WHEELTEC
INTELLIGENT TECHNOLOGY

东莞市微宏智能科技有限公司

淘宝店铺：minibalance.taobao.com

网址：www.wheeltec.net

STM32F103C8 核心板例程实验手册

推荐关注我们的公众号获取更新资料

版本说明：

版本	日期	内容说明
V1.0	2020/06/18	第一次发布

序言

本实验手册是 STM32 单片机的入门教程，从点亮 LED 灯到读取 ADC，一步步指导大家进行学习，且每一个例程讲的都是实用性很强的功能，是大部分项目都必须用到的功能。本实验手册基于 STM32F103C8T6 芯片，提供全套源码、原理图、芯片资料及全套软件工具。

本实验手册共有 11 节，分别是新建工程、点亮 LED、按键控制 LED、串口通信、外部中断、定时中断、定时器的 PWM 输出模式、PWM 输出控制舵机、定时器的输入捕获模式、定时器的编码器模式、ADC 采集。

目录

1. 新建工程

1.1 新建工程及相关文件准备

① 创建项目与相关文件夹

新建一个文件夹【NewProject】，用于保存工程库文件，同时在文件夹【NewProject】下创建一个文件夹【USER】。

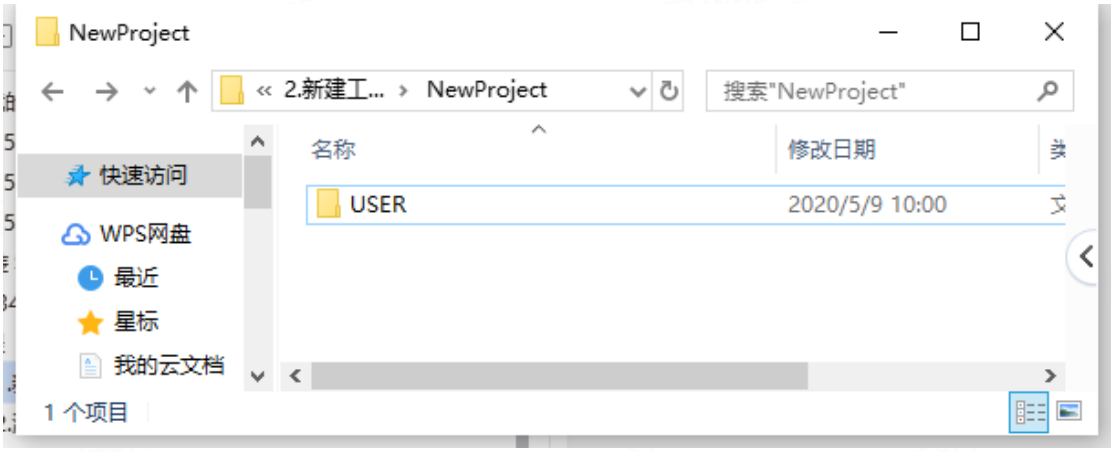

图 1-1-1 创建文件夹

打开 Keil5 软件后，点击【项目】，选择【新的 uVision 项目】

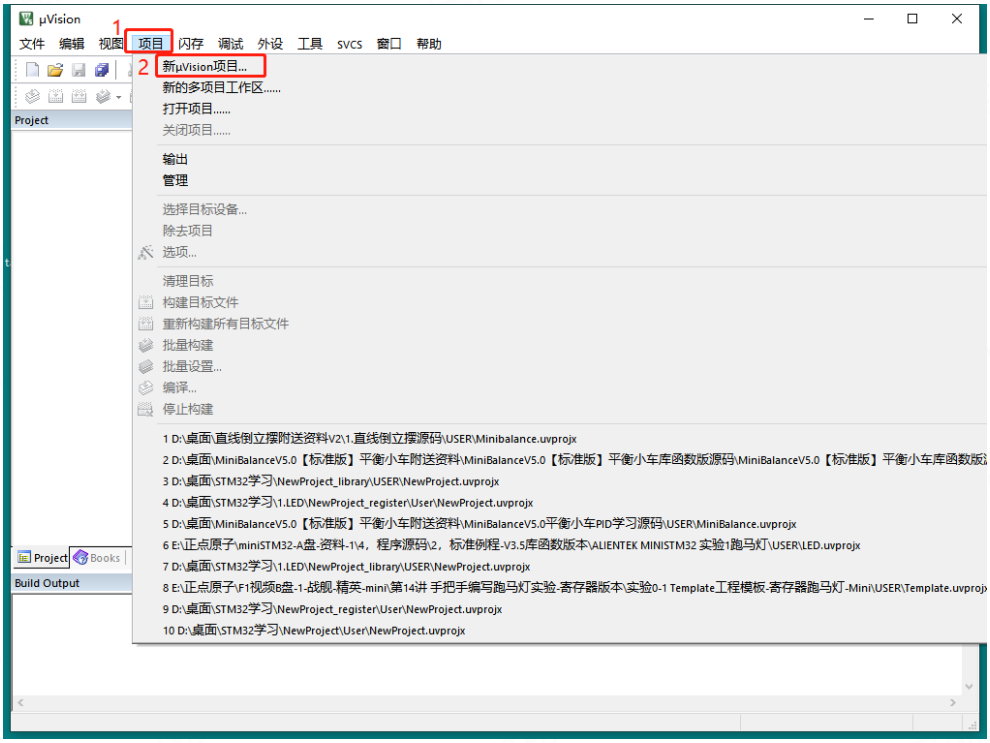

图 1-1-2 新建项目

在第一步新建的文件夹【NewProject】下的文件夹【USER】下创建工程文件，文件名为【NewProject】，点击【保存】。

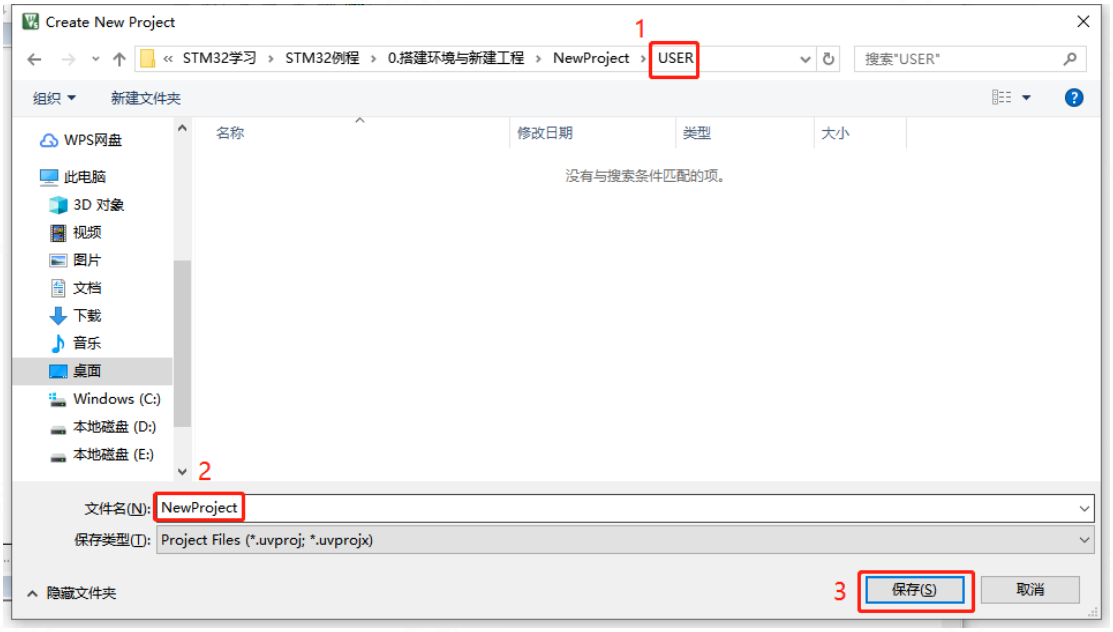

图 1-1-3 在【USER】文件夹下保存工程【NewProject】

② 选择工程芯片

选择 CPU，即要上传程序的 STM 芯片型号，这里以 STM32F103RTC6 为例，STMicroelectronics→STM32F1Series→STM32F103→STM32F103RC，点击【OK】。

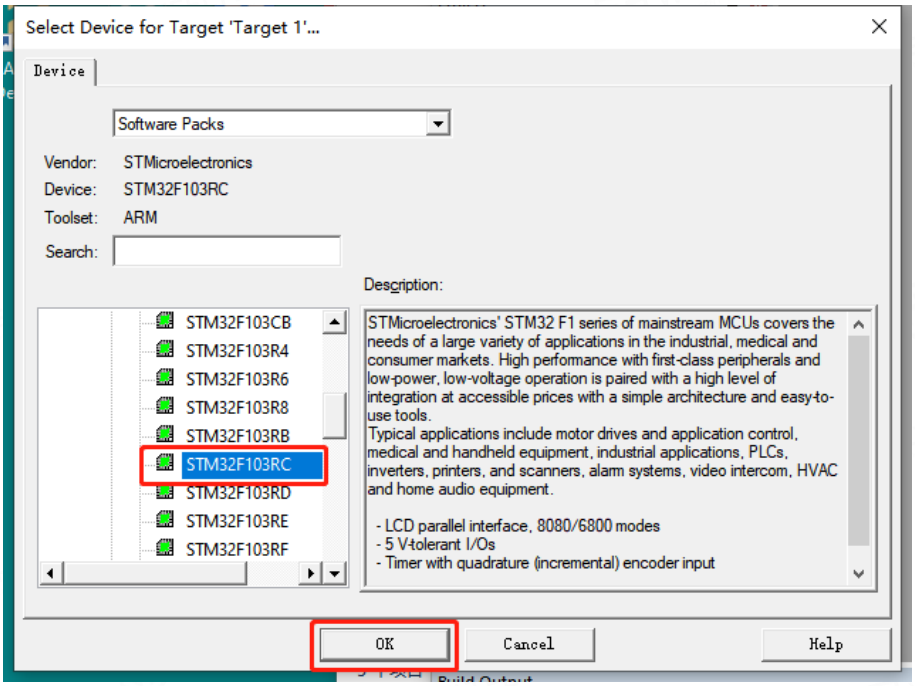

图 1-1-4 选择芯片型号

点击【Cancel】。

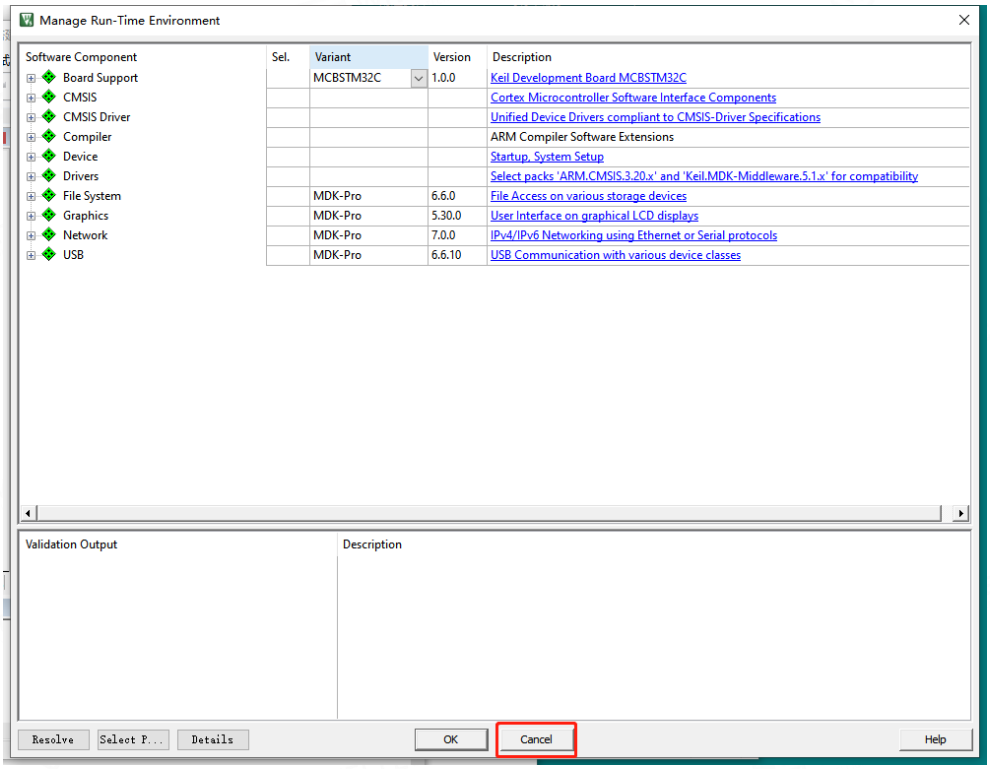

图 1-1-5 点击取消

③ 复制相关库文件

在文件夹【NewProject】下新建三个文件夹【CORE】、【OBJ】、【STM32F10x_FWLib】，文件夹【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 hex 文件，文件夹【STM32F10x_FWLi】用来存放 STM 官方提供的库函数源码文件。

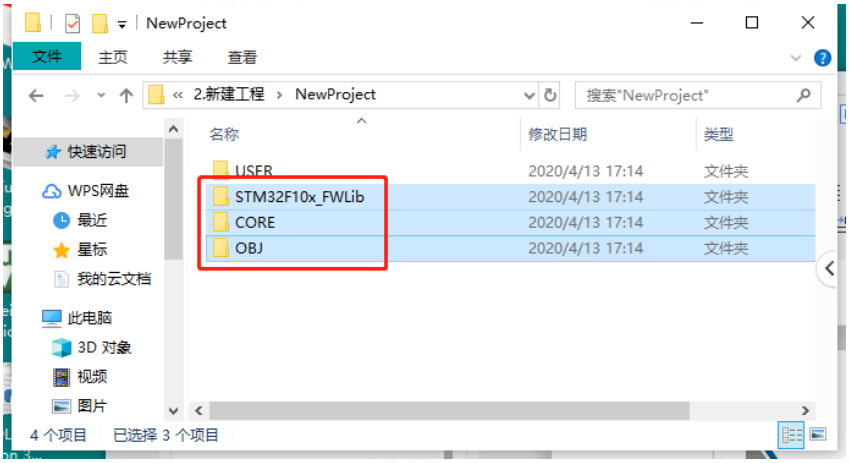

图 1-1-6 新建文件夹

复制文件夹【inc】、【src】到文件夹【STM32F10x_FWLib】下，您可以直接到我们提供的工程模板下复制。【inc】、【src】是官方固件库包，其中【inc】内是.h 头文件，【src】是.c 源文件。

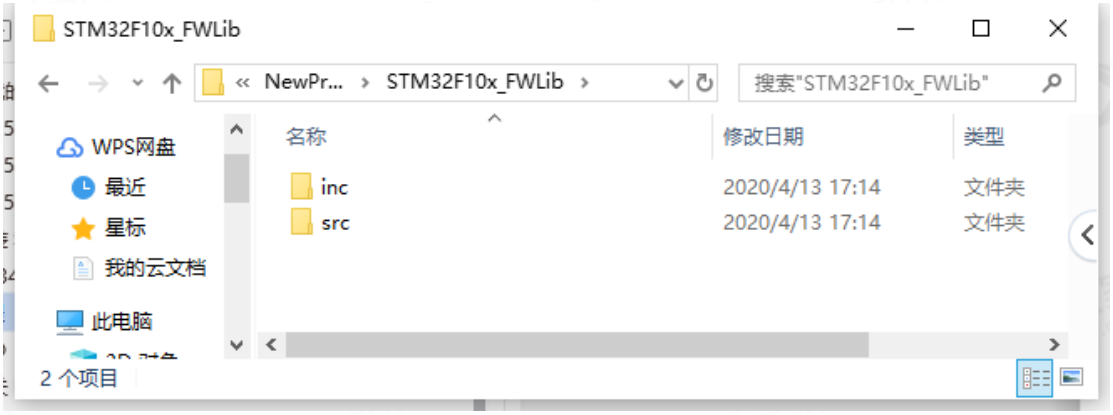

图 1-1-7 复制文件

复制文件【core_cm3.c】、【core_cm3.h】、【startup_stm32f10x_hd.s】到文件夹【CORE】下，您可以直接到我们提供的工程模板下复制。其中【core_cm3.c】、【core_cm3.h】为核心文件，【startup_stm32f10x_hd.s】为启动文件，启动文件除了【startup_stm32f10x_hd.s】外还有【startup_stm32f10x_ld.s】和【startup_stm32f10x_md.s】，三者适用与不同容量的 STM32 芯片。

【startup_stm32f10x_ld.s】适用与小容量（FLASH 的大小≤32K）的芯片，【startup_stm32f10x_md.s】适用与中容量（64K≤FLASH 的大小≤128K）的芯片，【startup_stm32f10x_ld.s】适用与大容量（256K≤FLASH 的大小）的芯片。

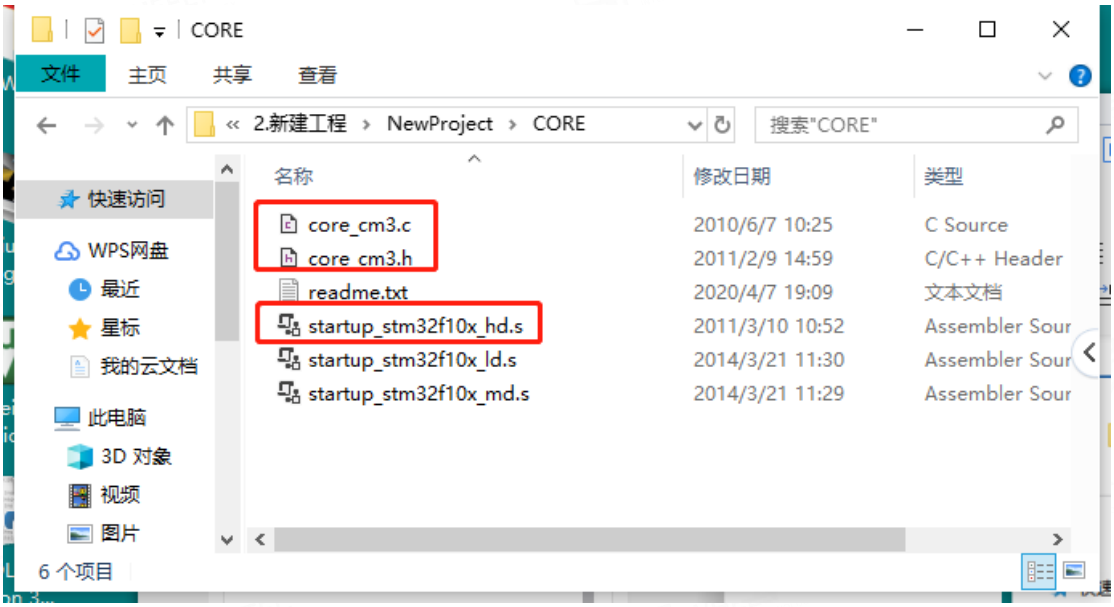

图 1-1-8 复制文件

复制文件【main.c】、【stm32f10x.h】、【stm32f10x_conf.h】、【stm32f10x_it.c】、【stm32f10x_it.h】、【system_stm32f10x.c】、【system_stm32f10x.h】到文件夹【USER】下。

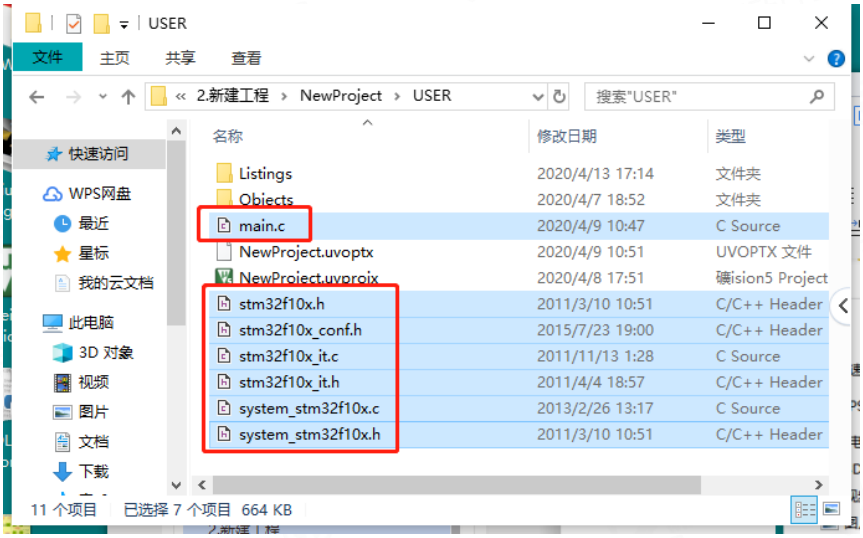

图 1-1-9 复制文件

1.2 添加官方库和 HEX 文件设置

① 新建工程文件分组

我们已经将需要的固件库相关文件复制到了我们的工程目录下面，但是我们还要将这些文件加入工程中去才能使用。右键点击 【Target1】，选择【Manage Project Items】。

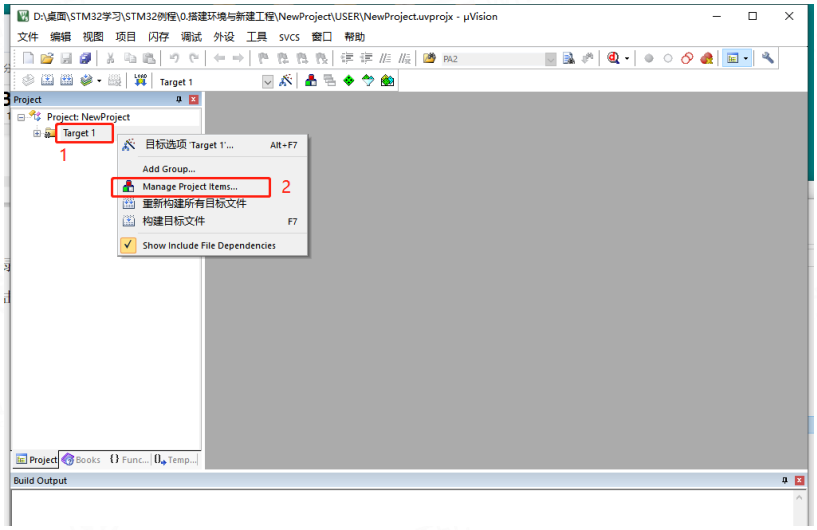

图 1-2-1 选择项目管理器

【Project Targets】一栏，我们将【Target1】重命名为【NewProject】,然后在【Groups】 一栏删掉【Source Group1】，建立三个 Groups：【USER】、【CORE】、【STM32F10x_FWLib】。点击 OK。可以看到【NewProject】下多了三个子项目【USER】、【CORE】、【STM32F10x_FWLib】。

