F5 开始执行。

4、之前在 KEIL 里面写的那个代码的效果是让输出 LED（左上）闪烁，是否成功了呢？

5、关于之前的代码的意思以及如何编写代码，将在后面的内容中讲解。

四、单片机简介

A、单片机的功能

1、什么是单片机？

单片机，顾名思义，就是在一块集成电路上的计算机，它本身包含了计算机的基本组成部分：CPU、RAM/ROM 等。

2、单片机一般有什么功能？

身为计算机，首先肯定具有的功能是计算。而作为单片机，至少有控制 IO 口的能力。而现在的单片机功能更多，一般会集成一些方便的外设，如定时器、UART 等。

3、主办方提供的单片机有哪些功能？

本次比赛提供的单片机为 STM32F103RC，其外设丰富，详细的内容可以在“ST 官方文件”中找到，例如，看看“技术参考手册”的目录：

在这里，每种功能都会介绍对应的特性、功能以及寄存器，ST 官方写得极为详细，最重要的是它是中文的！比去年的英文文档要方便多了。各位同学在开发的时候一定会经常翻看此文件的。如果对其中某项功能有疑问，推荐自行百度，毕竟网上有很多现成的资料，效率还是很高的。

B、寄存器：软件和硬件的桥梁

刚刚提到了技术参考手册会介绍每种功能对应的寄存器，那么寄存器是什么呢？

在使用外设时，寄存器可以视为软件和硬件的桥梁。

对软件而言，这些寄存器和其他的变量也没有特别大的区别，反正都是对地址进行数据的读写操作，只不过是写入的地址不同而已，对于我们而言，如果不关心地址，就当是变量名不一样好了。

而对于硬件，寄存器是时序电路中非常重要的组成部分，其保存的值会直接影响到逻辑状态和工作方式，简而言之，修改了寄存器的值就会改变硬件的运行状态，可以通过修改寄存器达到控制硬件的目的，也可以通过读取寄存器来获取硬件的当前状态。

一句话：像操作变量那样操作寄存器对硬件进行控制。

C、寄存器的说明及使用

1、在哪里可以找到具体有哪些寄存器及其控制的内容？

前面已经提到过了，在“技术参考手册”中：

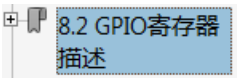
例如，之前曾经往单片机里面烧写过的对 IO 口进行操作的代码，使用了 3 个寄存器：RCC->APB2ENR、GPIOD->CRL、GPIOD->ODR，是需要从手册中找到并设置的。

（General Purpose Input Output（通用输入/输出）简称为 GPIO，即普通的 IO 口）

一开始，我们只是看了各种功能，然后选择使用某种功能，比如 GPIO，之后我们需要知道要操作哪些寄存器才能达到想要的功能，于是，在“技术参考手册”的 GPIO 这一部分，可以看到：

表52　GPIO寄存器地址映像和复位值

偏移	寄存器	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000h	GPIOx_CRL	CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]		CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
	复位值	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
004h	GPIOx_CRH	CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]		CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
	复位值	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
008h	GPIOx_IDR	保留																IDR[15:0]															
	复位值																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00Ch	GPIOx_ODR	保留																ODR[15:0]															
	复位值																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
010h	GPIOx_BSRR	BR[15:0]																BSR[15:0]															
	复位值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
014h	GPIOx_BRR	保留																BR[15:0]															
	复位值																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
018h	GPIOx_LCKR	保留															LCKK	LCK[15:0]															
	复位值																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



具体这些寄存器都是什么功能呢？再仔细一找，有：

点开来一看，发现需要用的是：GPIOx_CRL (x=A..E)和 GPIOx_ODR (x=A..E)，具体每个寄存器的每一位是什么意思已经写得很清楚了，各位可以自己点开看看。

然而仅操作这两个寄存器并没能成功地让它们工作起来，因为我们还没有使能 GPIO，在这里，我们补充一个概念：使用 ARM 内核的单片机（如 STM32），使用 AHB 和 APB 总线（可能不止一条），而挂载在总线上的设备可以单独使能，其使能寄存器归属于 RCC，所以不单要看 GPIO，还要在 6 小容量、中容量和大容量产品的复位和时钟控制(RCC) 里面找到对应的使能寄存器：RCC_AHBENR、RCC_APB2ENR 和 RCC_APB1ENR 的对应位进行使能操作。

