Lab3 按键控制 LED 灯_IO 输入操作

一、实验主要内容：

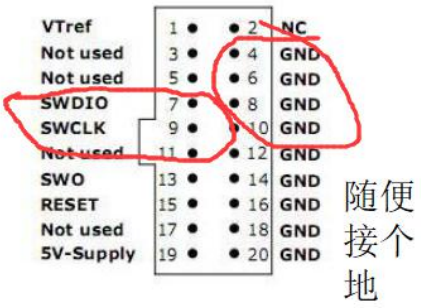
1. 熟悉按键输入机制，通过按键输入控制 LED；

2. 检查 lab2 作业 LED 模拟多种状态功能（仿真和下板实验）；【本周完成检查，未接受检查者检查部分 30%不得分】。

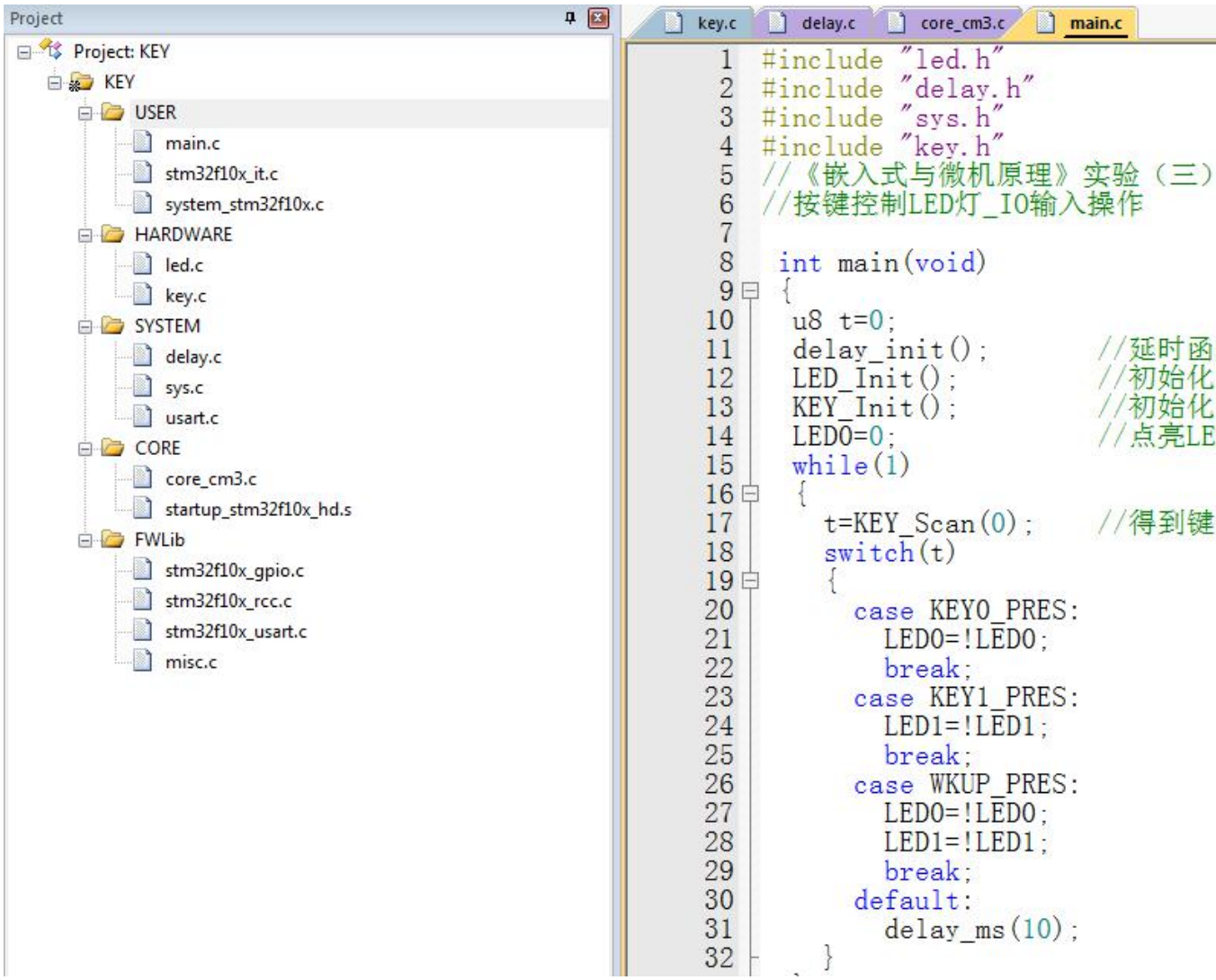
二、创建工程

采用手动导入的方式建立工程，添加所有需要的文件，参考 lab1、lab2 文档。
注意：提供的部分文件中的主要函数可能不完整，请仔细对照文档及《STM32 不完全手册》