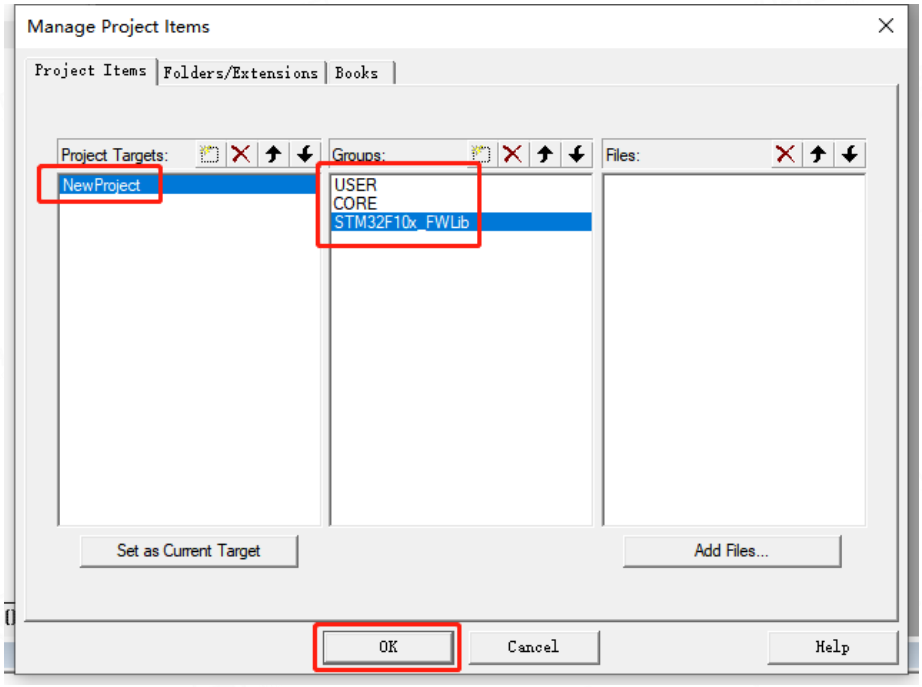

图 1-2-2 新建分组【USER】、【CORE】、【STM32F10x_FWLib】

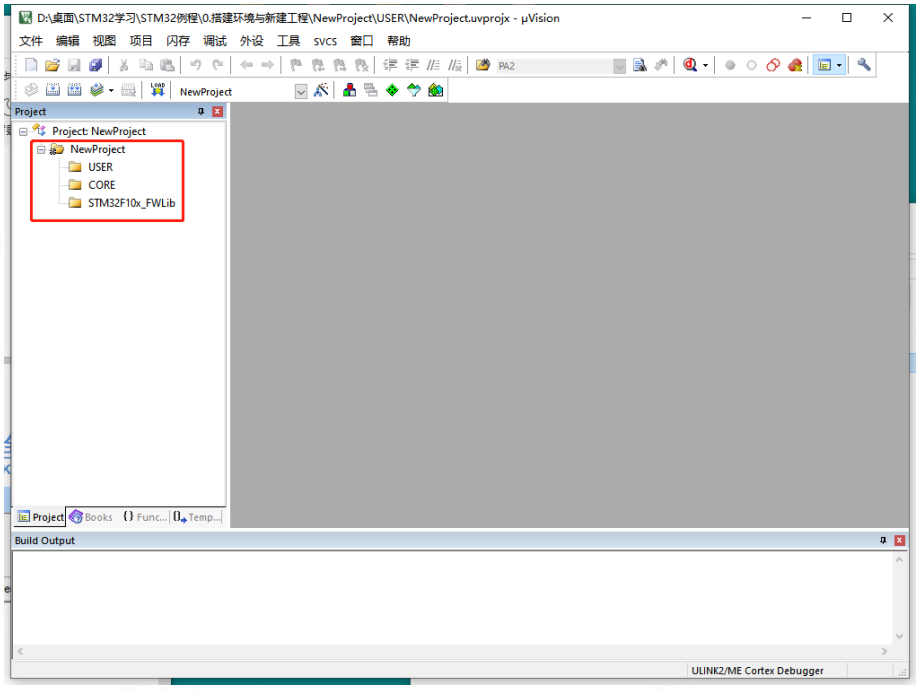

图 1-2-3 分组效果

② 向【USER】分组添加库文件

右键点击 【NewProject】，选择【Manage Project Items】。

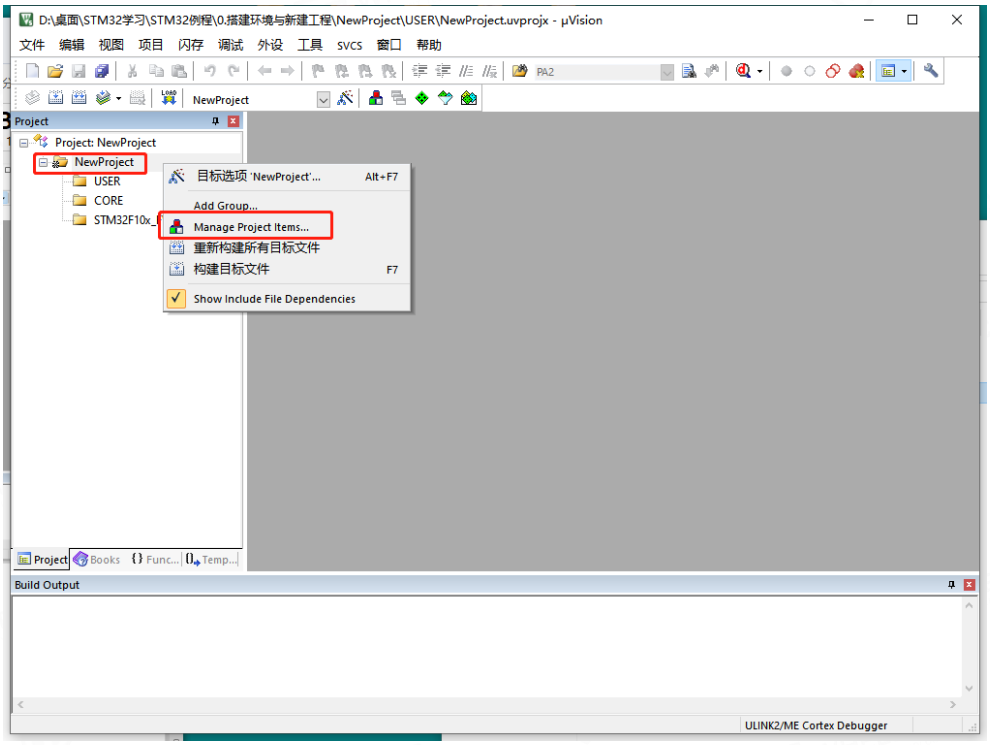

图 1-2-4 选择项目管理器

选中 Groups【USER】后，点击【Add Files】。

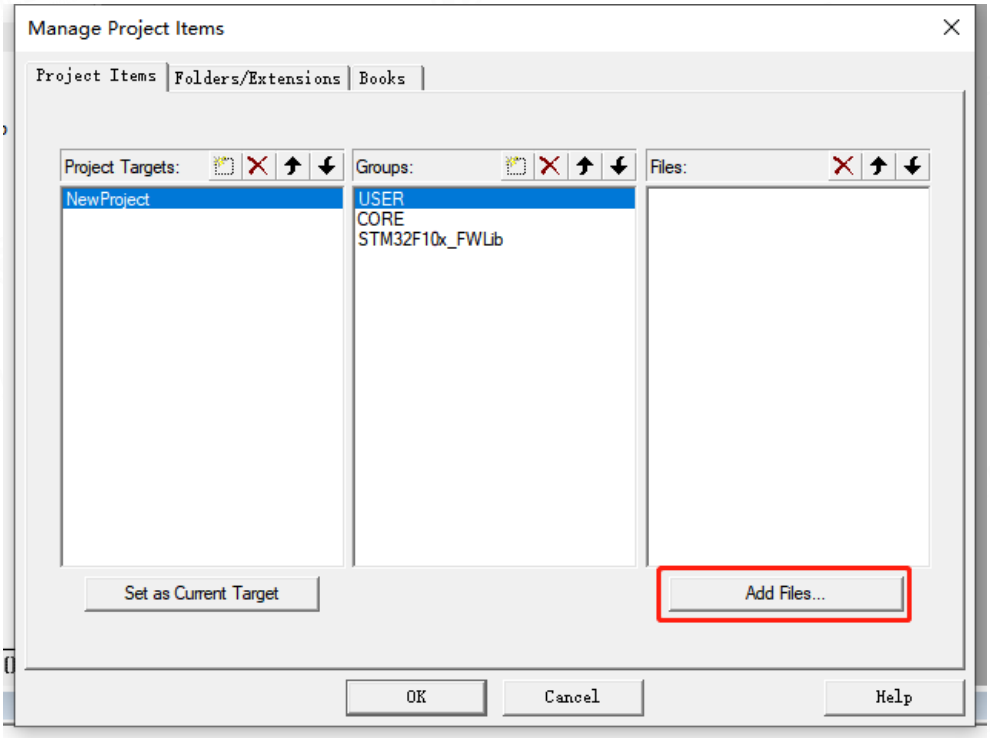

图 1-2-5 向【USER】分组添加文件

选中文件夹【USER】下的【main.c】、【stm32f10x_it.c】、【system_stm32f10x.c】，点击【Add】，然后点击【Close】。可以看见 Groups【USER】下成功添加了 3 个文件。

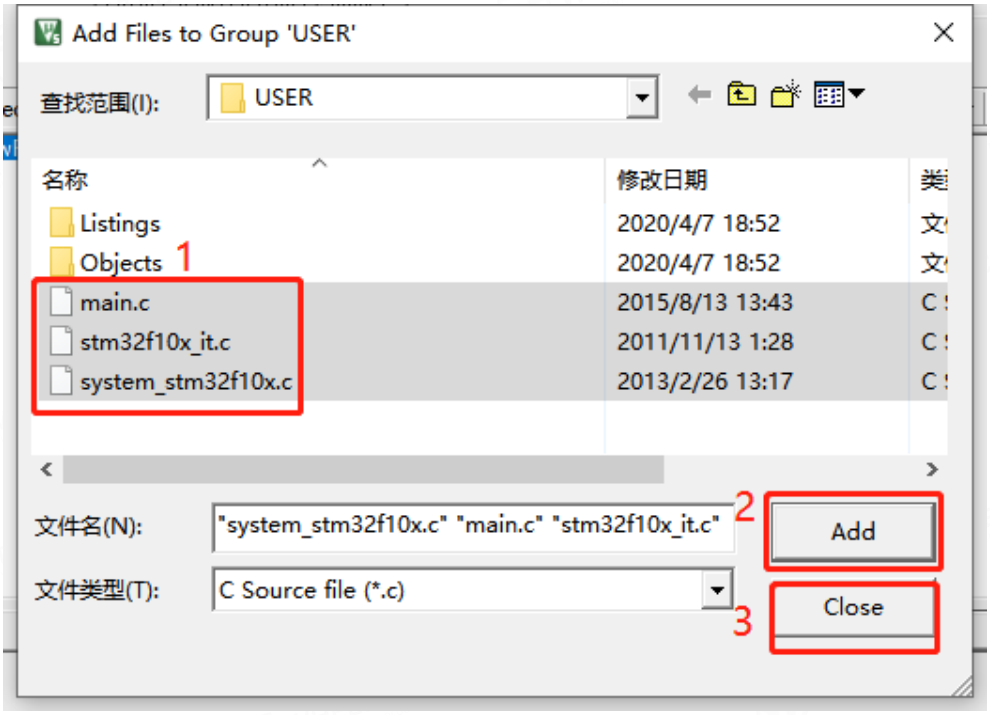

图 1-2-6 Add 源文件(.c 文件)

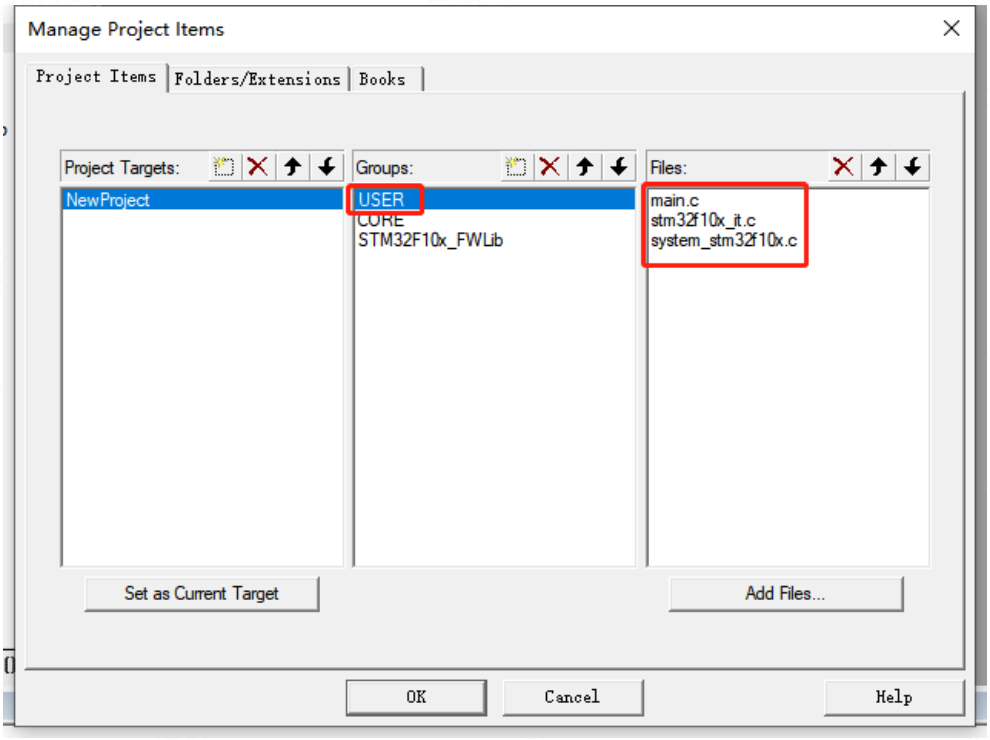

图 1-2-7 添加文件后的 USER 分组

③ 向【CORE】分组添加库文件

选中 Groups【CORE】后，点击【Add Files】。

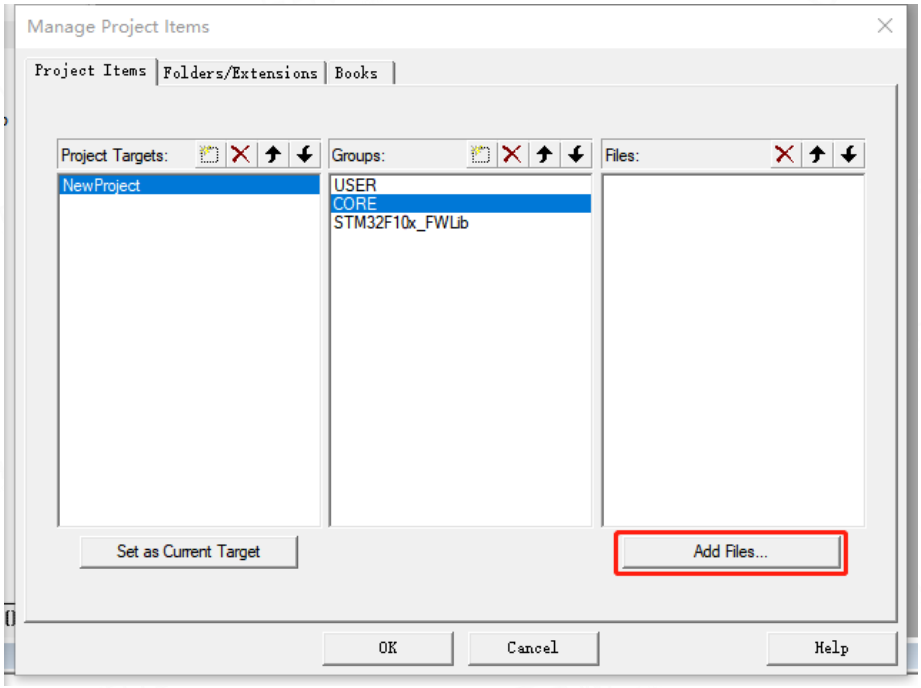

图 1-2-8 向【CORE】分组添加文件

点击【1】处，返回上一级文件夹，双击文件夹【CORE】，点击【2 处】，选择【All Files (*.*)】。

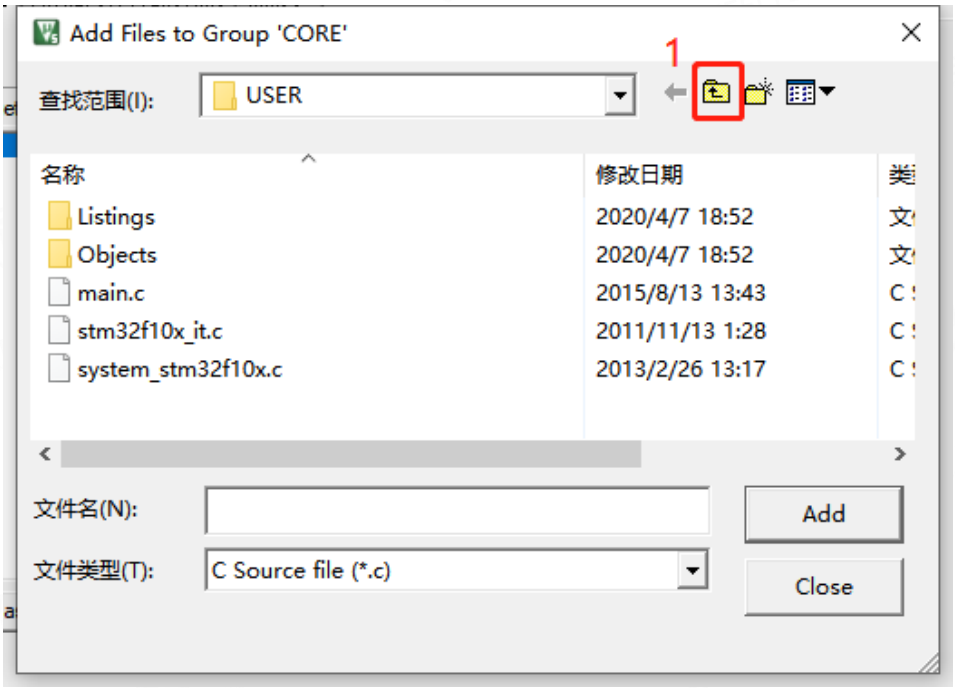

图 1-2-9 返回上一级

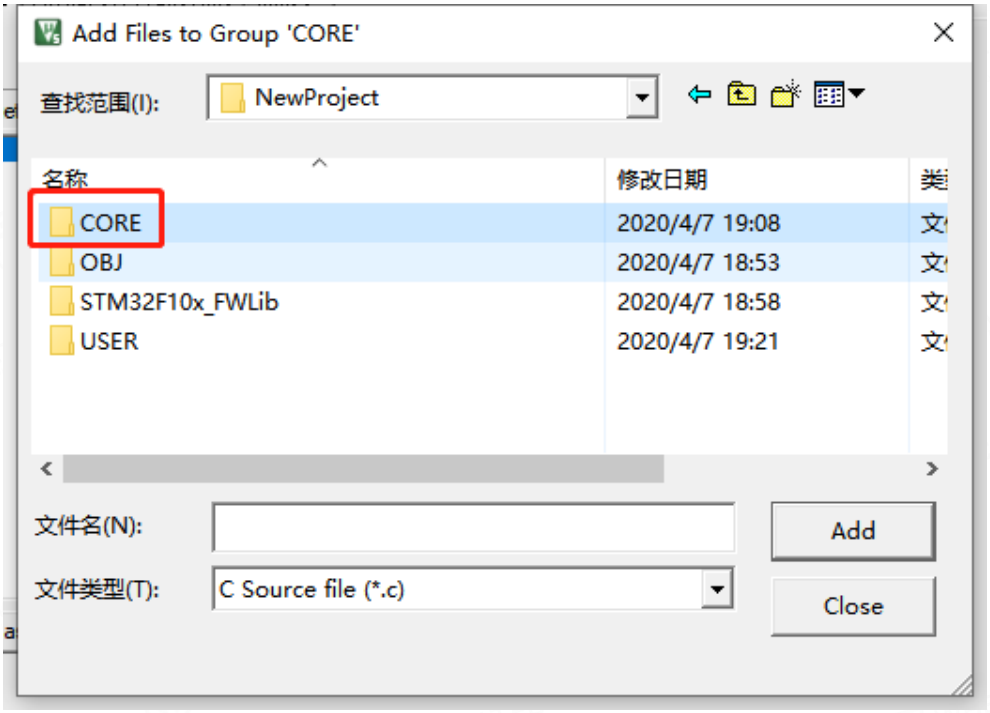

图 1-2-10 点击进入【CORE】文件夹

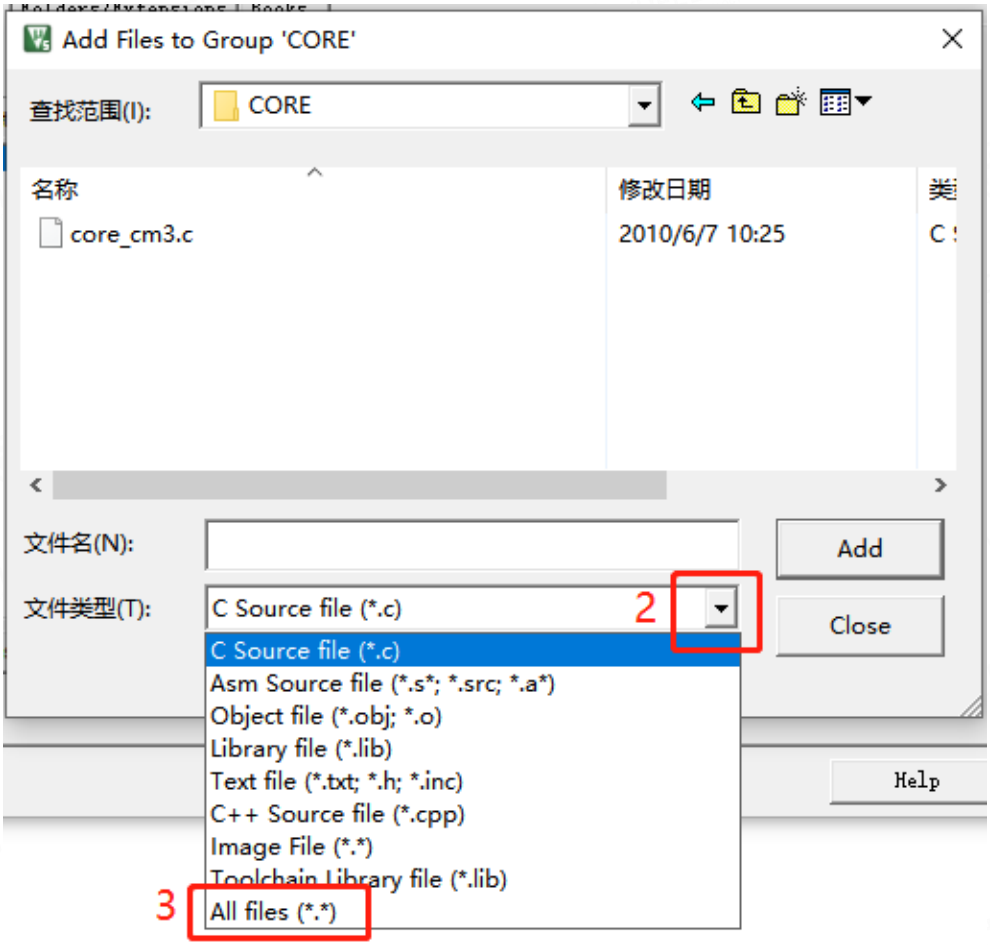

图 1-2-11 选择可识别文件类型

选中文件【core_cm3.c】、【core_cm3.h】和【startup_stm32f10x_hd.s】（启动文件根据芯片 FLASH 大小选择 hd、ld 还是 md），点击【Add】，然后点击【Close】。