现在，各位可以看看我们之前的代码（也可以看下面）具体是做什么的了。

相信各位已经发现了，我们操作的 IO 口是 PD2，但是我们怎么知道那个 LED 连接的是 PD2 呢？

这个就属于 PCB 的设计了，那么理应是由**卖家提供** PCB 的**原理图**的，果然，在卖家提供的**压缩包里**有一个 **STM32F103RBT6-2.pdf**，在此文件里就可以看到 LED 是连接在 PD2 的啦。

2、现在我们重新看看之前的代码

```
#define STM32F10X_HD          // STM32F10X_HD：STM32 High density devices
#include <stm32f10x.h>        // stm32f10x.h：KEIL 自带的头文件，包含寄存器定义

int main()
{
    RCC->APB2ENR|=1<<5;          //使能 PORTD 时钟
    GPIOD->CRL&=0xFFFFF0FF;      //清掉 PD2 原来的设置
    GPIOD->CRL|=0x00000300;      //PD2  推挽输出
    GPIOD->ODR=0x0004;           //PD2=1
    while(1)
    {
        unsigned int t,i,k;
        for(t = 0; t < 20000; t++)
            for(i = 0; i < 100; i++);
        GPIOD->ODR=k ? 0x0000 : 0x0004;
        k = !k;
    }
}
```

（此处已有注释；延时及反转）

是不是基本已经了解了？

D、中断

在单片机的使用过程中，**中断**也算是**很重要的一项功能**了，可以说是一定会用到。

那么中断是什么呢？想象一下，假如你身为一个团队的领导（CPU），带领着大家（外设）在做一项工作，你将任务分配给了每个人之后，自己应该如何做呢？

1、轮询：你时不时地就去每个人那边逛逛看看进度，而大家也只有在你去询问的时候才会告知你当前的状况（或者将当前状况写在某个公共的地方），结果就是你天天在四处乱晃……使用此种方法，就是不断地读取状态寄存器的值来确定是否达到了自己想要的效果。

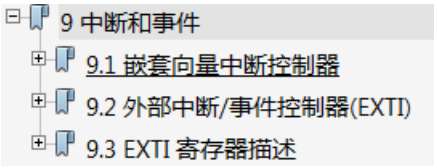
2、中断：你告诉大家如果完成了任务一定要告诉你，于是大家任务完成之后就来找你了，并且同时需要你分配下一步的任务，在这种方式下，当你被团队成员打断时，你可以处理团队成员汇报的结果，同时你需要下达新的指令，而不同的团队成员需要的处理方式不同。使用此种方法，会在某些条件（如串口接收到数据）达成时触发中断，需要**针对不同的中断分别编写中断处理函数**。

当然，实际应用中也许大家还需要别的中断，比如和别的团队（外部电路）合作时，他们传来了一些消息，也会来找你（外部中断），也是可以处理的。

于是，大家已经大致了解了中断的意思，还稍微需要注意一下的是中断的优先级：

当同时有多个中断达成触发条件（很多人来找你）时，通过中断优先级的设置可以确定哪个中断是最优先被处理的。

关于中断的更多内容，可以参考“技术参考手册”：



五、库函数的使用

第十七届电设平台组样车组组长推荐这篇文章：[stm32 固件库 V3.5 中文说明]