三、下载器连接：不要在通电情况下拔插杜邦线！！！



四、工程说明



1. main.c 文件是自行添加的源文件，用来编写 main 函数。

2.misc.c ： This file provides all the miscellaneous firmware functions (add-on to CMSIS functions).

提供了 NVIC(中断向量嵌套)的外设驱动

3. stm32f10x_gpio.c ： This file provides all the GPIO firmware functions.

提供了关于 IO 口操作的函数

4. stm32f10x_rcc.c : This file provides all the RCC firmware functions.

提供了处理内部时钟相关函数文件

5. startup_stm32f10x_hd.s，system_stm32f10x.c ：

启动文件

五、源代码说明

#include "led.h"

```
#include "delay.h"
#include "sys.h"
#include "key.h"
//《嵌入式与微机原理》实验（三）
//按键控制 LED 灯_IO 输入操作

 int main(void)
 {
     u8 t=0;
     delay_init();            //延时函数初始化
     LED_Init();              //初始化与 LED 连接的硬件接口
     KEY_Init();              //初始化与按键连接的硬件接口
     LED0=0;                  //点亮 LED
     while(1)
     {
         t=KEY_Scan(0);       //扫描监听得到键值
         switch(t)
         {
             case KEY0_PRES:
                 break;
             case KEY1_PRES:
                 break;
             case WKUP_PRES:
                 break;
             default:

         }
     }
}
```

其中主要函数包括 delay_init, LED_Init, KEY_Init, KEY_Scan 等。

```
void delay_init()
{
#if SYSTEM_SUPPORT_OS                                   //如果需要支持 OS.
     u32 reload;
#endif
     SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);   //选择外部时钟
HCLK/8
     fac_us=SystemCoreClock/8000000;                    //为系统时钟的 1/8
#if SYSTEM_SUPPORT_OS                                   //如果需要支持 OS.
```

```
    reload=SystemCoreClock/8000000;                 //每秒钟的计数次数  单位为 M
    reload*=1000000/delay_ostickspersec;            //根据 delay_ostickspersec 设定溢出时间
                                                    //reload 为 24 位寄存器,最大值:16777216,在 72M 下,约合 1.86s 左右
    fac_ms=1000/delay_ostickspersec;                //代表 OS 可以延时的最少单位


    SysTick->CTRL|=SysTick_CTRL_TICKINT_Msk;        //开启 SYSTICK 中断
    SysTick->LOAD=reload;                           //每 1/delay_ostickspersec 秒中断一次
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;         //开启 SYSTICK

#else
    fac_ms=(u16)fac_us*1000;                        //非 OS 下,代表每个 ms 需要的 systick 时钟数
#endif
}

void LED_Init(void)
{

 GPIO_InitTypeDef    GPIO_InitStructure;

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOD, ENABLE);      //使能 PA,PD 端口时钟

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;              //LED0-->PA.8  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(GPIOA, &GPIO_InitStructure);                 //根据设定参数初始化 GPIOA.8
 GPIO_SetBits(GPIOA,GPIO_Pin_8);                        //PA.8  输出高

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;              //LED1-->PD.2  端口配置，推挽输出
 GPIO_Init(GPIOD, &GPIO_InitStructure);                 //推挽输出 ，IO 口速度为 50MHz
 GPIO_SetBits(GPIOD,GPIO_Pin_2);                        //PD.2  输出高
}

void KEY_Init(void)
{
```

```
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOC,ENABLE);//使能 PORTA,PORTC 时钟

    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);//关闭 jtag，使能 SWD，可以用 SWD 模式调试

    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_15;//PA15
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU; //设置成上拉输入
    GPIO_Init(GPIOA, &GPIO_InitStructure);//初始化 GPIOA15

    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_5;//PC5
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU; //设置成上拉输入
    GPIO_Init(GPIOC, &GPIO_InitStructure);//初始化 GPIOC5

    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_0;//PA0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //PA0 设置成输入，默认下拉
    GPIO_Init(GPIOA, &GPIO_InitStructure);//初始化 GPIOA.0

}

//按键处理函数
//返回按键值
//mode:0,不支持连续按;1,支持连续按;
//返回值：
//0，没有任何按键按下
//KEY0_PRES，KEY0 按下
//KEY1_PRES，KEY1 按下
//WKUP_PRES，WK_UP 按下
//注意此函数有响应优先级,KEY0>KEY1>WK_UP!!
u8 KEY_Scan(u8 mode)
{
    static u8 key_up=1;   //按键按松开标志
    if(mode)key_up=1;   //支持连按
    if(key_up&&(KEY0==0||KEY1==0||WK_UP==1))
    {
        delay_ms(10);//去抖动
        key_up=0;
        if(KEY0==0)return KEY0_PRES;
        else if(KEY1==0)return KEY1_PRES;
        else if(WK_UP==1)return WKUP_PRES;
```

```
    }else if(KEY0==1&&KEY1==1&&WK_UP==0)key_up=1;
    return 0;// 无按键按下
}
```