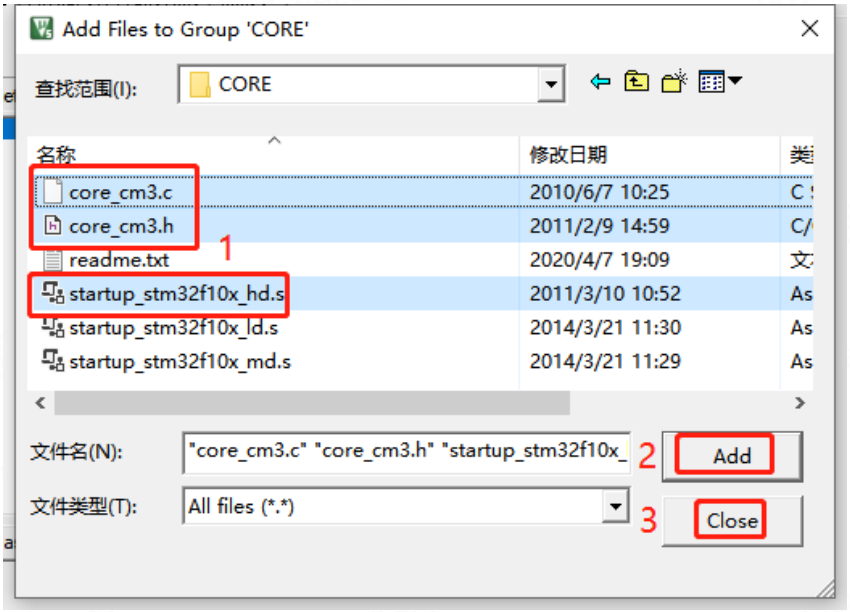

图 1-2-12 Add 源文件(.c 文件)

④ 向【STM32F10x_FWLib】分组添加库文件

选中 Groups【STM32F10x_FWLib】后，点击【Add Files】。

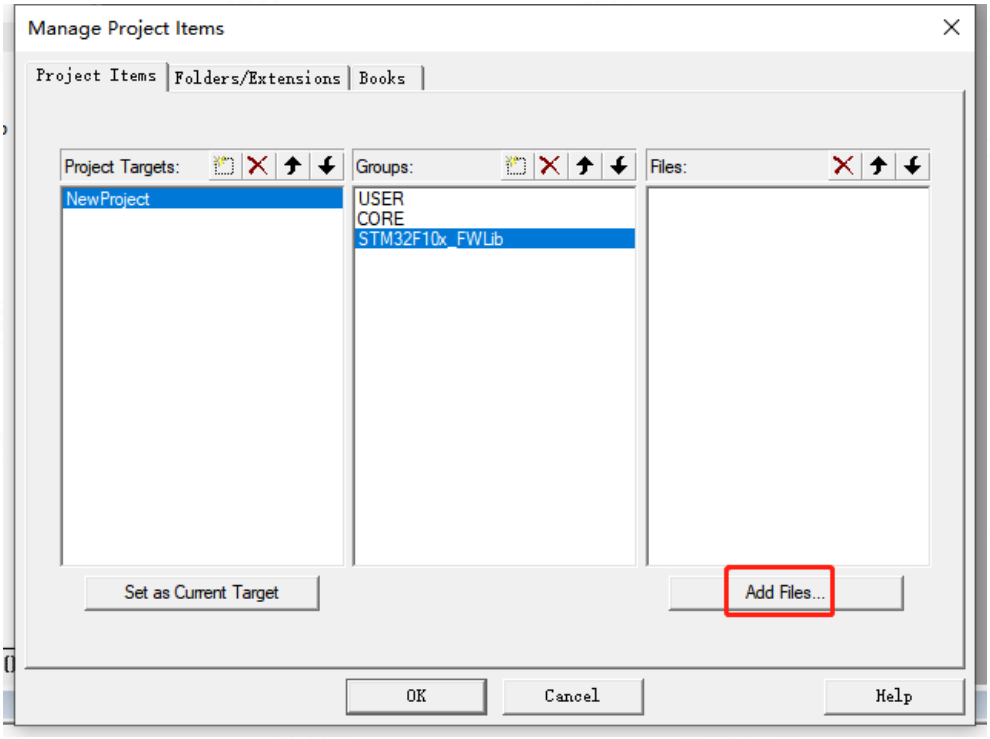

图 1-2-13 向【STM32F10x_FWLib】分组添加文件

点击【1】处，返回上一级文件夹，双击文件夹【STM32F10x_FWLib】，双击文件夹【src】。

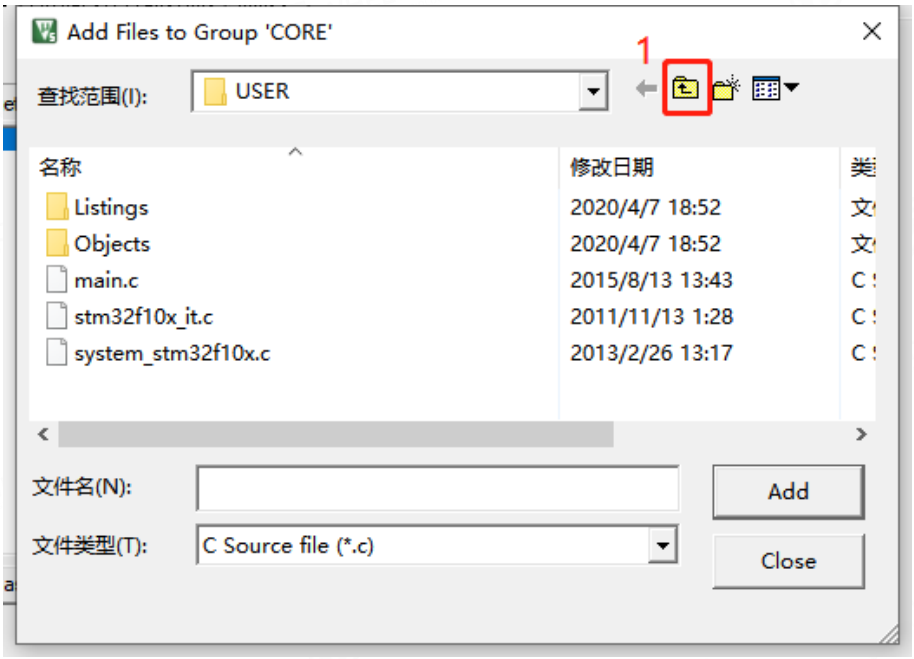

图 1-2-14 返回上一级文件夹

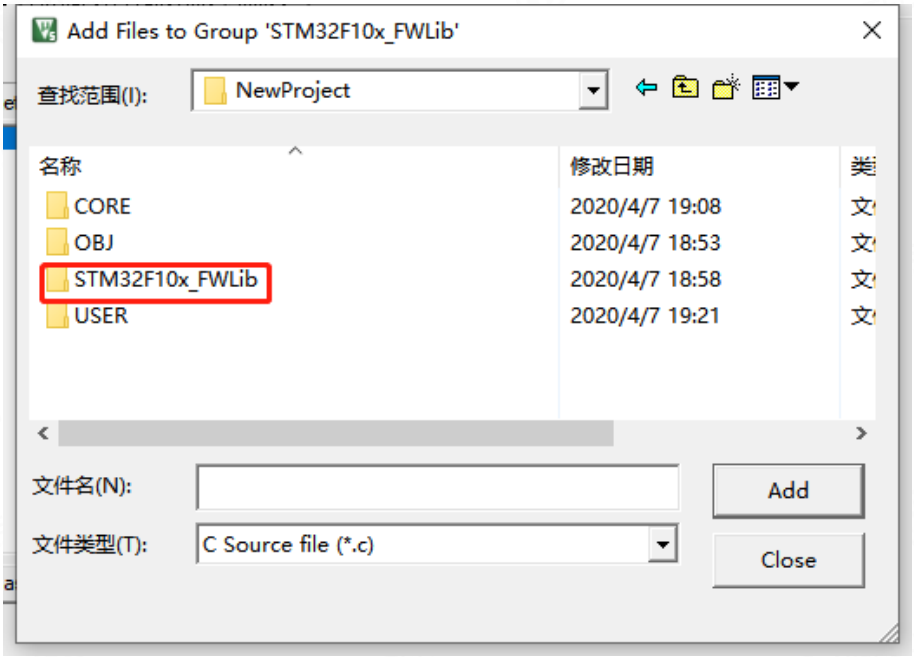

图 1-2-15 点击进入【STM32F10x_FWLib】文件夹

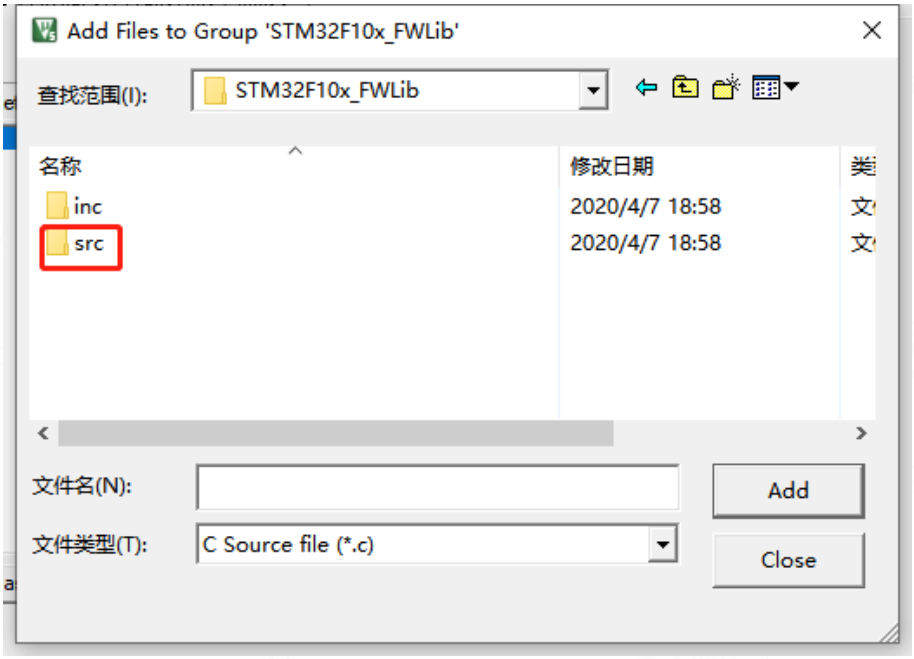

图 1-2-16 点击进入【src】文件夹

选中该文件夹下的所有文件，点击【Add】，然后点击【Close】。该文件夹下是外设库文件，如果我们只用到了其中的某个外设，我们可以不添加没有用到的外设的库文件。例如如果只用 GPIO，则可以只添加 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，这样做不好的方面是工程太大，编译起来速度慢。

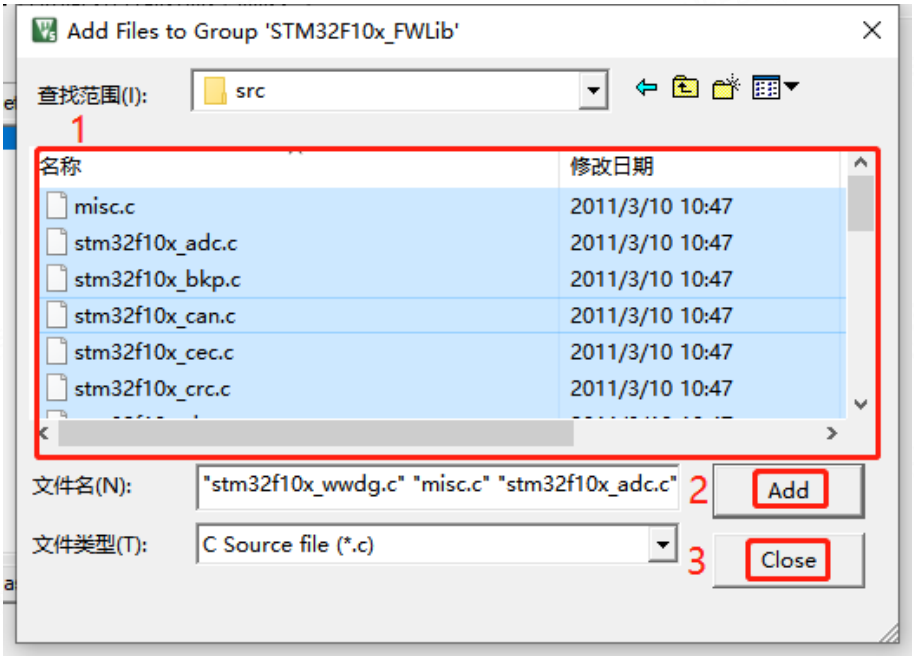

图 1-2-17 Add 源文件(.c 文件)

点击【OK】。

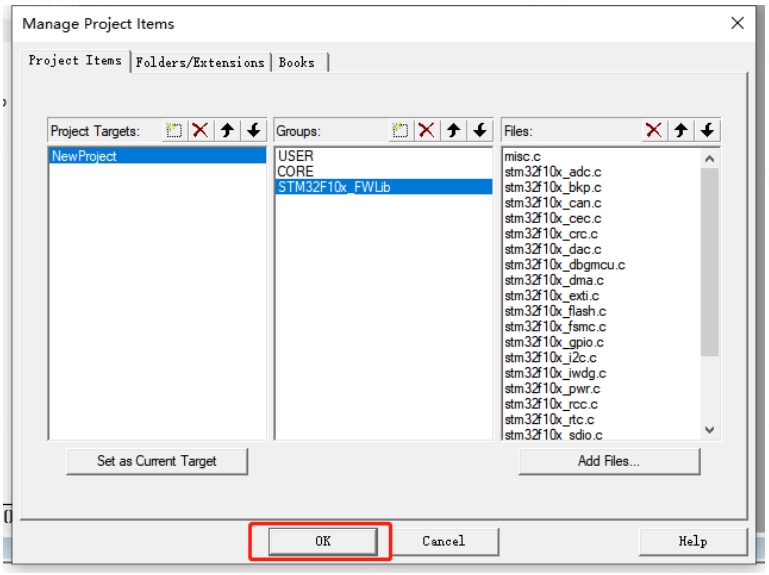

图 1-2-18 点击 OK 确认添加源文件

⑤ 编辑 HEX 文件生成路径

接下来选择 HEX 文件的存储路径，HEX 文件是用于上传至 STM32 芯片的文件（HEX 文件的默认存储路径为文件夹【USER】下的文件夹【Objects】）。