A、将库加入工程并使用

1、库文件在哪里

在“ST 官方文件”目录下，有一个“固件库_V3.5.0.zip”，这个压缩包是在 ST 官网上下载的，包含样例和说明等。而实际会用到的文件主要是：

固件库_V3.5.0.zip\STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver 这个目录。

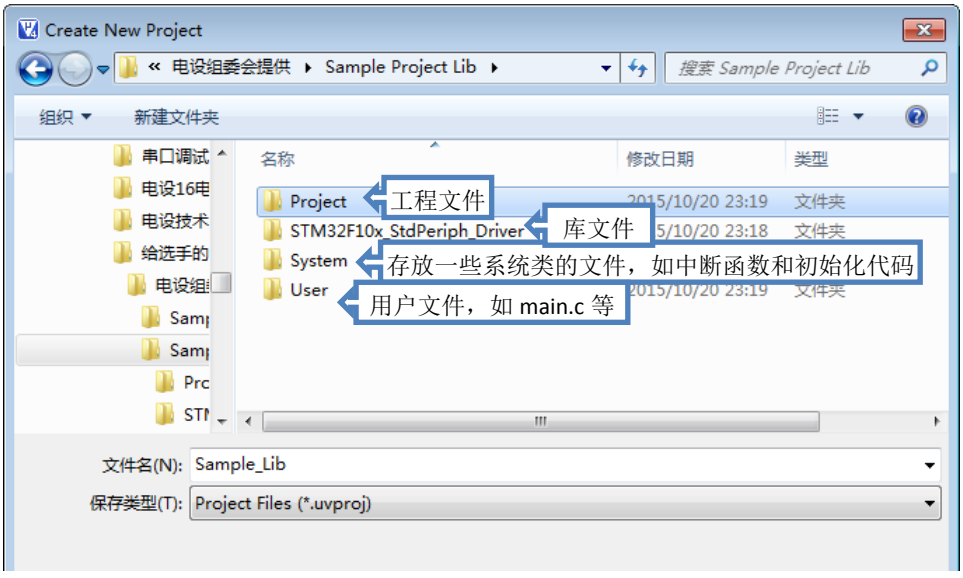
2、库文件的概述

有没有觉得每次都要翻手册查寄存器特别麻烦？是不是觉得应该把某些硬件功能封装成函数调用一下会方便很多？使用**外设库**吧，可以**简化**硬件的配置。

库文件本身就是一堆 **C 代码**以及**对应的头文件**（包含函数声明和结构定义），分别位于上述目录的 src 和 inc 子目录中。**src 目录**中每一个 **c 代码**都**包含**一系列与某种功能相关的**函数**，例如 stm32f10x_gpio.c 就包含各种与 GPIO 操作有关的函数，能够使得开发变得稍微简单一些。当然，为了使用这些函数我们需要声明它们，而**它们的声明**则**在 inc 目录**中**对应的头文件**里。

3、在工程中加入库文件需要的准备

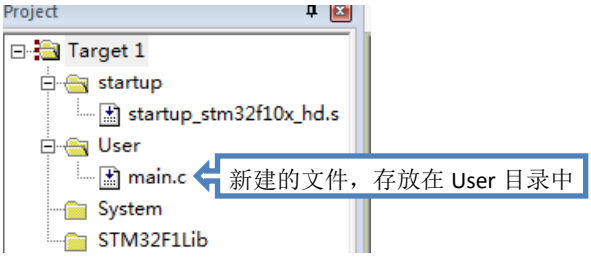
由于整个库所包含的文件较多，编译后产生的中间文件也较多，建议在新建工程时，将**工程文件**本身**保存到**在工程目录内**新建的 Project 文件夹**中（即与三 B4 步不同），同时建议新建一些文件夹**分类存放**文件方便管理。然后就是**将库文件也复制过来**。



由于此样例较简单，只建立了 4 个目录，各位选手在**实际使用中可以建立更多的目录**来方便管理文件。

4、让我们开始操作吧

为了方便对文件的管理，先对**左侧的 Project 栏**的内容进行一些修改，变成这样：



当然，工程的**其他设置同前面**章节所用的**样例**，在此不再赘述。

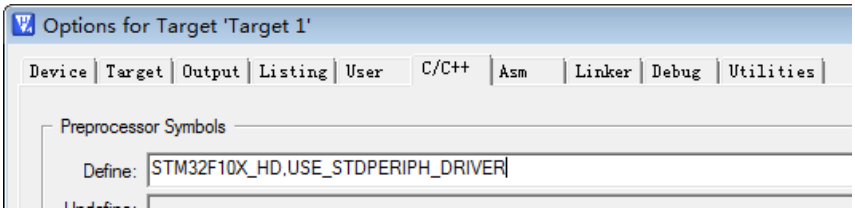
5、将 **ST 官方**给的**模板文件**中的几个文件**复制到 System 目录**下（也可以直接复制官方给的 main.c 模板，这里为了代码的干净并没有使用模块的 main.c）