GPIO 库函数说明： 参见 STM32 固件库.PDF P122 – P133

Table 179. GPIO 库函数

函数名	描述
GPIO_DeInit	将外设 GPIOx 寄存器重设为缺省值
GPIO_AFIODeInit	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_ReadInputDataBit	读取指定端口管脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 管脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 管脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定管脚的映射
GPIO_EXTILineConfig	选择 GPIO 管脚用作外部中断线路

Table 182. 函数 GPIO_Init

函数名	GPIO_Init
函数原形	void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：GPIO_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_InitTypeDef structure

GPIO_InitTypeDef 定义于文件***"stm32f10x_gpio.h"***：

```
typedef struct
{
u16 GPIO_Pin;
GPIOSpeed_TypeDef GPIO_Speed;
GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
```

Table 184. GPIO_Speed 值

GPIO_Speed	描述
GPIO_Speed_10MHz	最高输出速率 10MHz
GPIO_Speed_2MHz	最高输出速率 2MHz
GPIO_Speed_50MHz	最高输出速率 50MHz

Table 185. GPIO_Mode 值

GPIO_Speed	描述
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮空输入
GPIO_Mode_IPD	下拉输入
GPIO_Mode_IPU	上拉输入
GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_Out_PP	推挽输出
GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_AF_PP	复用推挽输出

Table 187. 描述了函数 GPIO_StructInit

Table 187. 函数 GPIO_StructInit

函数名	GPIO_StructInit
函数原形	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	把 GPIO_InitStruct 中的每一个参数按缺省值填入
输入参数	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 188. 给出了 GPIO_InitStruct 各个成员的缺省值

Table 188. GPIO_InitStruct 缺省值

成员	缺省值
GPIO_Pin	GPIO_Pin_All
GPIO_Speed	GPIO_Speed_2MHz
GPIO_Mode	GPIO_Mode_IN_FLOATING

例：

```
/* Initialize the GPIO Init Structure parameters */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_StructInit(&GPIO_InitStructure);
```

Table 193. 描述了 GPIO_SetBits

Table 193. 函数 GPIO_SetBits

函数名	GPIO_SetBits
函数原形	void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	设置指定的数据端口位
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待设置的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the GPIOA port pin 10 and pin 15 */
GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