点击 1 处的【魔术棒】，点击【Output】，勾选【Create HEX File】点击【Select Folder for Objects...】。

选择到文件夹【NewProject】下的文件夹【OBJ】作为 HEX 文件保存路径，点击 OK。

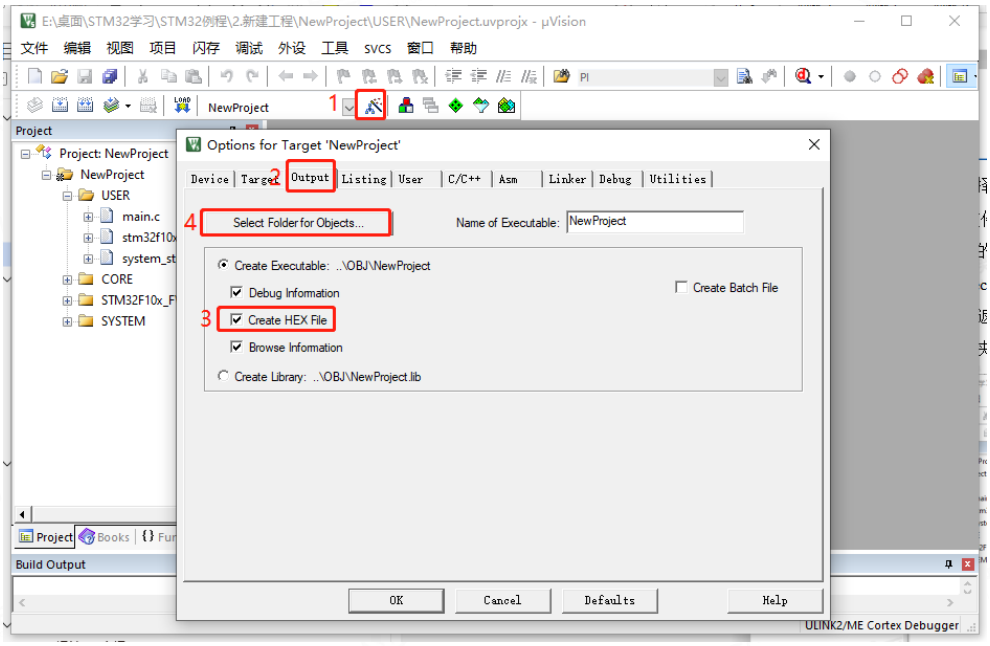

图 1-2-19 确认生成 HEX 文件并选择 HEX 文件保存路径

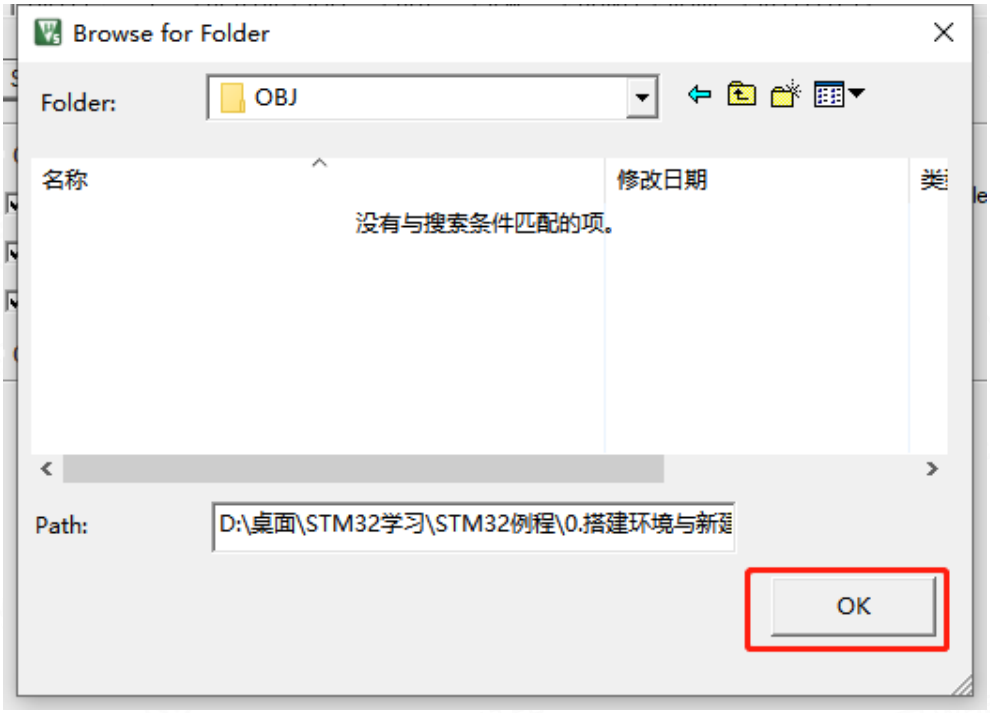

图 1-2-20 选择到文件夹【NewProject】下的文件夹【OBJ】，点击 OK

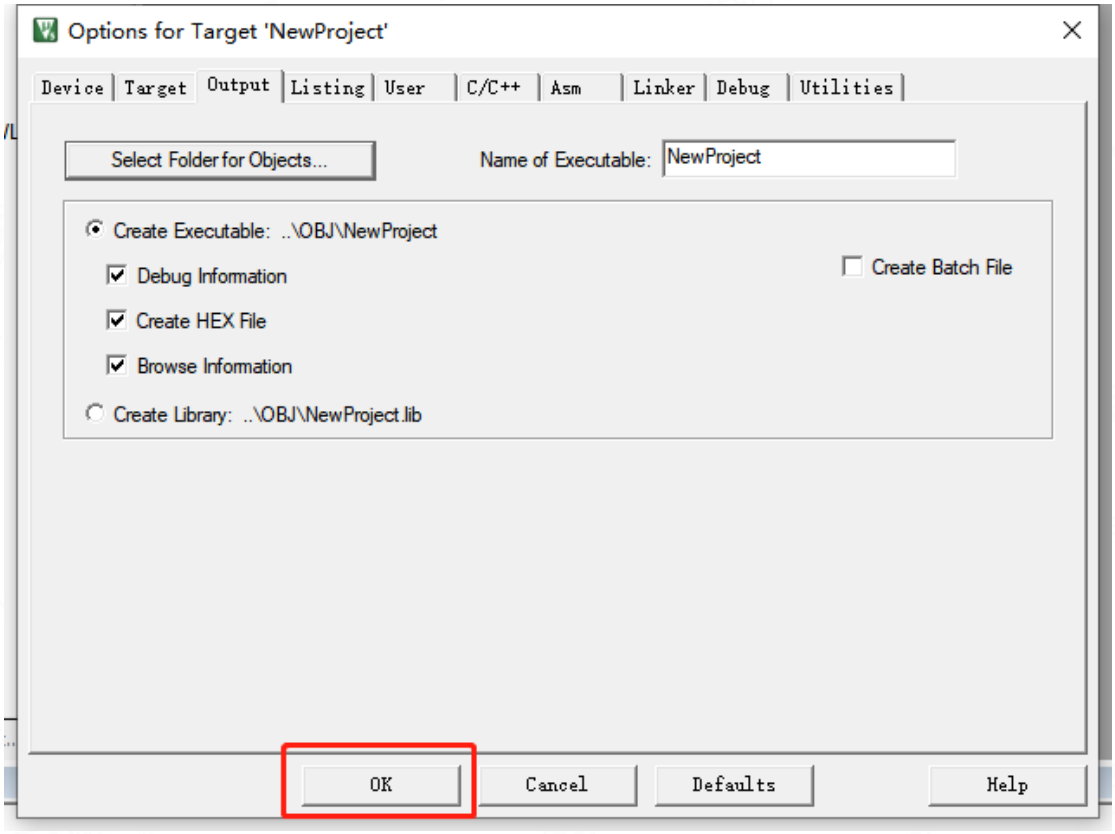

图 1-2-21 点击 OK

⑥ 添加头文件路径

接下来还需要添加头文件所在路径，否则点击 1 处【构建】时会报错，如 2 处。要注意，所有使用到的头文件都必须添加其所在路径进工程。

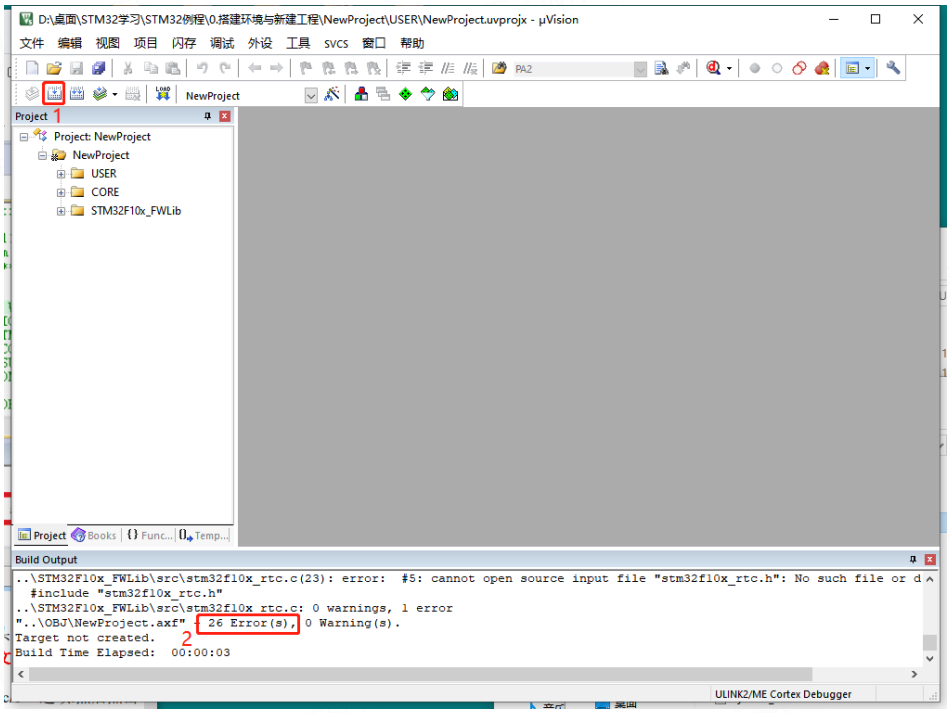

图 1-2-22 存在报错

点击【魔术棒】，点击【C/C++】，点击 3 处【...】。

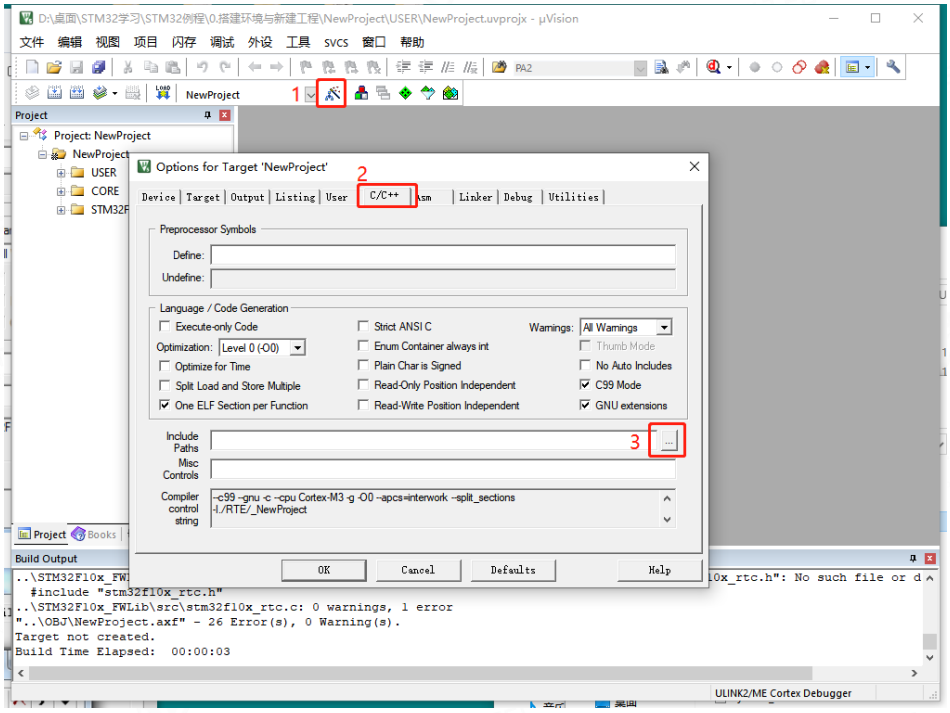

图 1-2-23 选择添加系统头文件(.h 文件)所在路径

点击 1 处，点击 2 处。

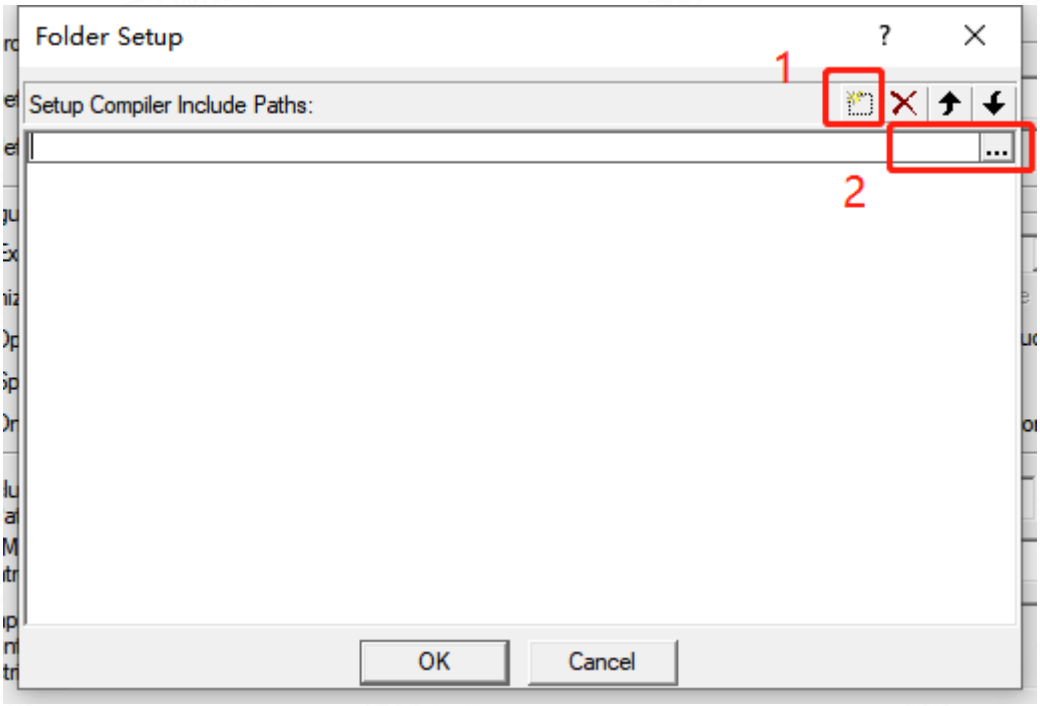

图 1-2-24 系统头文件(.h 文件)所在路径设置

分别在在路径【NewProject\USER】、【NewProject\CORE】、【NewProject\STM32F10x_FWLib\inc】下点击【选择文件夹】，以设置系统头文件(.h 文件)所在路径。

图 1-2-25 选择【USER】文件夹

图 1-2-26 选择【CORE】文件夹

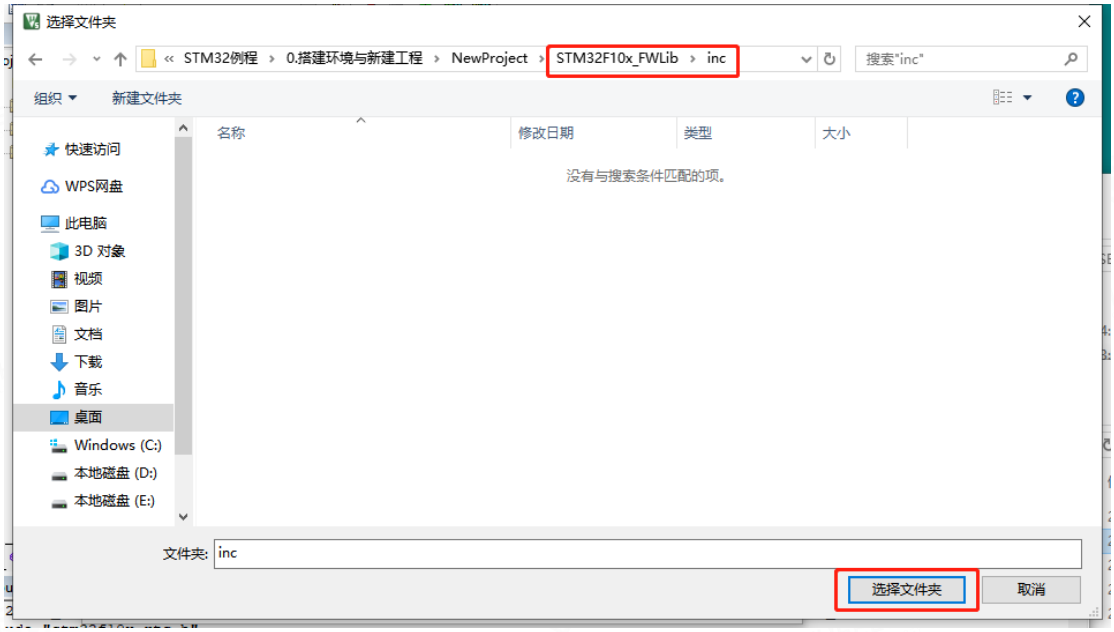

图 1-2-27 选择【STM32F10x_FWLib】文件夹

添加头文件设置路径成功，点击【OK】。

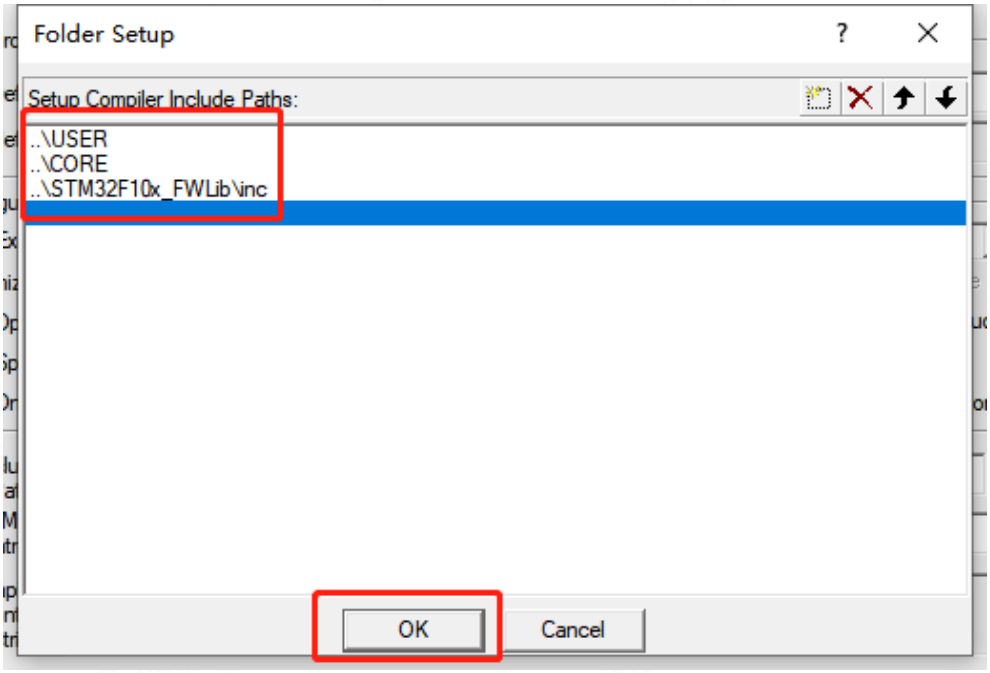

图 1-2-28 确认系统头文件(.h 文件)所在路径

⑦ 添加宏定义与编译测试

在【Define】下输入“STM32F10X_HD,USE_STDPERIPH_DRIVER”以添加宏定义，点击【OK】。注意其中的【STM32F10X_HD】对应大容量：256K≤FLASH 的 STM 芯片。中容量则应该改为【STM32F10X_MD】，小容量则应该改为【STM32F10X_LD】。

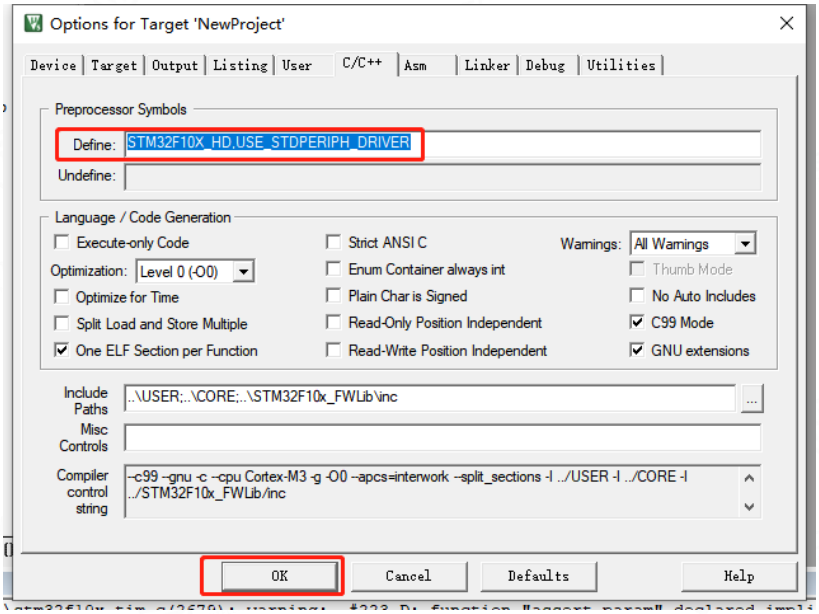

图 1-2-29 添加宏定义

此时点击 1 处【全部重新构建】，显示运行成功，无错误，无警告。注意 main.c 里的最后一行要为空行，以后建立的所有程序，最后一行都要为空行，否则会有一个警告，该警告属于 Keil 的 BUG。

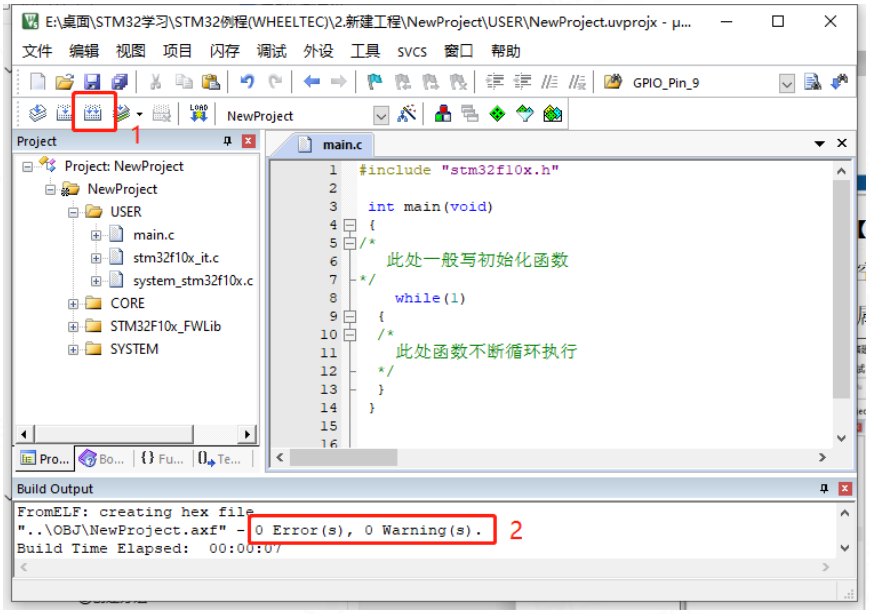

图 1-2-30 无报错

1.3 添加 delay、sys 和 usart 库

① 添加【.C】文件

经过以上步骤，工程基本完成了，接下来我们在添加几个库函数，方便以后编写程序。在文件夹【NewProject】下新建文件夹【SYSTEM】，复制文件夹【delay】、【sys】、【usart】到文件夹【SYSTEM】下，您可以直接到我们提供的工程模板下复制。