（模板在 固件库_V3.5.0.zip\STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template 中）

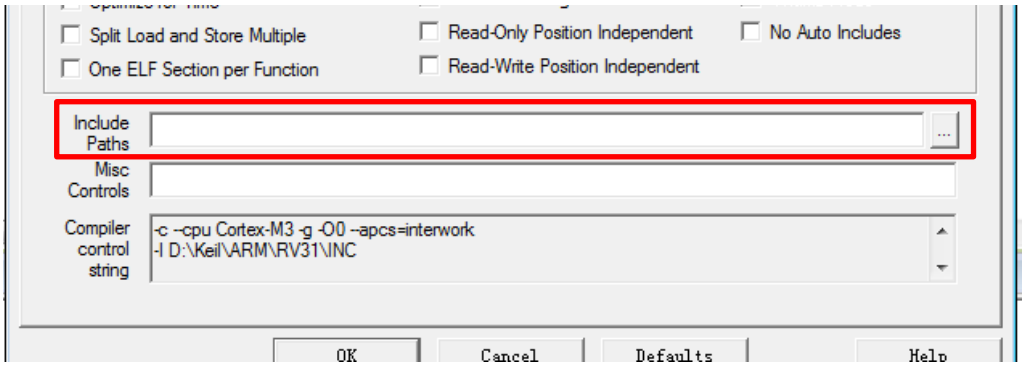
同时**把**这里的**两个.c 文件加入**到**工程**中。

6、**工程**的**选项设置**，打开方式同三 B9

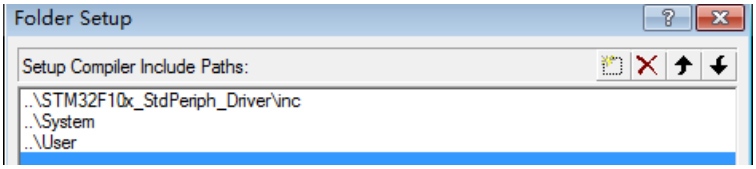
在前面的章节中使用的样例代码里曾经有一句“#define STM32F10X_HD”，**当时**是因为**只需要**在**一个文件中**定义所以直接写在了代码里，然而**现在**需要让它**变成全局**的**定义**，于是我们又一次打开了**工程选项（Alt+F7）**，在**“C/C++”**选项卡中，有一个 **Define**，我们把它加在这里，**同时**也**加入 USE_STDPERIPH_DRIVER**，以顺利地使用外设库。



同时，由于目录的增加，许多**头文件**被放在了**子目录**中，这样会在**编译时找不到头文件**，所以**在**下面的 **Include Paths 里面加入需要的**包含头文件的**目录**。