Table 194. 描述了 GPIO_ResetBits

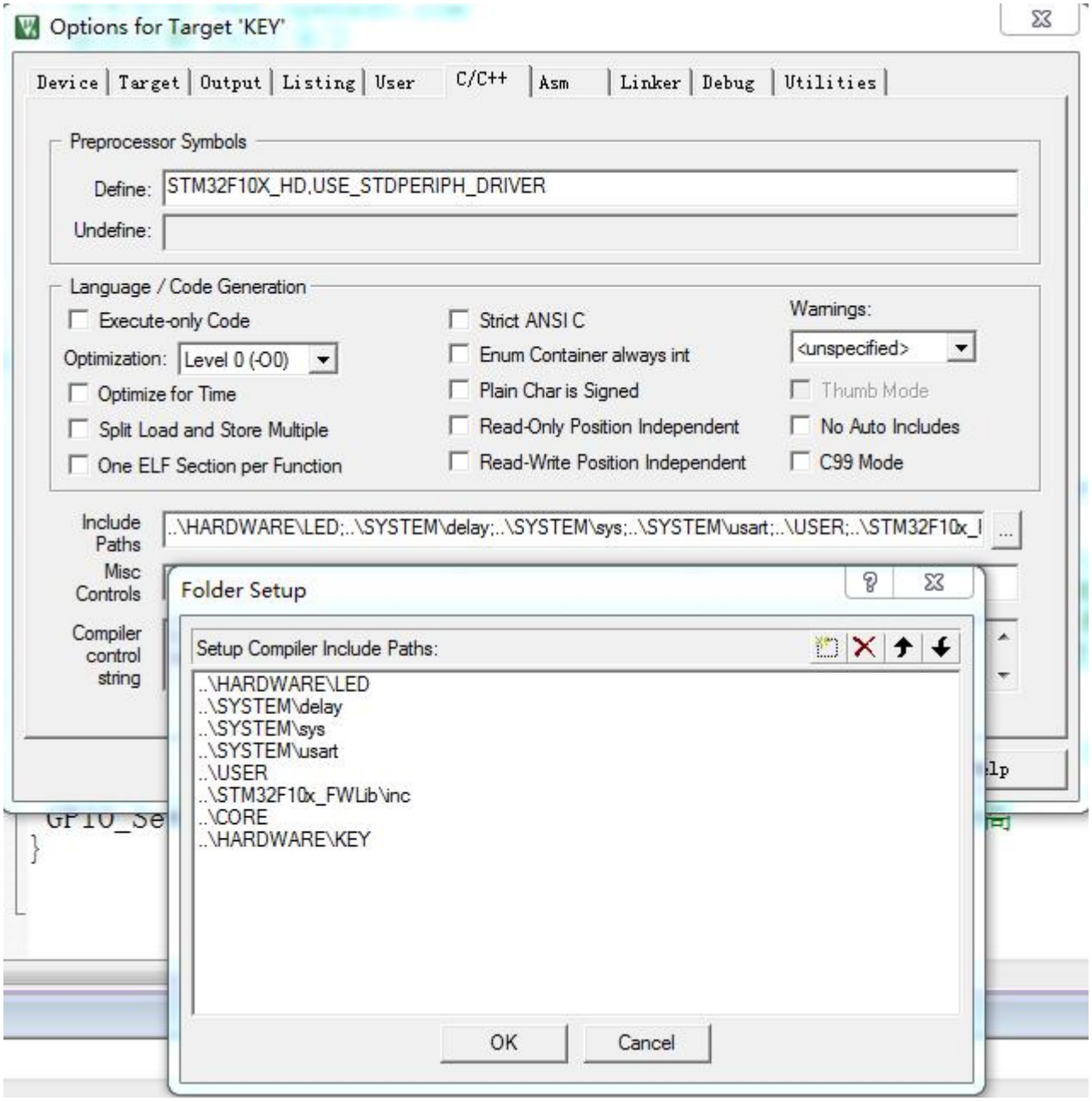
Table 194. 函数 GPIO_ResetBits

函数名	GPIO_ResetBits
函数原形	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待清除的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clears the GPIOA port pin 10 and pin 15 */
GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