【delay】里面是封装好的延迟函数，【sys】里面是封装好的 IO 口控制函数，【usart】里面是封装好的串口使用函数。

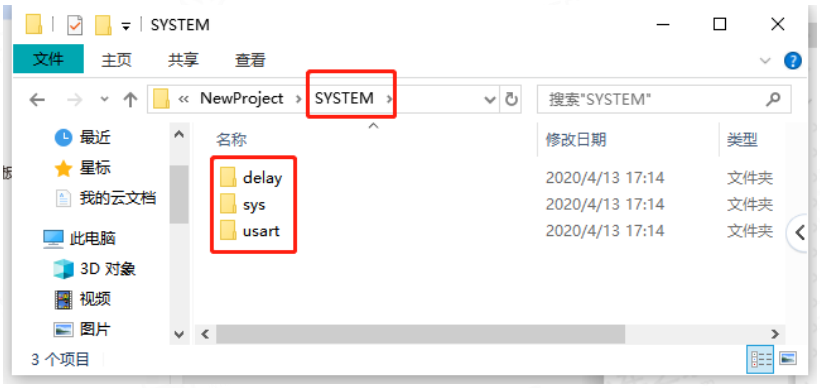

图 1-3-1 复制粘贴已准备好的库函数

接下来将相关.c 文件添加进工程。右键点击 【NewProject】，选择【Manage Project Items】。

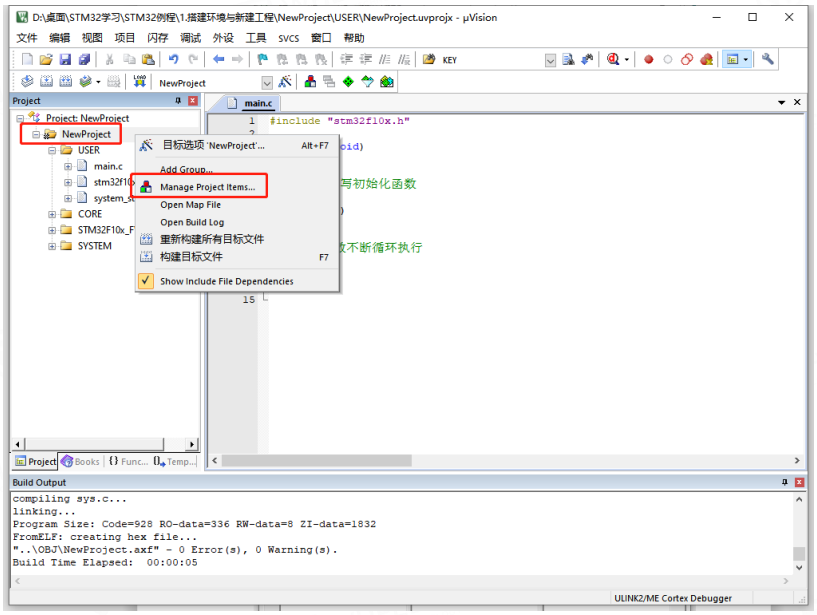

图 1-3-2 选择项目管理器

在【Groups】下新建【SYSTEM】，选中【SYSTEM】后点击【Add Files】。

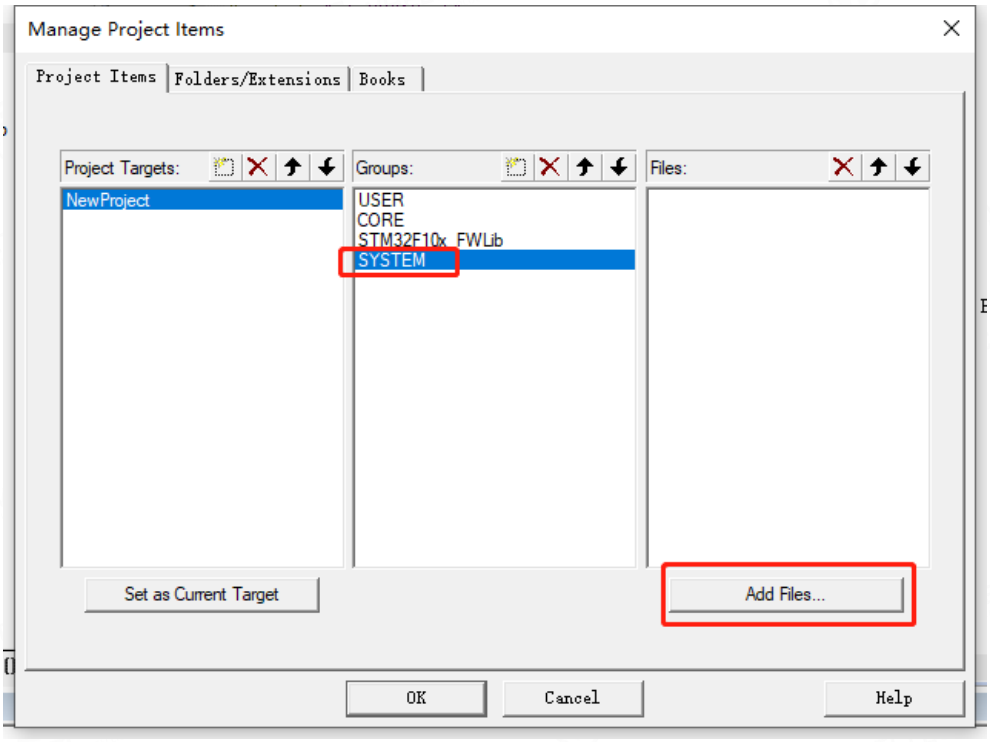

图 1-3-3 向【SYSTEM】分组添加文件

在路径下【NewProject\SYSTEM\delay】，点击【Add】。

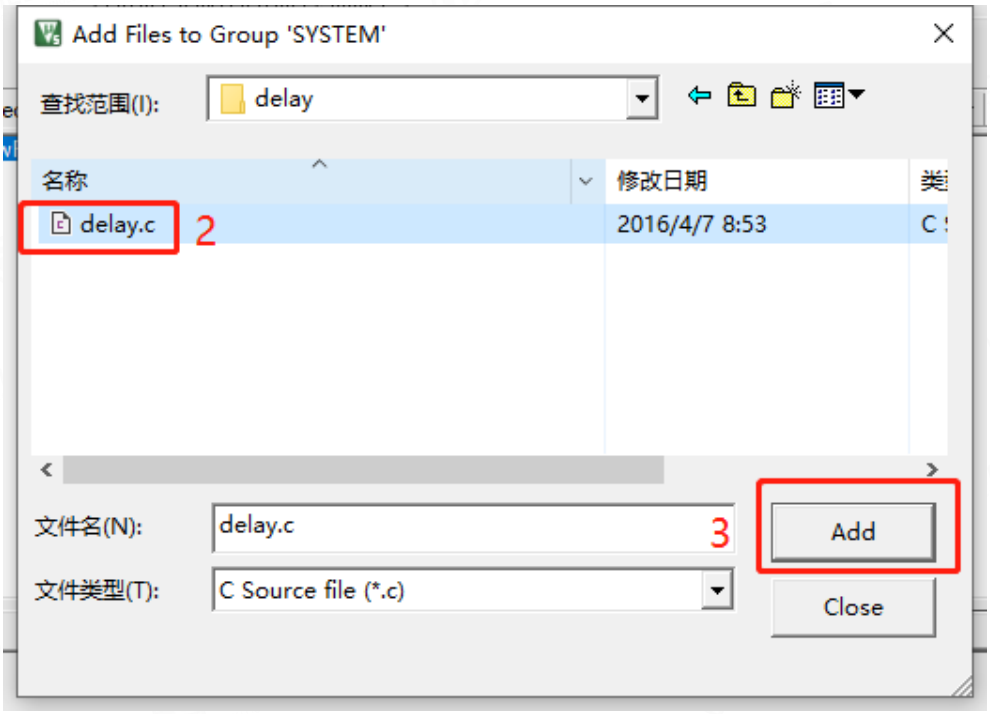

图 1-3-4 Add 源文件(.c 文件)

重复类似前面的步骤，将文件夹【sys】下的文件【sys.c】和文件夹【usart】下的文件【usart.c】都添加进 Groups【SYSTEM】中。点击【OK】。