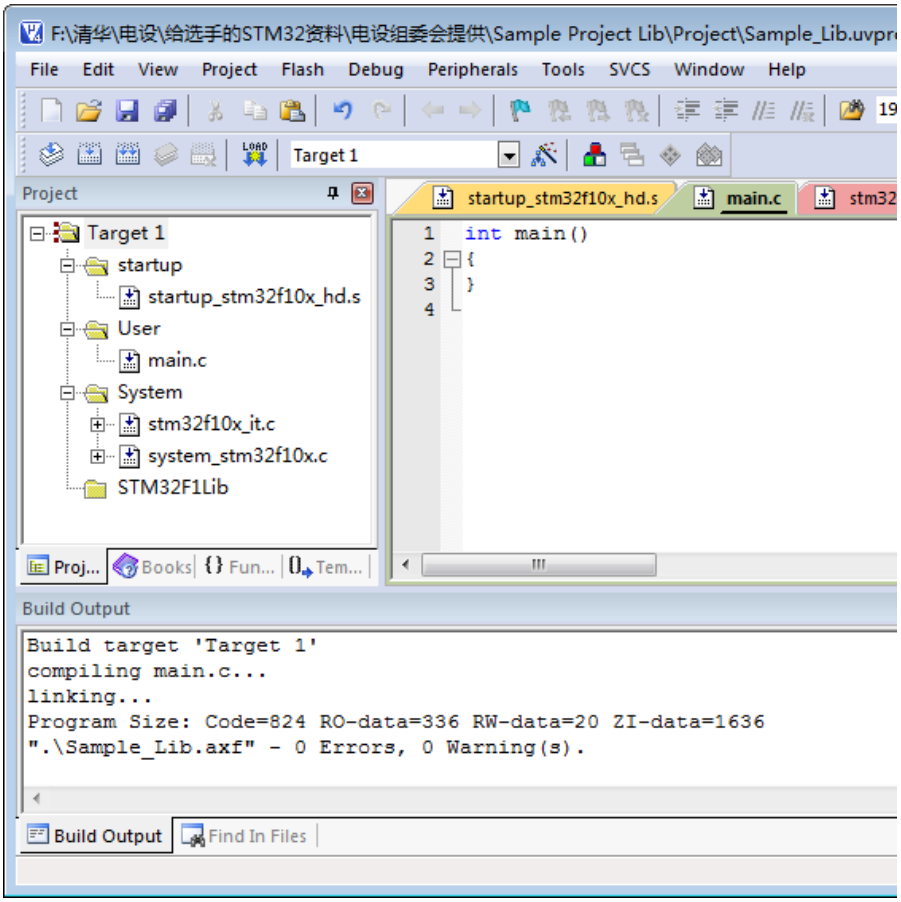


我们可以用右边的“…”方便地添加文件夹。

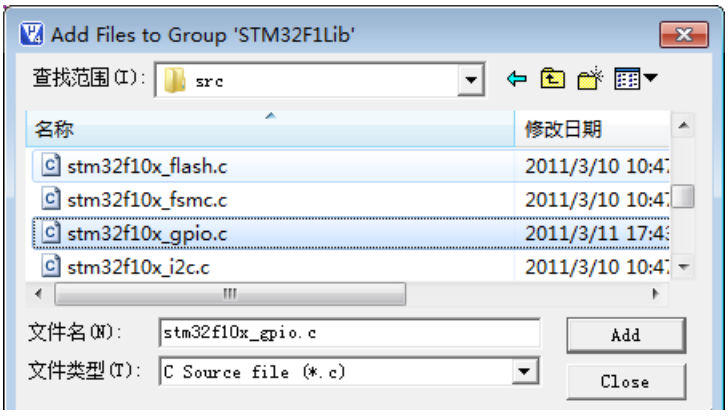


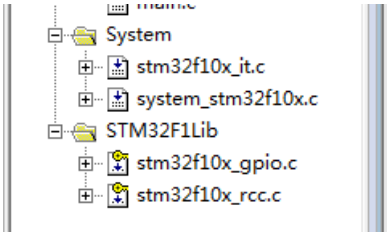
由于只有 3 个目录会包含头文件（Project 目录不放源代码，库文件的头文件在 inc 目录中），所以变成了上面的情况。然后此步配置就完成了。

7、先编译一下试试，在 main.c 中写一个空的 main 函数，编译通过。



8、加入一个库文件并准备编写代码，这里我们先选择使用前面使用过的 GPIO 进行演示，将库目录里的 src 目录中的 stm32f10x_gpio.c 和 stm32f10x_rcc.c 加入工程（基本上无论使用何种外设都是需要 RCC 的，之前已有解释）。





如果没有意外，此时应该仍然能编译通过，于是我们可以使用这两个 C 文件中定义的函数了。

9、使用库函数编写代码，将如下内容复制进 main.c 中，编译并烧写进板子中：

```
#include <stm32f10x_conf.h>

int main()
{
	GPIO_InitTypeDef GPIO_InitStructure;//GPIO 初始化所用结构

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);//使能 GPIOD

	//引脚 2，可以使用位或连接，例如 GPIO_Pin_2|GPIO_Pin_3，可以同时初始化多个引脚
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//输出，推挽
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
	GPIO_Init(GPIOD, &GPIO_InitStructure);//此初始化作用于 GPIOD，于是 PD2 被初始化了

	while(1)
	{
		unsigned int t,i,k;
		for(t = 0; t < 20000; t++)
			for(i = 0; i < 100; i++);
		k = !k;
		//将 PD2 设置成 k 的值，这里使用的类型本来应该是 BitAction，使用了强制转换
		GPIO_WriteBit(GPIOD, GPIO_Pin_2, (BitAction)k);
	}
}
```