注意：1.要在 options for target1 中的 c/c++中添加这几个文件的路径，不然会提示找不到文件。



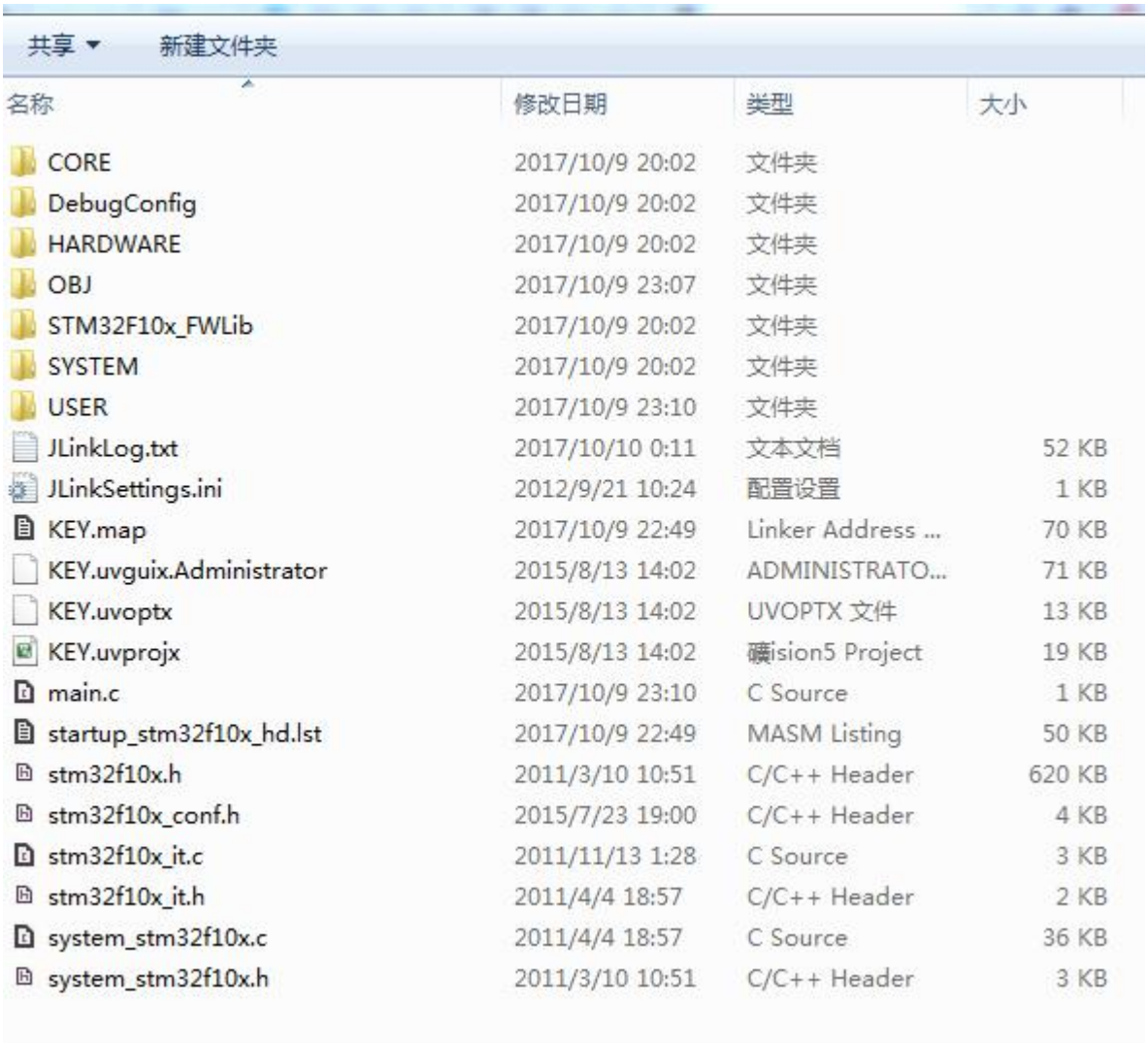
2. 按键消抖

在编写程序时需要注意的是，我们使用的开关为机械弹开关，在按键按下时不会马上形成稳定的电路，会伴随有一连串的抖动，一般时间为 5ms 到 10ms，按键的抖动会引起按键被误读多次。为了确保 CPU 对按键的一次闭合只做一次处理，必须去除按键的抖动。一般来说一个简单的按键消抖就是先读取一次按键的状态，如果得到按键按下之后，延时 10ms 左右，再次读取一次按键的状态，如果按键还是按下状态，那么说明按键已经按下。

六、作业

本次作业不要求提交，会在下次 lab 合并检查

1. 重新建立工程，以学号为工程名称
（1）按照 lab1 导入方式建立工程；
（2）调整目录结构，将用户自建放在项目根目录下（目录结构如下），注意引用库文件的调用；



2. 熟悉修改按键操作指令，如初始时 DS0 和 DS1 同时处于不亮状态，按 KEY0 时 DS0 亮，KEY1 时 DS1 亮，WK_UP 时 DS0 和 DS1 同时不亮，可自行设计其他的按键控制 LED 灯状态的策略。