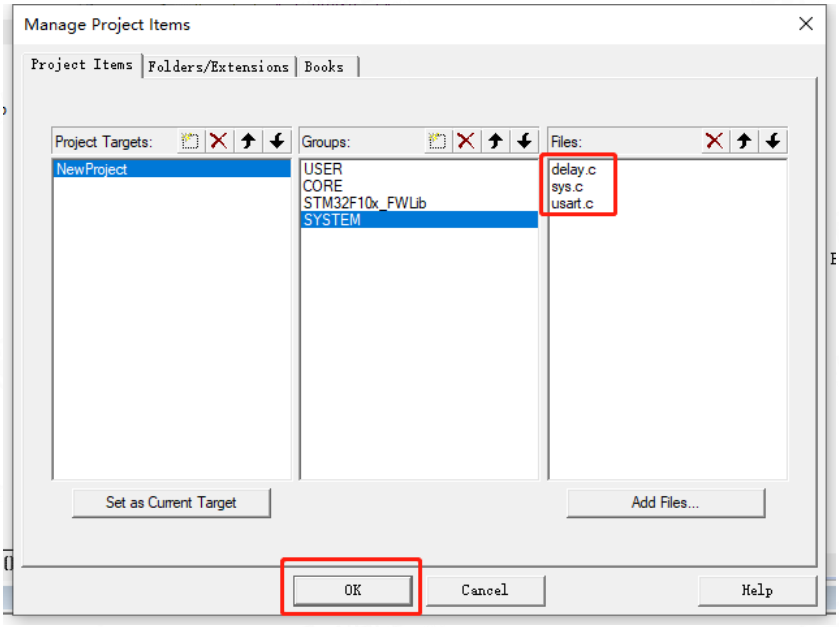

图 1-3-5 添加源文件【sys.c】、【usart.c】

② 添加对应头文件路径

点击【魔法棒】，点击【C/C++】，点击 3 处【...】。

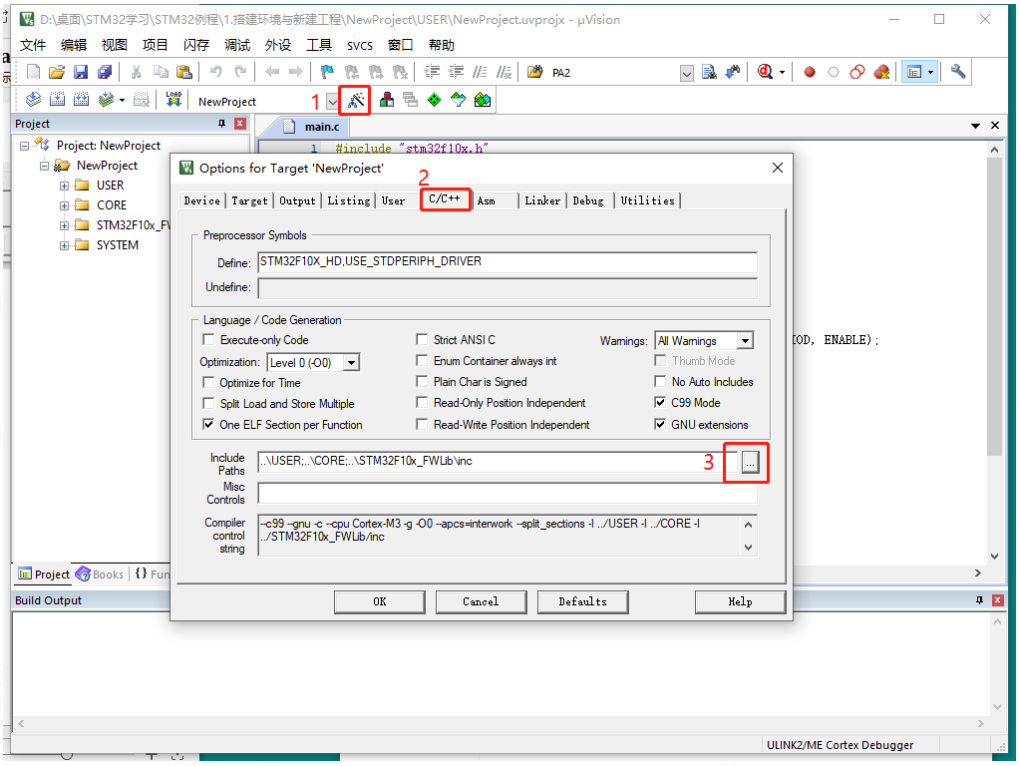

图 1-3-6 选择添加系统头文件(.h 文件)所在路径

点击 1 处后点击 2 处【...】。

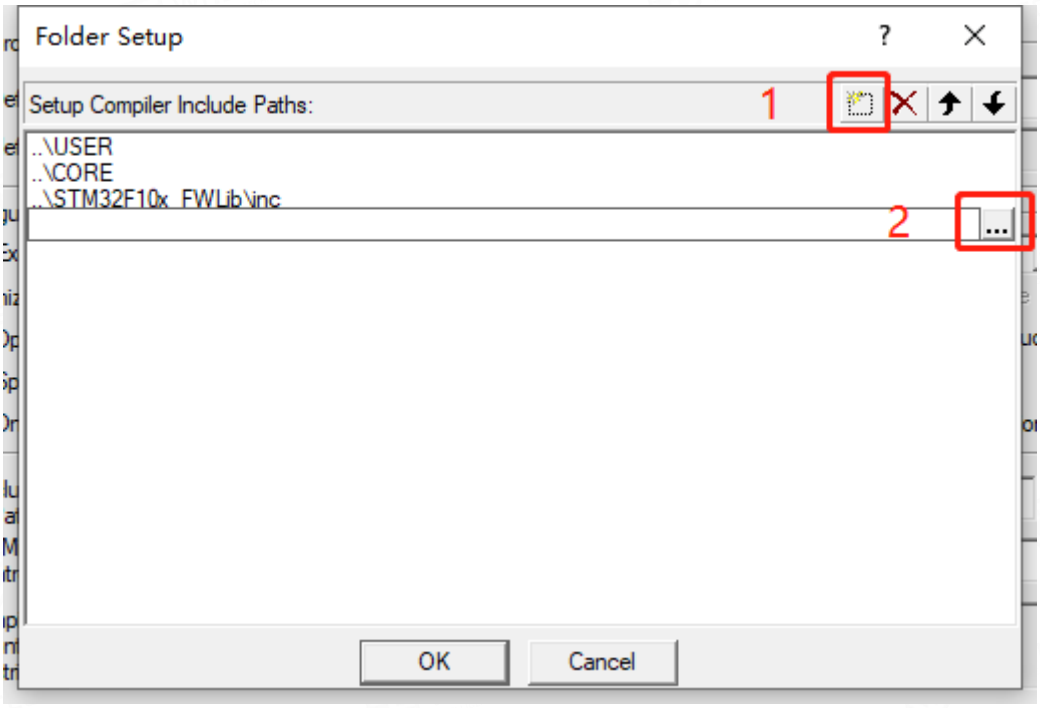

图 1-3-7 系统头文件(.h 文件)所在路径设置

进入到文件夹【SYSTEM】下的文件夹【delay】下，点击【选择文件夹】，则将头文件【delay.h】所在的路径添加进了工程。

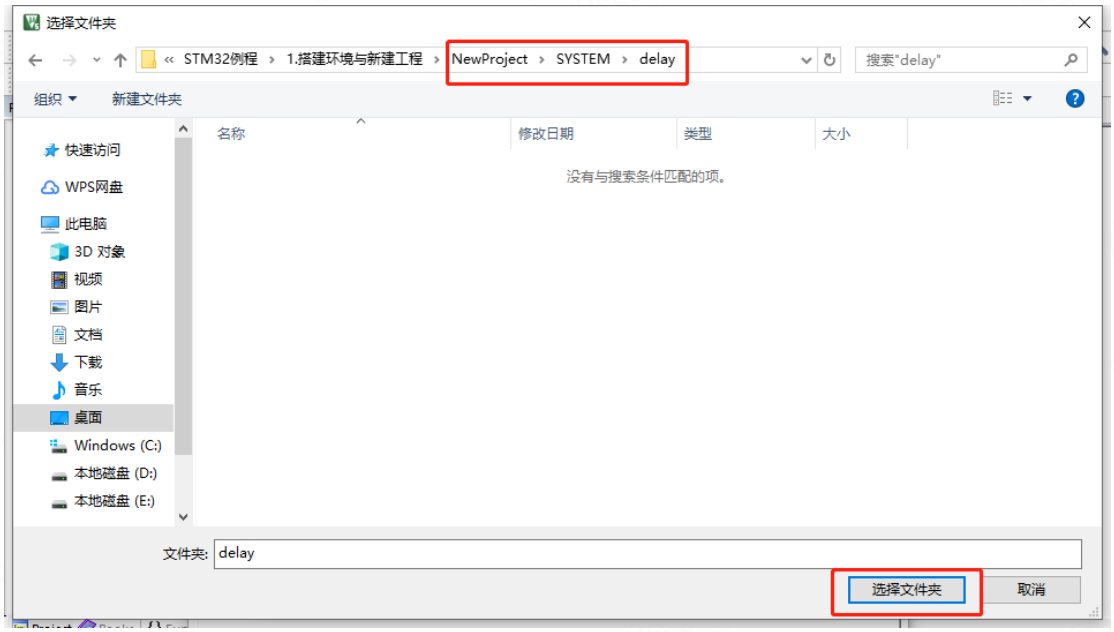

图 1-3-8 选择【delay】文件夹

重复类似前面两步的步骤，将头文件【sys.h】、【usart.h】所在的路径都添加进工程。点击 OK。再次点击【OK】。

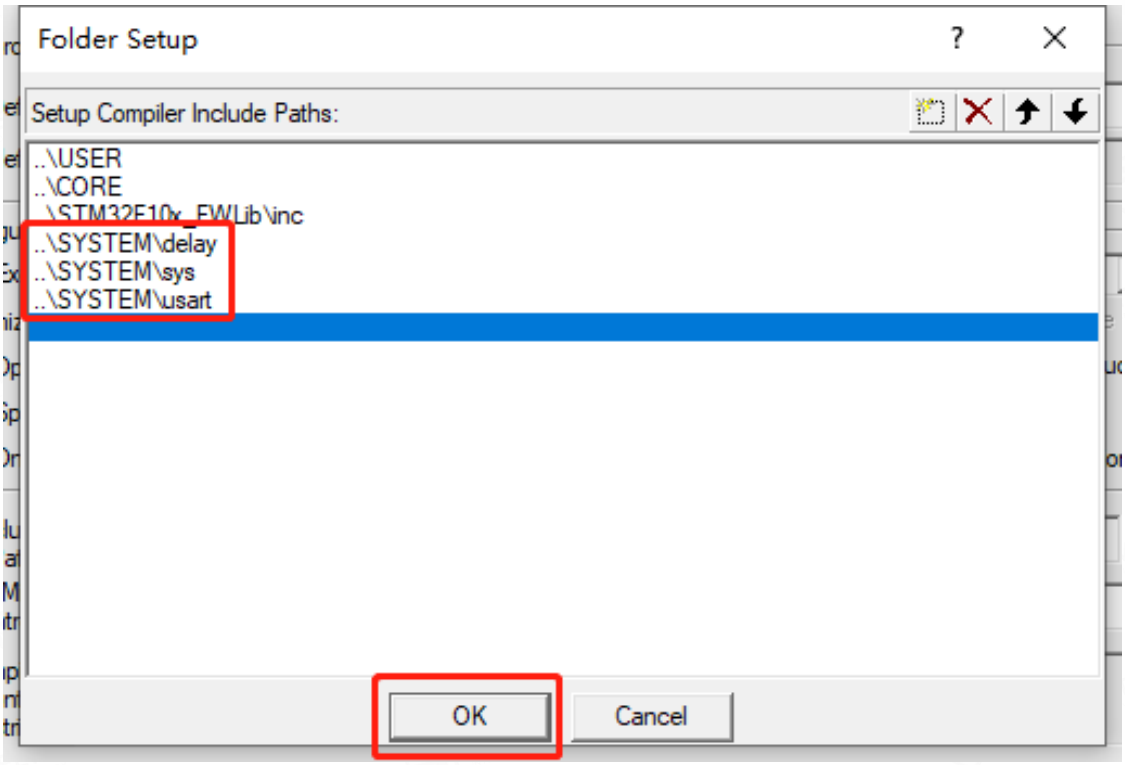

图 1-3-9 继续添加【sys】、【usart】文件夹路径

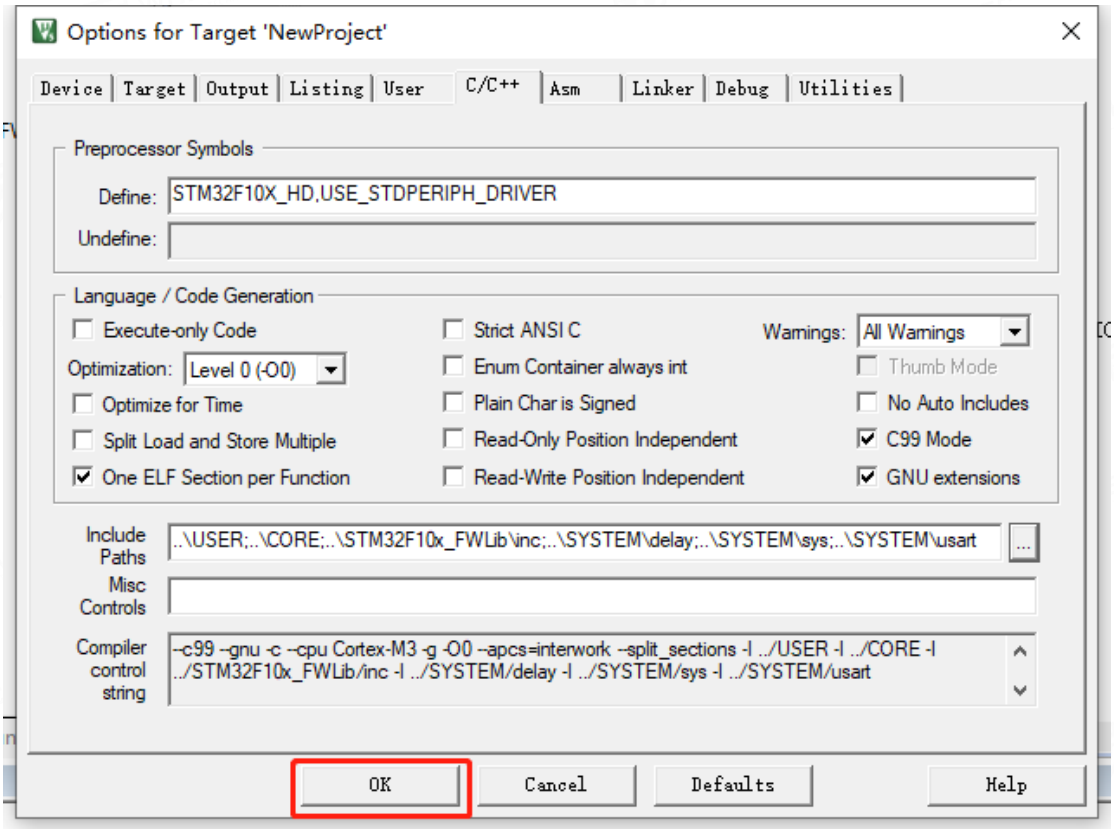

图 1-3-10 点击 OK

1.4 配置简体中文环境

接下来点击【编辑】，点击【配置】。

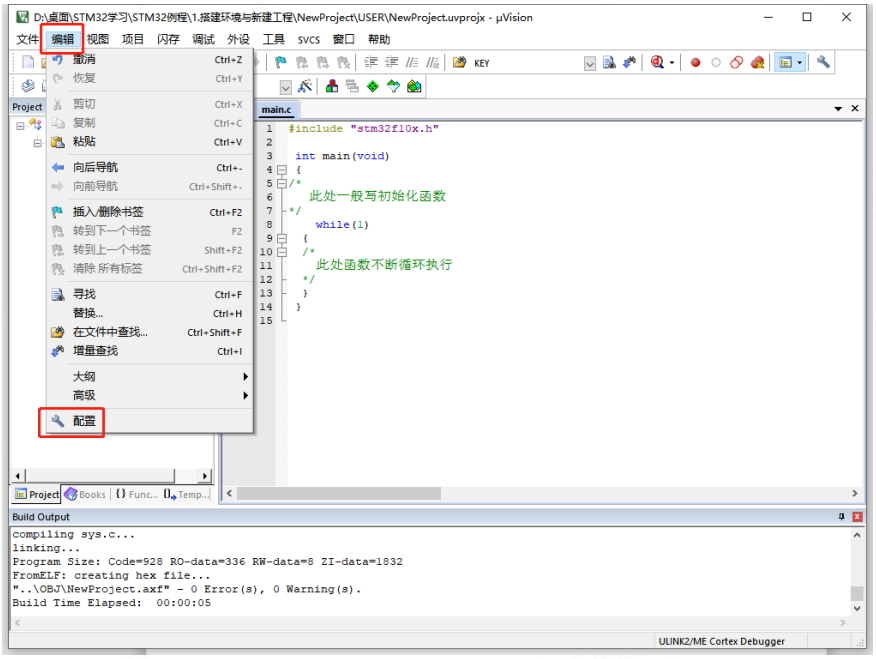

图 1-4-1 选择配置

选择【Encoding】下的【Chinese GB2312(Simplified)】，点击【OK】。此处是为了避免中文注释时出现乱码。

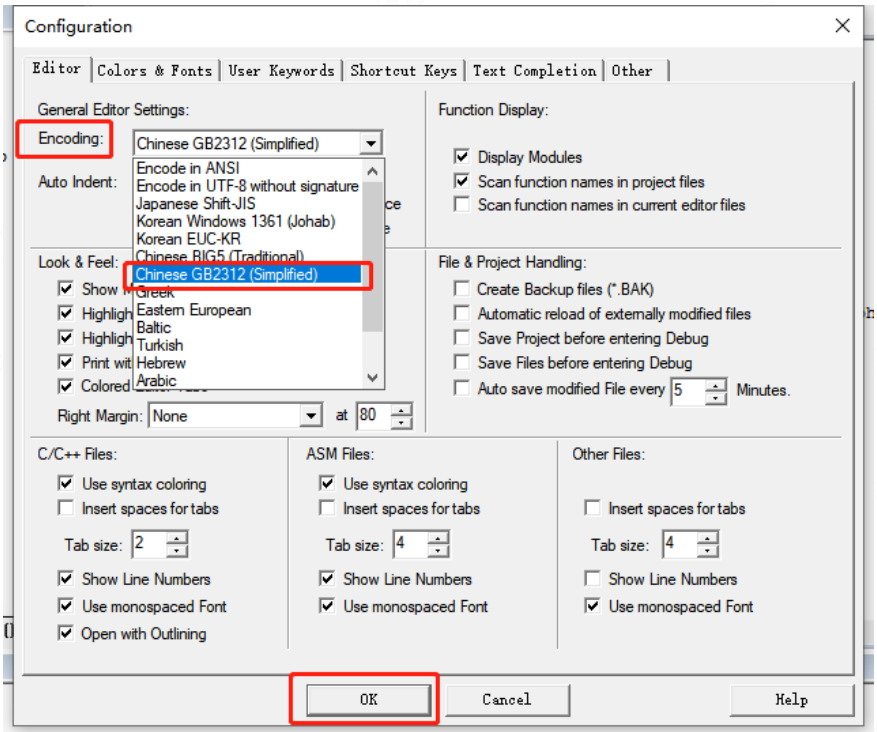

图 1-4-2 选择【Cinese GB2312(Simplified)】并确定

点击 1 处【全部保存】，新建工程完毕。

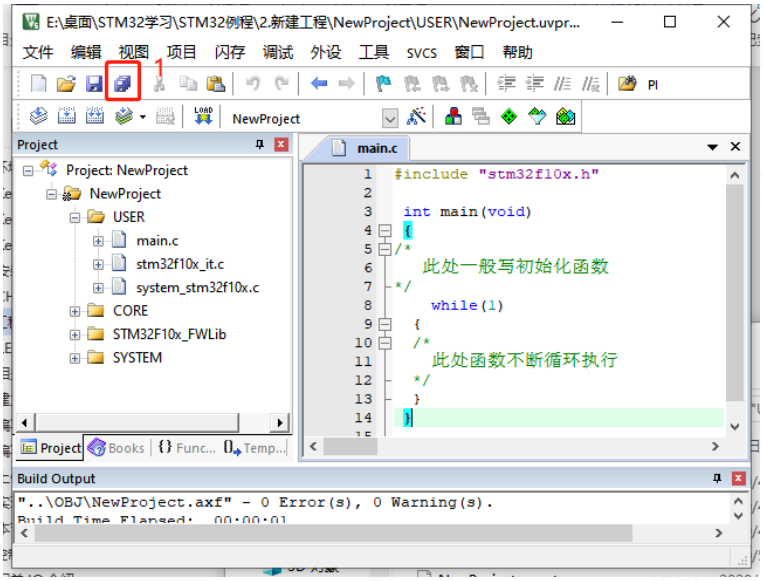

图 1-4-3 点击 1 处保存全部文件

2. 点亮 LED

2.1 相关 IO 介绍

Forest S1 上可控制的 LED 灯有 1 个，为蓝色 LED 灯，受引脚 PA4 控制，低电平时点亮。

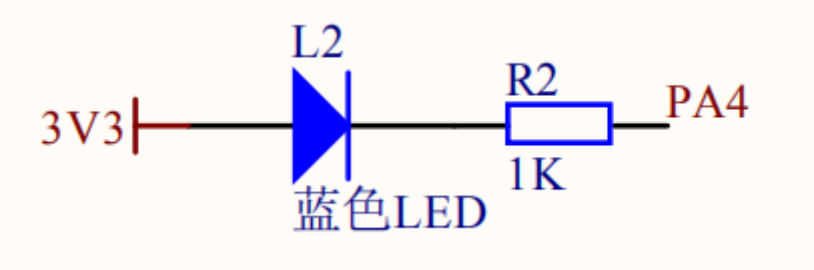

图 2-1-1 蓝色 LED 灯引脚接线图

2.2 创建硬件外设库函数文件并加入工程

为使工程规范、编程方便，一个好的习惯是为单片机要控制的每一个硬件建立库函数。接下来我们以上一节创建好的工程为基础，为 LED 灯建立一个库函数。

① 创建文件夹

首先在文件夹【NewProject】下新建文件夹【HARDWARE】，用于存放硬件库函数。

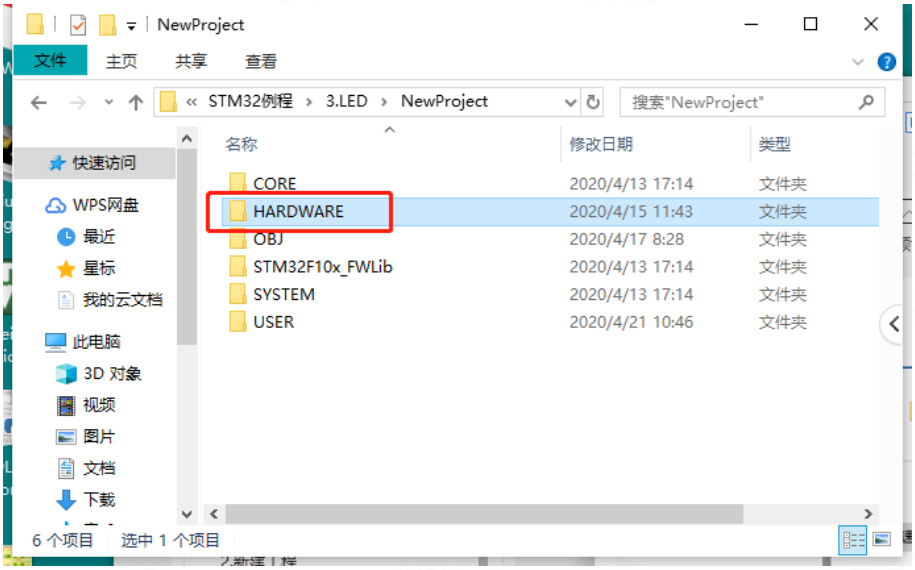

图 2-2-1 新建文件夹

② 创建分组

右键点击 【NewProject】，选择【Manage Project Items】，新建 Groups 【HARDWARE】，点击【OK】。（为节省篇幅，章节【新建工程】演示过的操作不再展示）。

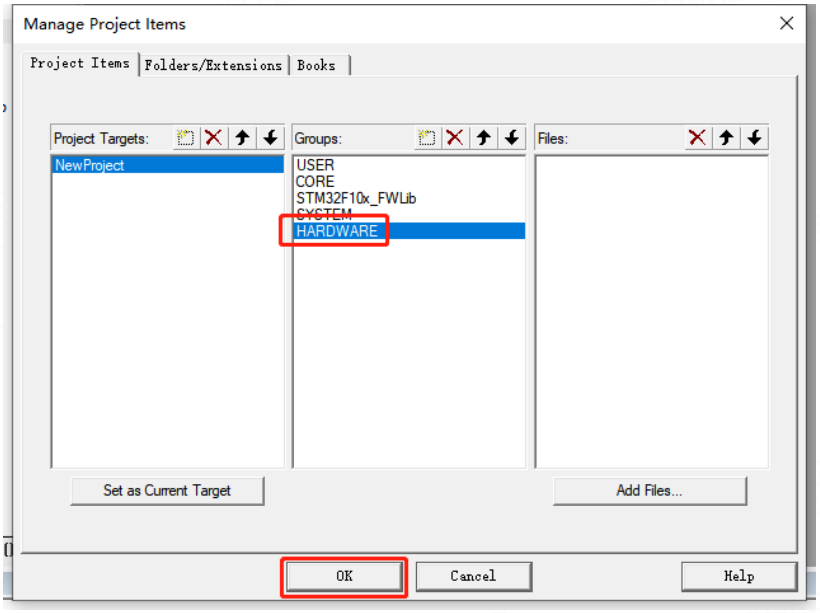

图 2-2-2 新建分组【HARDWARE】

③ 创建库文件

点击 1 处【新建文件】，点击 2 处【保存文件】。

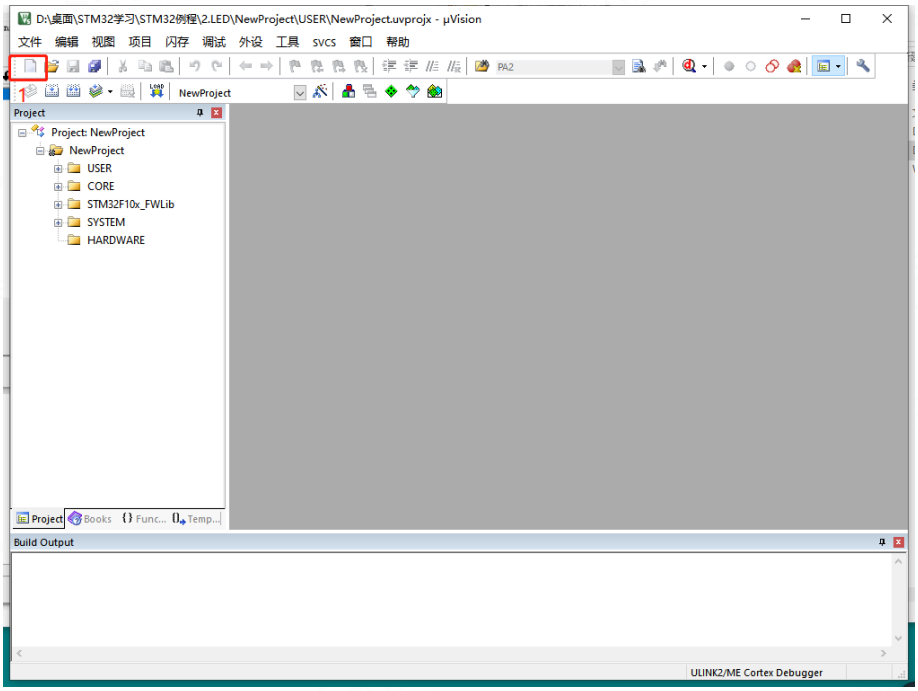

图 2-2-3 新建文件

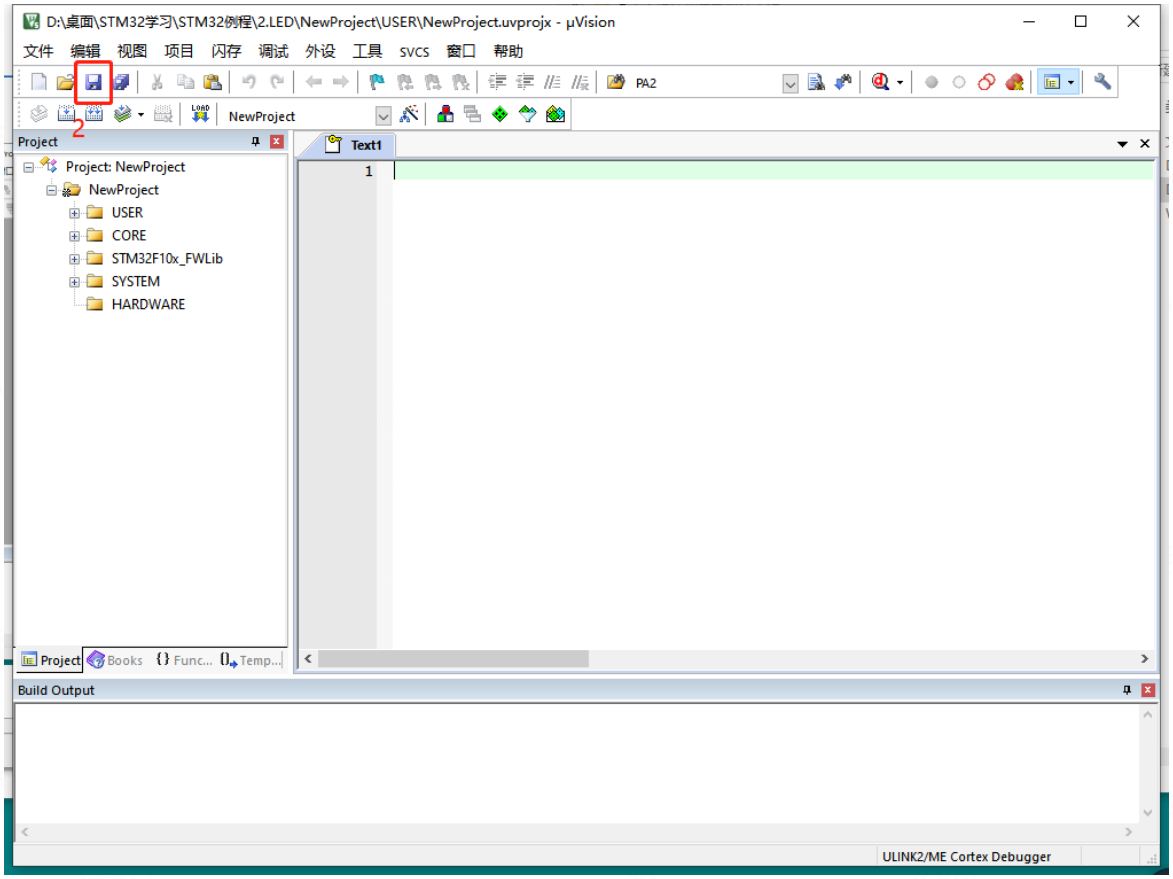

图 2-2-4 点击保存文件

在文件夹【HARDWARE】下，文件名为【LED.h】，点击【保存】。

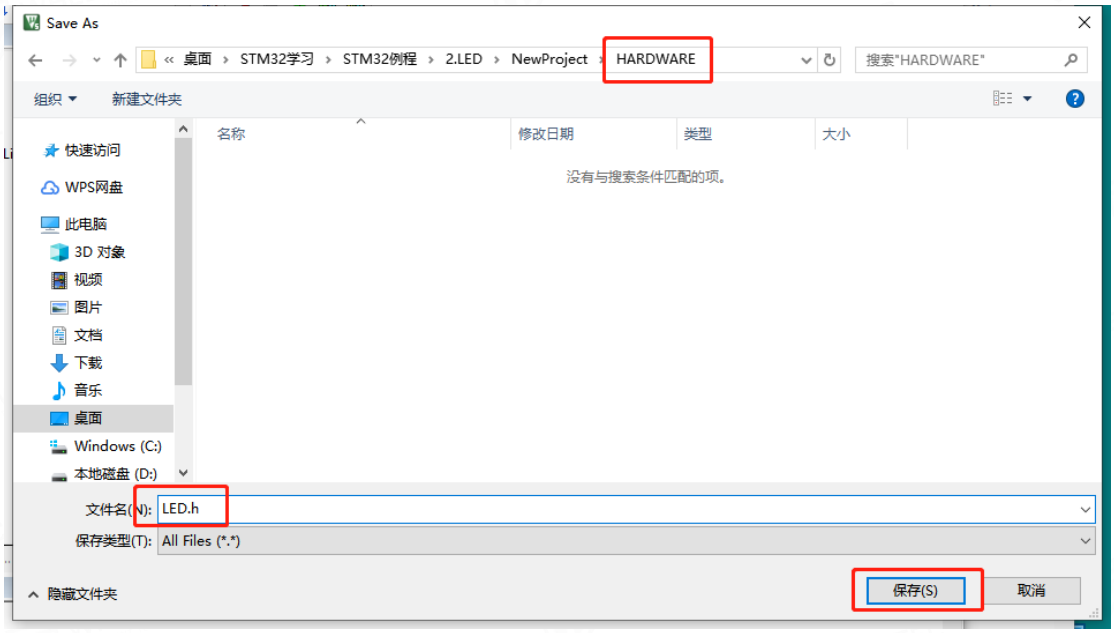

图 2-2-5 保存文件【LED.h】

重复类似前面的步骤，【新建文件】、【保存文件】，在文件夹【HARDWARE】下保存文件名为【LED.c】。

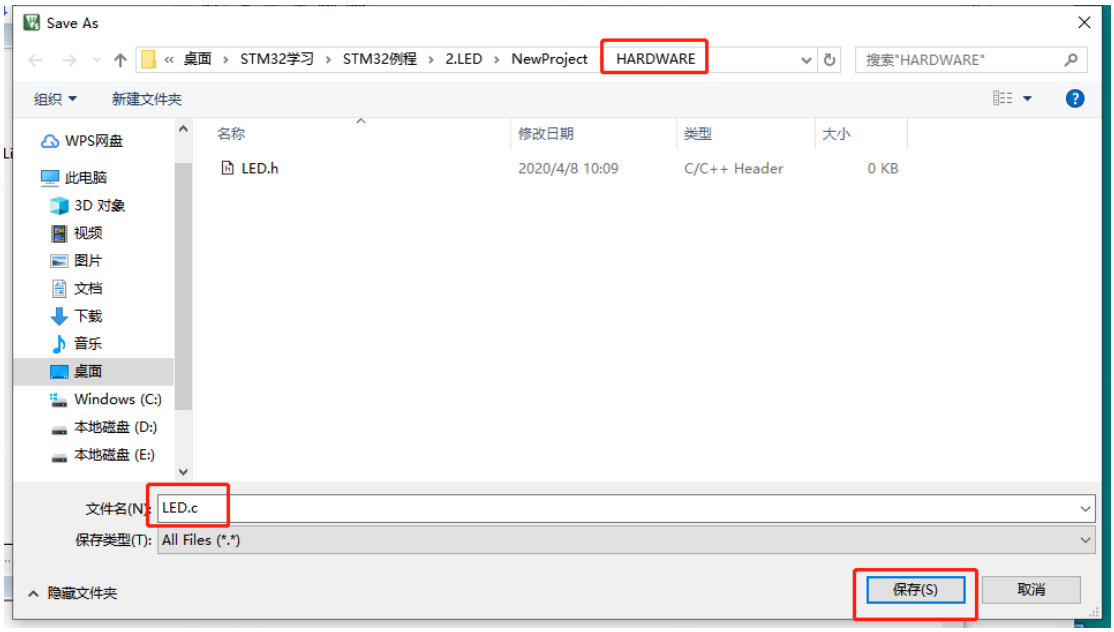

图 2-2-6 新建并保存文件【LED.c】

④ 添加库文件到分组

右键点击【HARDWARE】，单击【Add Existing Files to Group “HARDWARE”...】

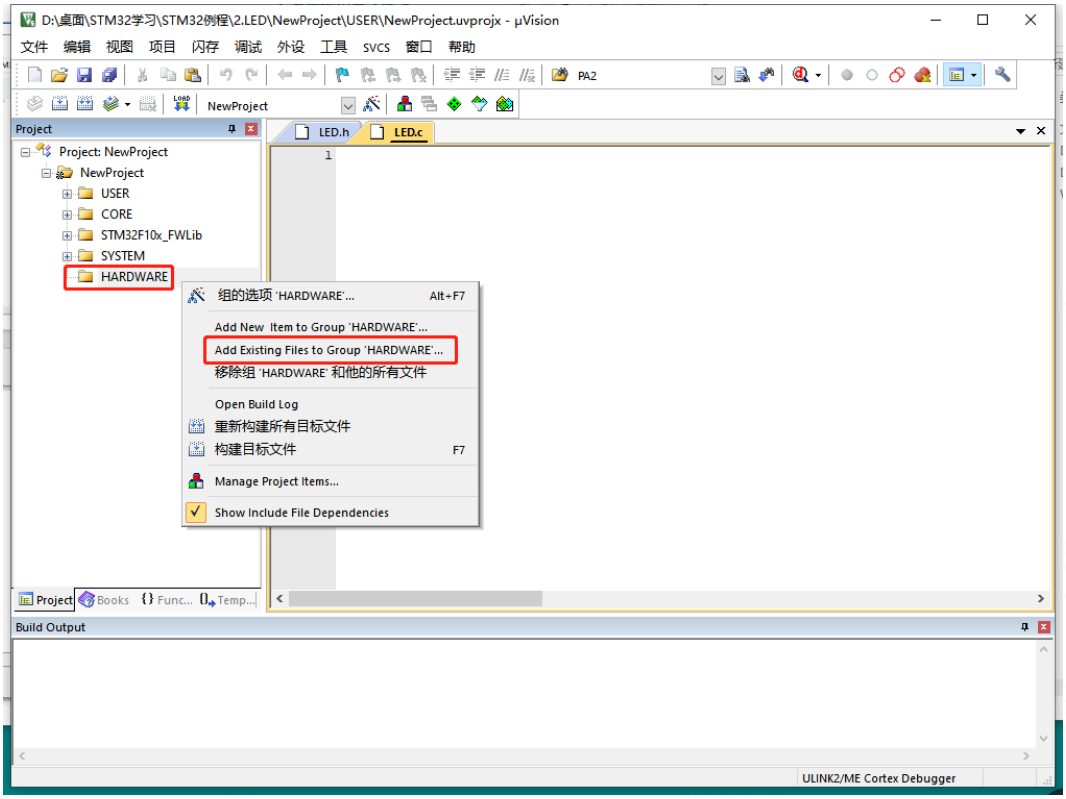

图 2-2-7 添加文件到【HARDWARE】分组

选择文件夹【HARDWARE】下的文件【LED.c】，单击【Add】。

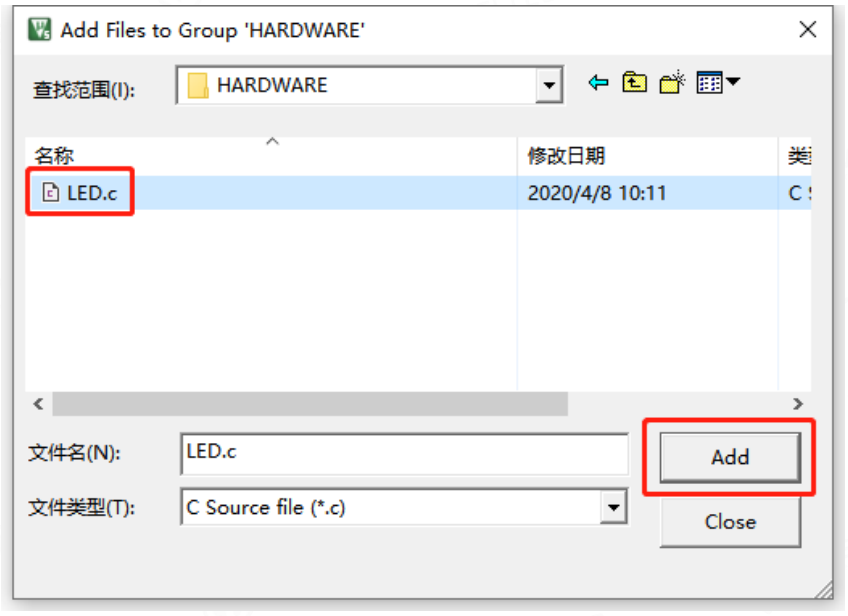

图 2-2-8 Add 源文件(.c 文件)