在这里，我们直接将之前的代码改写成了库函数形式，为了方便各位观察，对其它部分所做的修改尽可能的少，虽然看上去代码量变多了，但是比之前容易理解多了不是吗？甚至连注释都不怎么需要了。

不过，当烧写进板子之后，诸位可能会发现：闪烁的频率明显比之前的代码高了，这又是为什么呢？

还记得当初我们写了一个空的 **void SystemInit()函数**吗？叫这个函数名的函数会在单片机上电时被自动调用，而当时我们使用单片机默认的设置（即寄存器默认值），所以在函数中什么也没做。然而这次我们并没有写这个函数，这又是为什么呢？因为我们从模板中复制了 **system_stm32f10x.c** 文件，此文件中已经含有此函数了，而且它设置的时钟频率较高，所以代码的执行速度明显加快了，于是闪烁就变快了。各位可以自行修改 system_stm32f10x.c 里面的函数内容以达到自己想要的效果。

10、关于 **stm32f10x_conf.h** 文件

诸位可能也注意到了，之前复制过来的 stm32f10x_conf.h 文件被包含在了我们的 main.c 中，那么这个文件是做什么的呢？打开来一看，发现其实就是包含了几乎所有的库的头文件，其中有这么一句话：

```
/* Uncomment/Comment the line below to enable/disable peripheral header file inclusion */
```

也就是说，下面的那些头文件如果用不到是可以注释掉的，比如此例中我们可以只保留 stm32f10x_gpio.h 和 stm32f10x_rcc.h 就可以了。

于是，我们只需在代码中包含此头文件就相当于包含了所有需要使用的库头文件，不过这还不是我们必须使用这个文件的理由，真正的原因是所有的库的 C 代码本身都会需要这些头文件，而当我们定义了 USE_STDPERIPH_DRIVER 之后，在 stm32f10x.h 中就会自动包含此头文件，而所有的库也都会包含 stm32f10x.h，于是间接地包含了 stm32f10x_conf.h，以此来保证所有的结构和函数都被正确地声明了。

B、ST 官方提供的样例

当我们已经配置到了可以运行库函数的程度时，本说明也就基本接近尾声了，因为无论需要什么功能，都可以直接上网搜索代码了，现在无论是寄存器还是库函数各位都已经会使用了，具体每个功能的使用方式如果真要一点一点写就又能写出一本“技术参考手册”那么厚的书来，所以还是各位自行研究比较好，毕竟资料还是很多的。当然，之前也有提到过，ST 官方提供的这个库是包含样例和说明的，那么样例在哪里呢？当然还是在压缩包里啦：固件库_V3.5.0.zip\STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Examples 一看到这个目录叫 Examples，相信各位也了解它是做什么用的了。官方提供的样例一般只有几个文件，而这几个文件之前我们都接触过，而且样例中还会包含 ReadMe 提供基本的信息，如果担心出问题就直接替换掉工程中的对应文件就好啦~关键是环境已经配置好，所以这几个文件加入得非常顺利。

六、其他的一些信息

A、每种功能对应的引脚是哪些？

引脚定义在“数据手册”中，第 20 页有引脚定义的表，注意我们此次的单片机封装是 LQFP64。

B、可能会用到哪些功能？

一般来说，肯定会用到的是 UART（USART），可能会用到的有定时器（TIM）、ADC、DMA、EXTI、IIC(I2C)、SPI 等，对此不了解的可以自己搜索一下。

C、之前的例子们都在哪里呢？

本说明中所创建的工程都在放在了“电设组委会提供\Samples”目录下，供参考。

七、后记

这份说明可以说写得并不是很全面，很多方面没有深入，正如一开始所说，这是为新人们准备的一份稍微详细一些的文档，至少能让新人们成功地进行第一次开发。之后的内容就看各人的努力了，毕竟到了这个水平已经可以稍微看懂各方面的资料了，不至于看到代码也不知道是什么情况，所以这篇文档就到此为止吧，说不定明年内容会变得更加充实呢~