⑤ 添加头文件路径

下面添加头文件 LED.h 所在路径到工程。老规矩，【魔法棒】→【C/C++】→【...】。

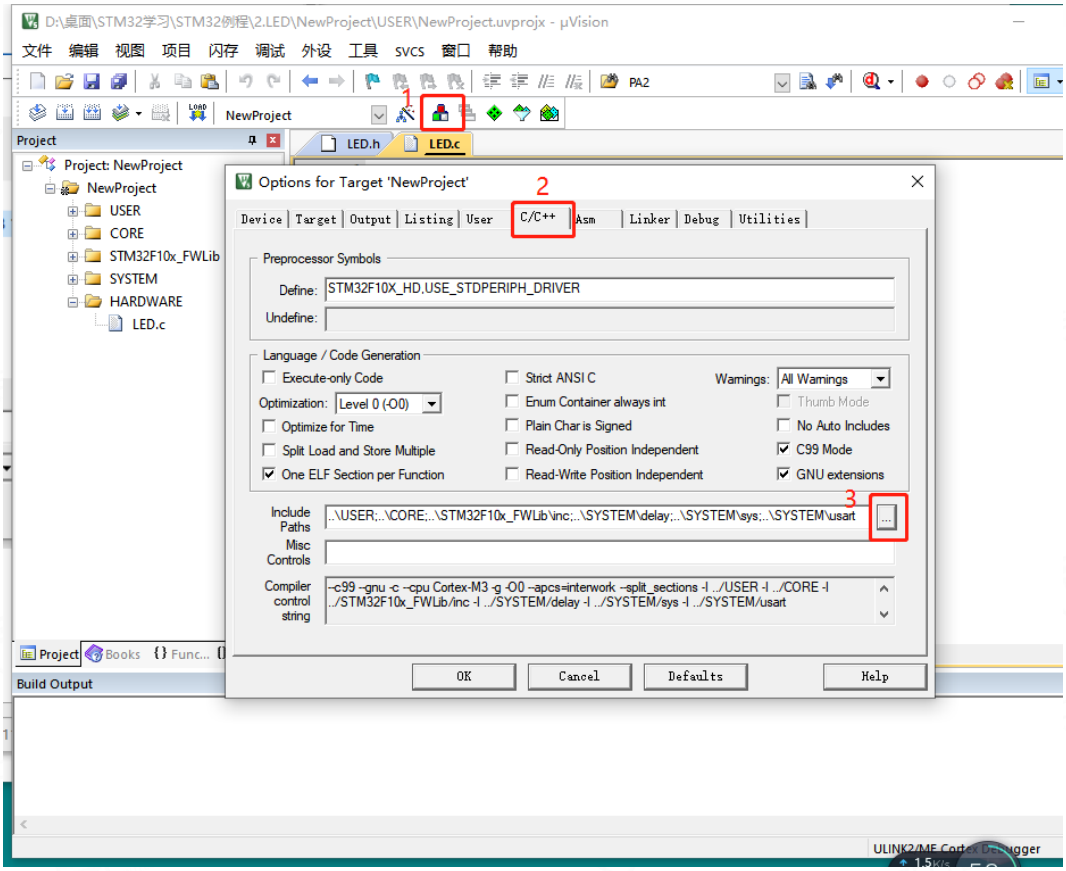

图 2-2-9 选择添加系统头文件(.h 文件)所在路径

单击 1 处，单击 2 处。

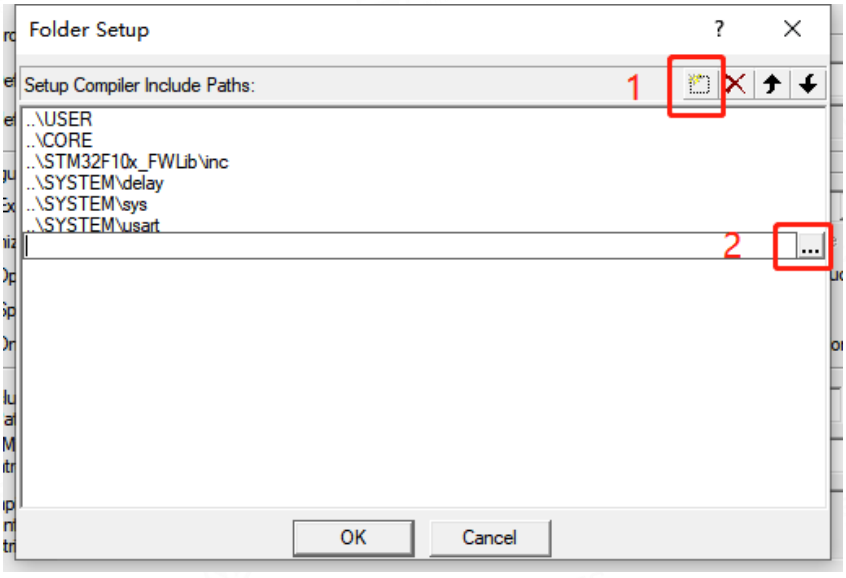

图 2-2-10 系统头文件(.h 文件)所在路径设置

在文件夹【HARDWARE】下，点击【选择文件夹】。然后点击【OK】，再次点击【OK】。

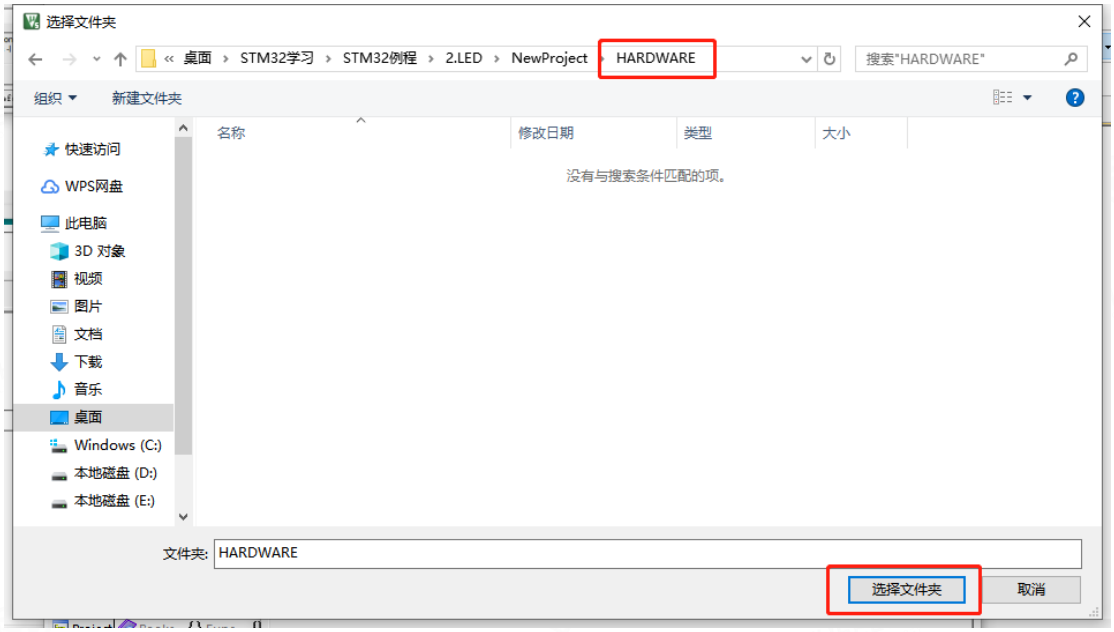

图 2-2-11 选择【HARDWARE】文件夹

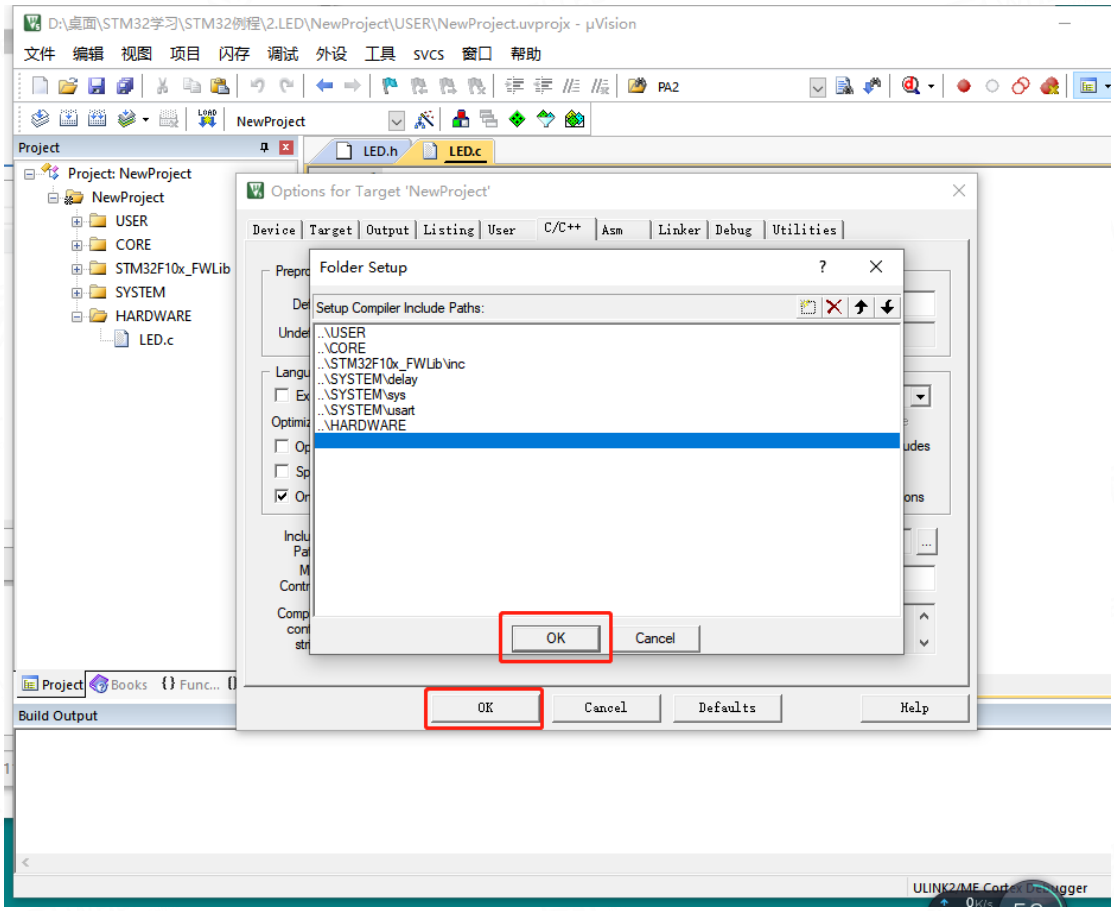

图 2-2-12 点击 OK

2.3 编写硬件外设库函数

① LED.h

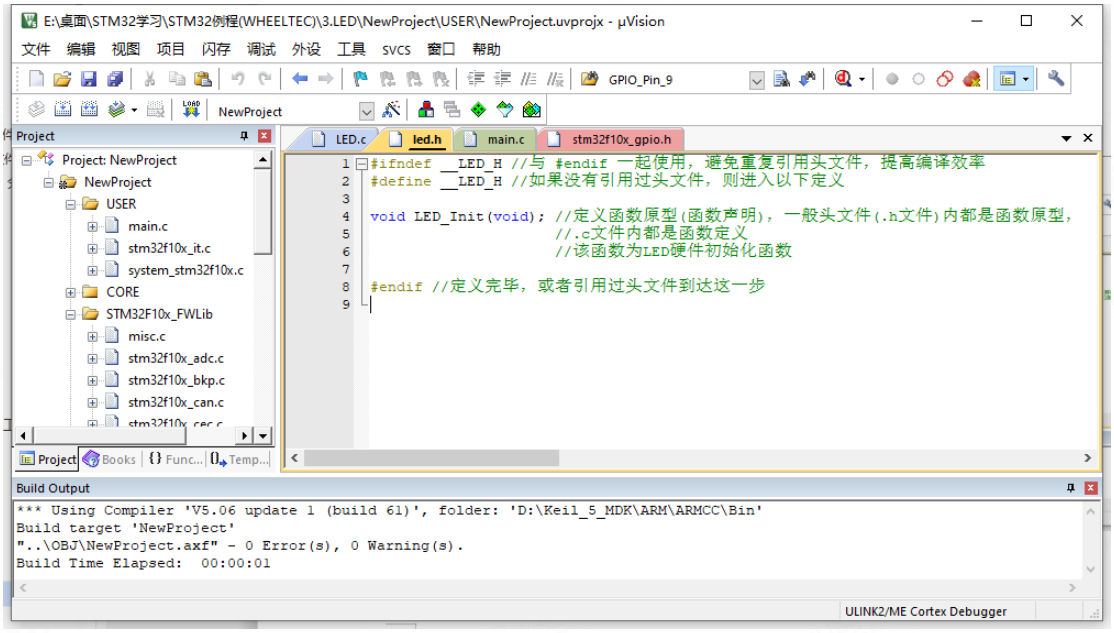

图 2-3-1 LED.h

注意，全文灰色阴影部分为程序。

```
#ifndef __LED_H  //与 #endif 一起使用，避免重复引用头文件，提高编译效率
#define __LED_H  //如果没有引用过头文件，则进入以下定义

void LED_Init(void);  //定义函数原型(函数声明)，一般头文件(.h 文件)内都是函数原型，.c 文件内都是函数定义，该函数为 LED 硬件初始化函数

#endif  //定义完毕，或者引用过头文件到达这一步
```

.h 文件称为头文件，#ifndef、#define 与#endif 一起使用，在#define 与#endif 中间写函数原型。

② LED.c

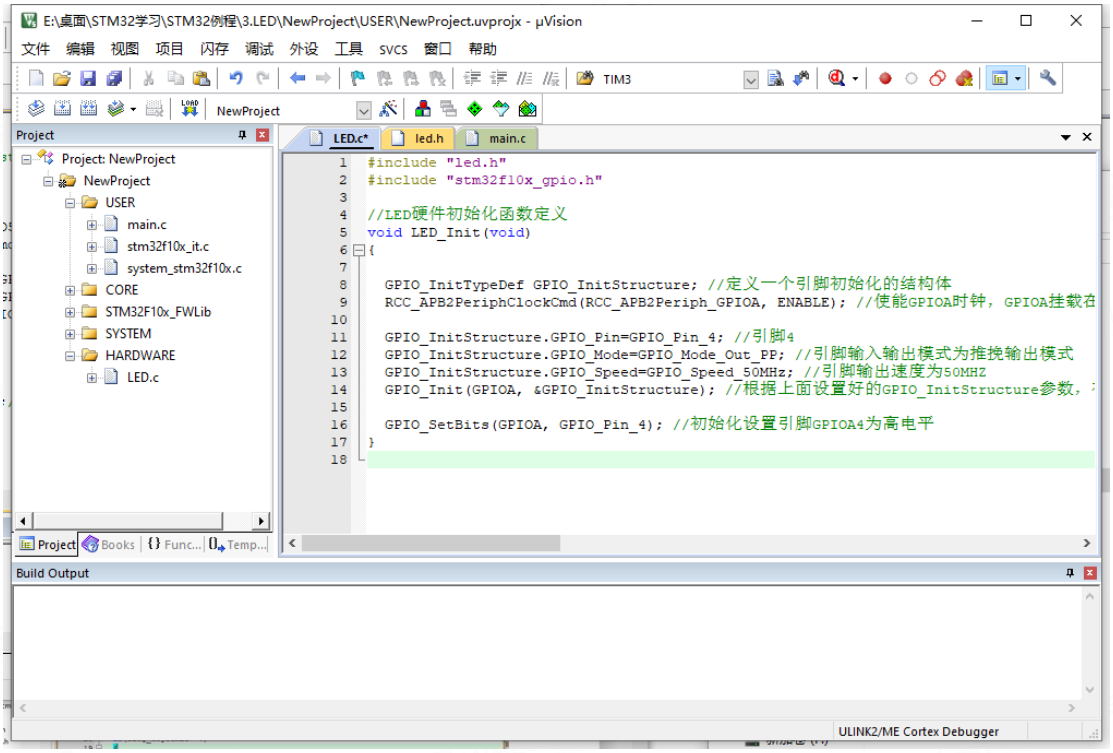

图 2-3-2 LED.c

```
#include "led.h"
#include "stm32f10x_gpio.h"

//LED 硬件初始化函数定义
void LED_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure; //定义一个引脚初始化的结构体
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使能 GPIOA 时钟,GPIOA
挂载在 APB2 时钟下，在 STM32 中使用 IO 口前都要使能对应时钟

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_4; //引脚 4
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //引脚输入输出模式为推挽输出
模式
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //引脚输出速度为 50MHZ
    GPIO_Init(GPIOA, &GPIO_InitStructure); //根据上面设置好的 GPIO_InitStructure
参数，初始化引脚 GPIOA_PIN4

    GPIO_SetBits(GPIOA, GPIO_Pin_4); //初始化设置引脚 GPIOA4 为高电平
}
```

.c 文件称为源文件，一般在这里写函数定义，如初始化函数、功能函数等，最后在主函数中调用。

硬件外设库函数的.c 文件内开头必须引用对应头文件，如 LED 灯的 LED.c 开头引用了 LED.h 头文件。由于函数 LED_Init()内使用了 STM32 官方库函数中

的 GPIO 部分，那么开头也必须引用 STM32 官方库函数：#include "stm32f10x_gpio.h" 。

(1) 各 GPIO、功能都有对应的时钟名字，有的挂载在 APB1 时钟下，有的挂载在 APB2 时钟下、：

```
#define RCC_APB1Periph_TIM2              ((uint32_t)0x00000001)
#define RCC_APB1Periph_TIM3              ((uint32_t)0x00000002)
#define RCC_APB1Periph_TIM4              ((uint32_t)0x00000004)
#define RCC_APB1Periph_TIM5              ((uint32_t)0x00000008)
#define RCC_APB1Periph_TIM6              ((uint32_t)0x00000010)
#define RCC_APB1Periph_TIM7              ((uint32_t)0x00000020)
#define RCC_APB1Periph_TIM12             ((uint32_t)0x00000040)
#define RCC_APB1Periph_TIM13             ((uint32_t)0x00000080)
#define RCC_APB1Periph_TIM14             ((uint32_t)0x00000100)
#define RCC_APB1Periph_WWDG              ((uint32_t)0x00000800)
#define RCC_APB1Periph_SPI2              ((uint32_t)0x00004000)
#define RCC_APB1Periph_SPI3              ((uint32_t)0x00008000)
#define RCC_APB1Periph_USART2            ((uint32_t)0x00020000)
#define RCC_APB1Periph_USART3            ((uint32_t)0x00040000)
#define RCC_APB1Periph_UART4             ((uint32_t)0x00080000)
#define RCC_APB1Periph_UART5             ((uint32_t)0x00100000)
#define RCC_APB1Periph_I2C1              ((uint32_t)0x00200000)
#define RCC_APB1Periph_I2C2              ((uint32_t)0x00400000)
#define RCC_APB1Periph_USB               ((uint32_t)0x00800000)
#define RCC_APB1Periph_CAN1              ((uint32_t)0x02000000)
#define RCC_APB1Periph_CAN2              ((uint32_t)0x04000000)
#define RCC_APB1Periph_BKP               ((uint32_t)0x08000000)
#define RCC_APB1Periph_PWR               ((uint32_t)0x10000000)
#define RCC_APB1Periph_DAC               ((uint32_t)0x20000000)
#define RCC_APB1Periph_CEC               ((uint32_t)0x40000000)
```

图 2-3-3 挂载在 APB1 时钟下各引脚、功能对应的时钟名字

```
#define RCC_APB2Periph_AFIO              ((uint32_t)0x00000001)
#define RCC_APB2Periph_GPIOA             ((uint32_t)0x00000004)
#define RCC_APB2Periph_GPIOB             ((uint32_t)0x00000008)
#define RCC_APB2Periph_GPIOC             ((uint32_t)0x00000010)
#define RCC_APB2Periph_GPIOD             ((uint32_t)0x00000020)
#define RCC_APB2Periph_GPIOE             ((uint32_t)0x00000040)
#define RCC_APB2Periph_GPIOF             ((uint32_t)0x00000080)
#define RCC_APB2Periph_GPIOG             ((uint32_t)0x00000100)
#define RCC_APB2Periph_ADC1              ((uint32_t)0x00000200)
#define RCC_APB2Periph_ADC2              ((uint32_t)0x00000400)
#define RCC_APB2Periph_TIM1              ((uint32_t)0x00000800)
#define RCC_APB2Periph_SPI1              ((uint32_t)0x00001000)
#define RCC_APB2Periph_TIM8              ((uint32_t)0x00002000)
#define RCC_APB2Periph_USART1            ((uint32_t)0x00004000)
#define RCC_APB2Periph_ADC3              ((uint32_t)0x00008000)
#define RCC_APB2Periph_TIM15             ((uint32_t)0x00010000)
#define RCC_APB2Periph_TIM16             ((uint32_t)0x00020000)
#define RCC_APB2Periph_TIM17             ((uint32_t)0x00040000)
#define RCC_APB2Periph_TIM9              ((uint32_t)0x00080000)
#define RCC_APB2Periph_TIM10             ((uint32_t)0x00100000)
#define RCC_APB2Periph_TIM11             ((uint32_t)0x00200000)
```

图 2-3-4 挂载在 APB2 时钟下各引脚、功能对应的时钟名字

(2) GPIO_Pin 有 17 个选项，分别对应某个 GPIO 的 0-15 号引脚和某个 GPIO 的全部引脚：

```
#define GPIO_Pin_0                 ((uint16_t)0x0001)  /*!< Pin 0 selected */
#define GPIO_Pin_1                 ((uint16_t)0x0002)  /*!< Pin 1 selected */
#define GPIO_Pin_2                 ((uint16_t)0x0004)  /*!< Pin 2 selected */
#define GPIO_Pin_3                 ((uint16_t)0x0008)  /*!< Pin 3 selected */
#define GPIO_Pin_4                 ((uint16_t)0x0010)  /*!< Pin 4 selected */
#define GPIO_Pin_5                 ((uint16_t)0x0020)  /*!< Pin 5 selected */
#define GPIO_Pin_6                 ((uint16_t)0x0040)  /*!< Pin 6 selected */
#define GPIO_Pin_7                 ((uint16_t)0x0080)  /*!< Pin 7 selected */
#define GPIO_Pin_8                 ((uint16_t)0x0100)  /*!< Pin 8 selected */
#define GPIO_Pin_9                 ((uint16_t)0x0200)  /*!< Pin 9 selected */
#define GPIO_Pin_10                ((uint16_t)0x0400)  /*!< Pin 10 selected */
#define GPIO_Pin_11                ((uint16_t)0x0800)  /*!< Pin 11 selected */
#define GPIO_Pin_12                ((uint16_t)0x1000)  /*!< Pin 12 selected */
#define GPIO_Pin_13                ((uint16_t)0x2000)  /*!< Pin 13 selected */
#define GPIO_Pin_14                ((uint16_t)0x4000)  /*!< Pin 14 selected */
#define GPIO_Pin_15                ((uint16_t)0x8000)  /*!< Pin 15 selected */
#define GPIO_Pin_All               ((uint16_t)0xFFFF)  /*!< All pins selected */
```

图 2-3-5 GPIO_Pin

(3) GPIO_Mode 有 8 个选项（即 GPIO 的 8 种工作模式）：

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。

```
typedef enum
{  GPIO_Mode_AIN = 0x0,
   GPIO_Mode_IN_FLOATING = 0x04,
   GPIO_Mode_IPD = 0x28,
   GPIO_Mode_IPU = 0x48,
   GPIO_Mode_Out_OD = 0x14,
   GPIO_Mode_Out_PP = 0x10,
   GPIO_Mode_AF_OD = 0x1C,
   GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

图 2-3-6 GPIO_Mode

(4) GPIO_Speed 有 3 个选项，GPIO_Speed_2MHz、GPIO_Speed_10MHz、GPIO_Speed_50MHz，表示输出口的最高频率，GPIO 口设为输入时，无须设置该选项。

```
typedef enum
{
  GPIO_Speed_10MHz = 1,
  GPIO_Speed_2MHz,
  GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
```

图 2-3-7 GPIO_Speed

2.4 查看函数定义、声明及入口变量

有时候我们会想知道函数的作用、函数的定义、入口变量的意义、入口变量的其它选项等，这时我们可以这样做：

① 查看函数定义、声明

以函数 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);为例，我们首先将鼠标指针放在函数名上。

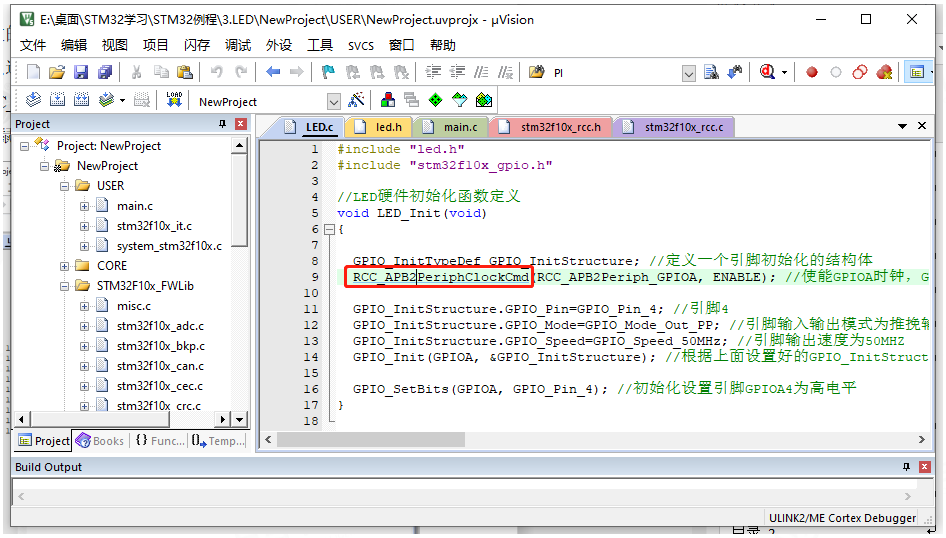

图 2-4-1 将鼠标指针放在函数名上

单击右键，选择【Go To Definition Of ‘xxxx’】。

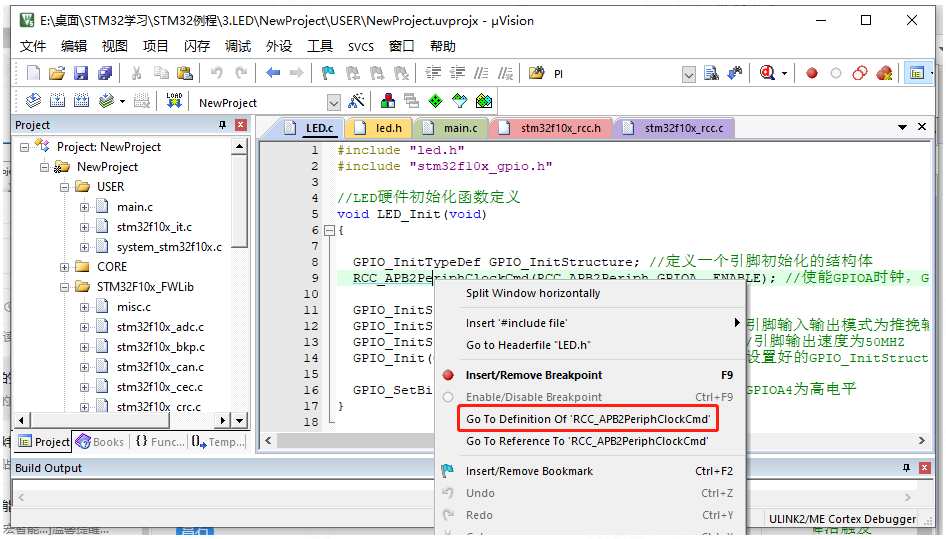

图 2-4-2 选择【Go To Definition Of ‘xxxx’】

即自动跳转到函数定义。

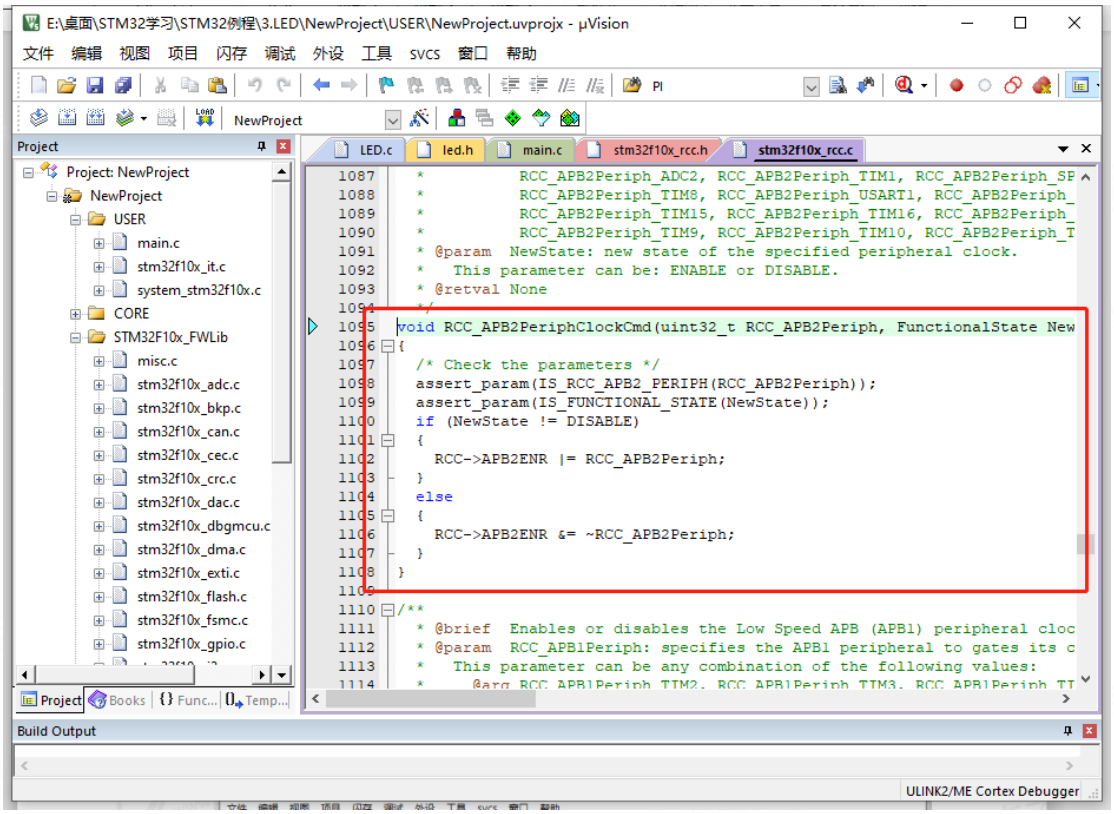

图 2-4-3 RCC_APB2PeriphClockCmd 的函数定义

如果右键单击后选择的是【Go To Reference To ‘xxxx’】,则跳转到函数声明(函数原型)。

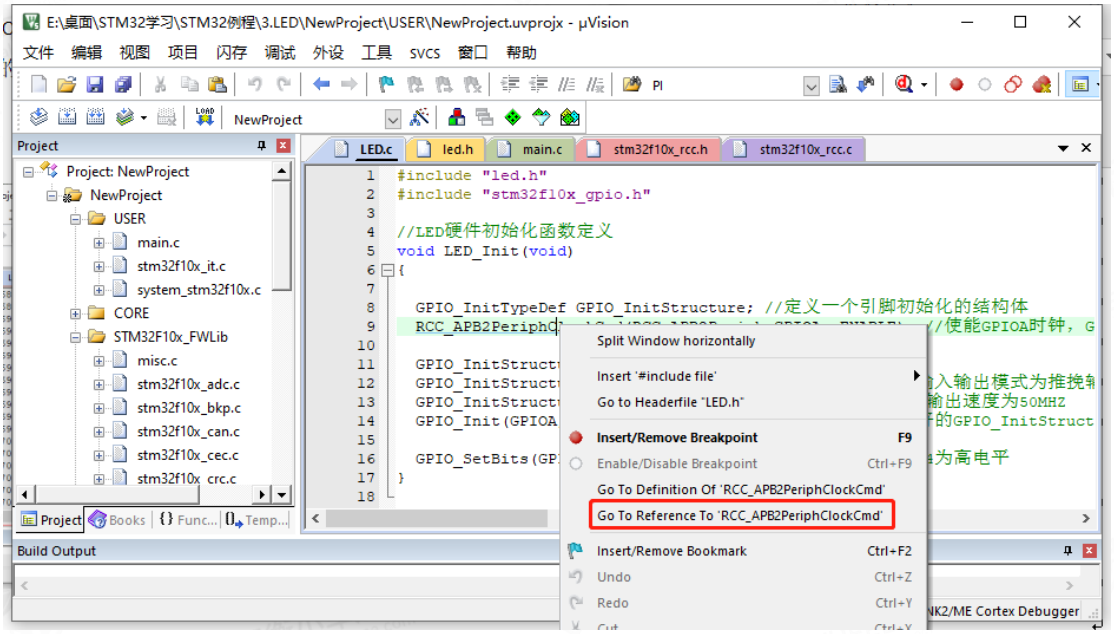

图 2-4-4 选择【Go To Reference To ‘xxxx’】

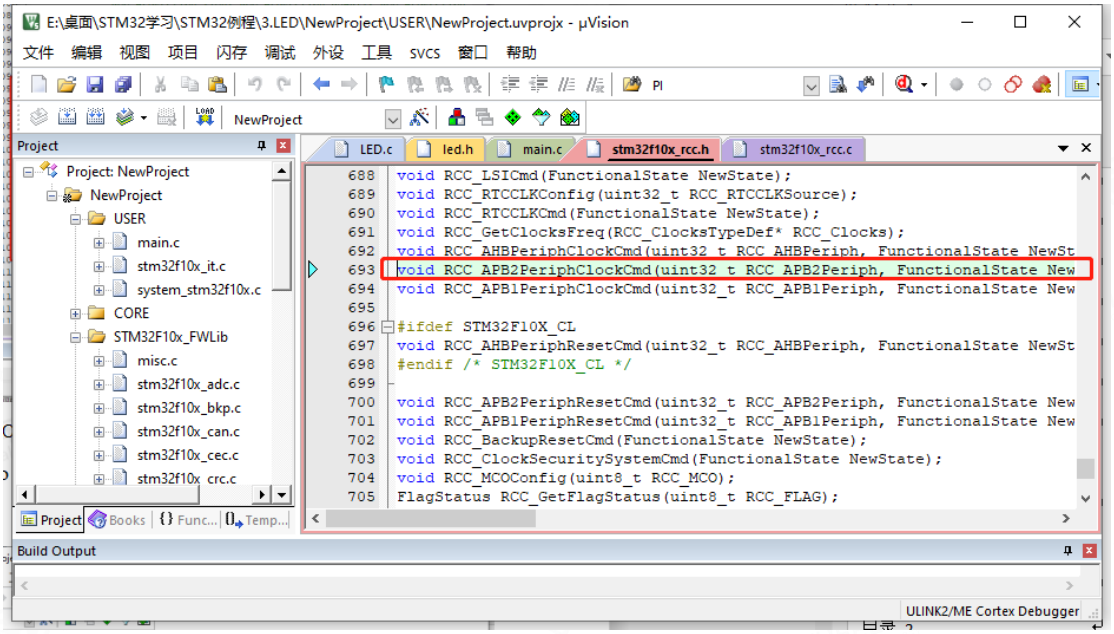

图 2-4-5 RCC_APB2PeriphClockCmd 的函数声明

② 查看入口变量、结构体成员

以函数 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);的 RCC_APB2Periph_GPIOA 为例查看函数入口变量，我们首先将鼠标指针放在变量名上。

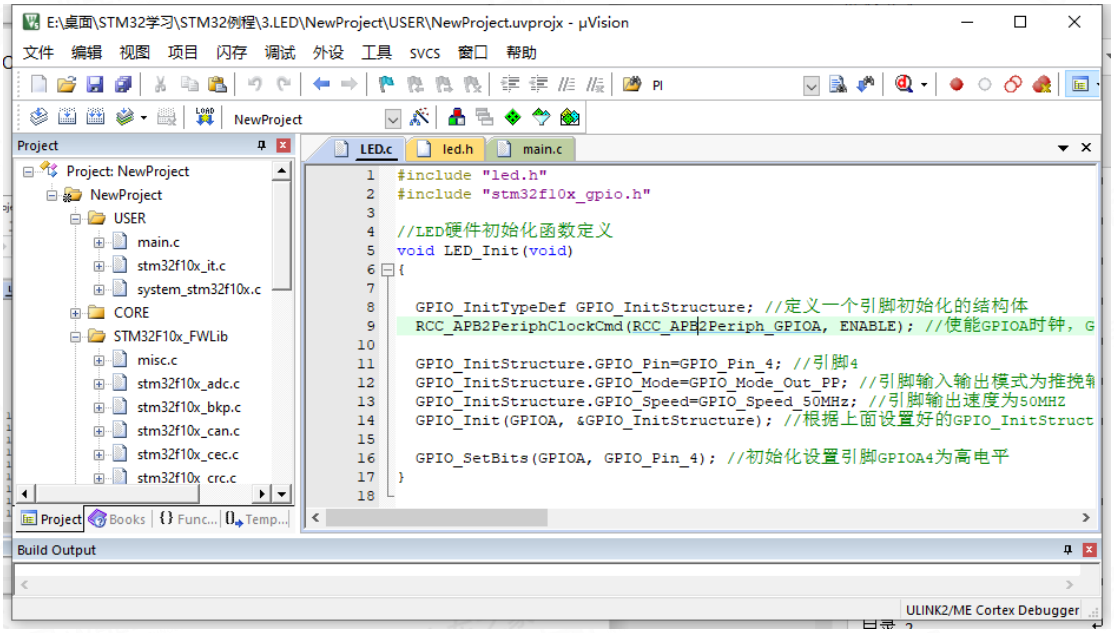

图 2-4-6 将鼠标指针放在变量名上

单击右键，选择【Go To Definition Of ‘xxxx’】。

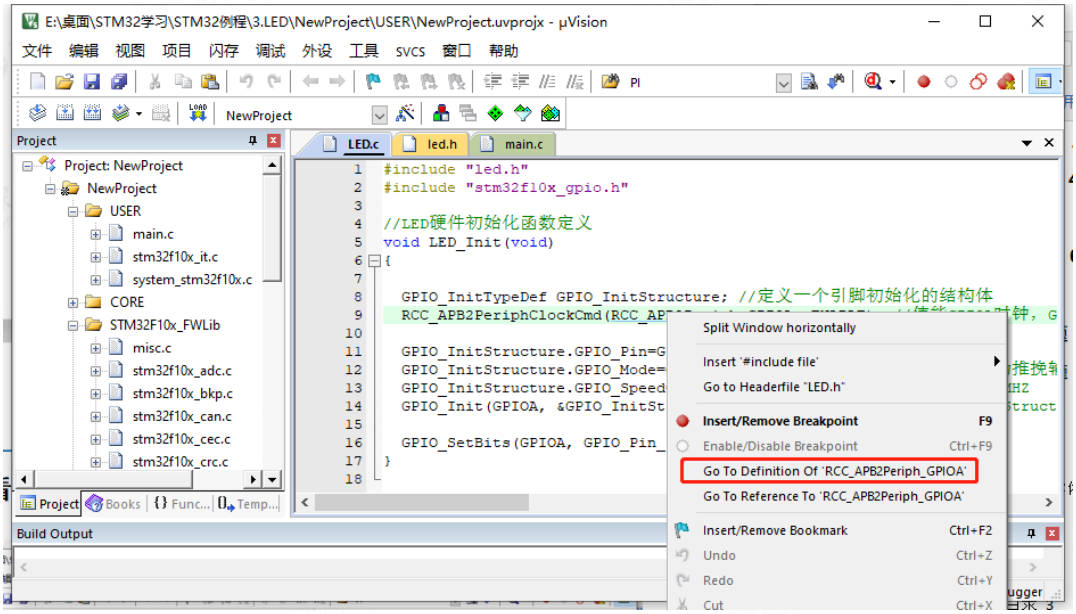

图 2-4-7 选择【Go To Definition Of ‘xxxx’】

即自动跳转到变量定义，并且可以看到附近有其它功能的时钟名，而它们都是挂载在 APB2 时钟下的，则函数 RCC_APB2PeriphClockCmd 的第一个入口变量可以为这些时钟名，再往下拉可以看见挂载在 APB1 时钟下的其它时钟。

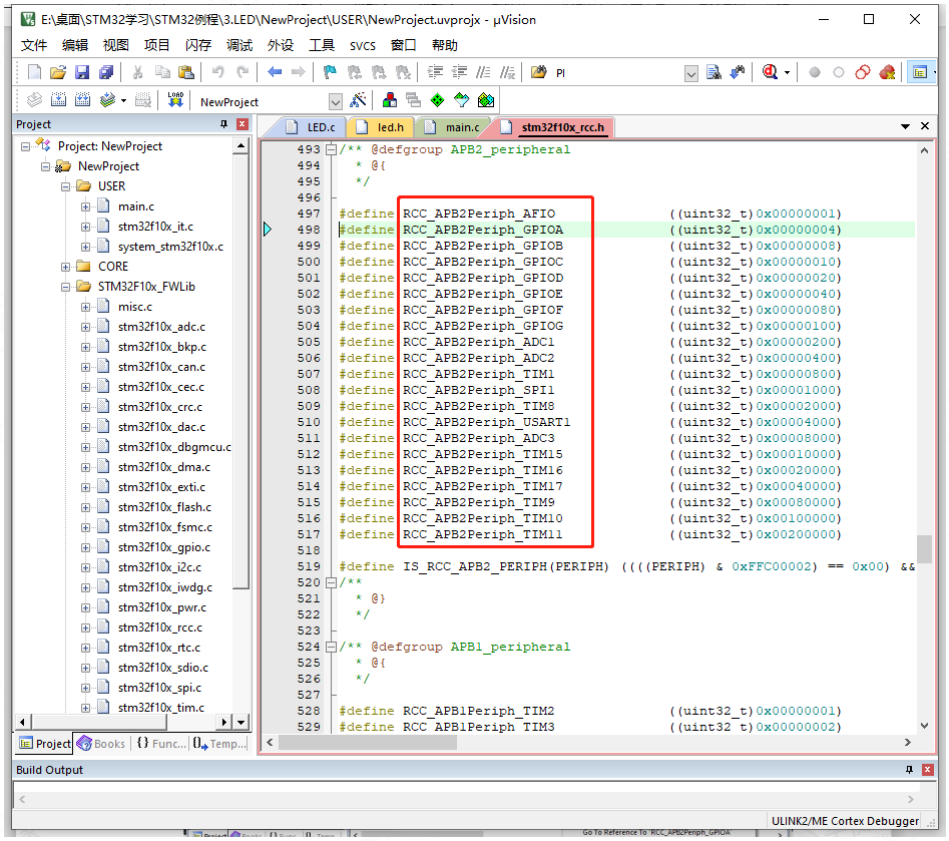

图 2-4-8 挂载在 APB2 时钟下的功能时钟名

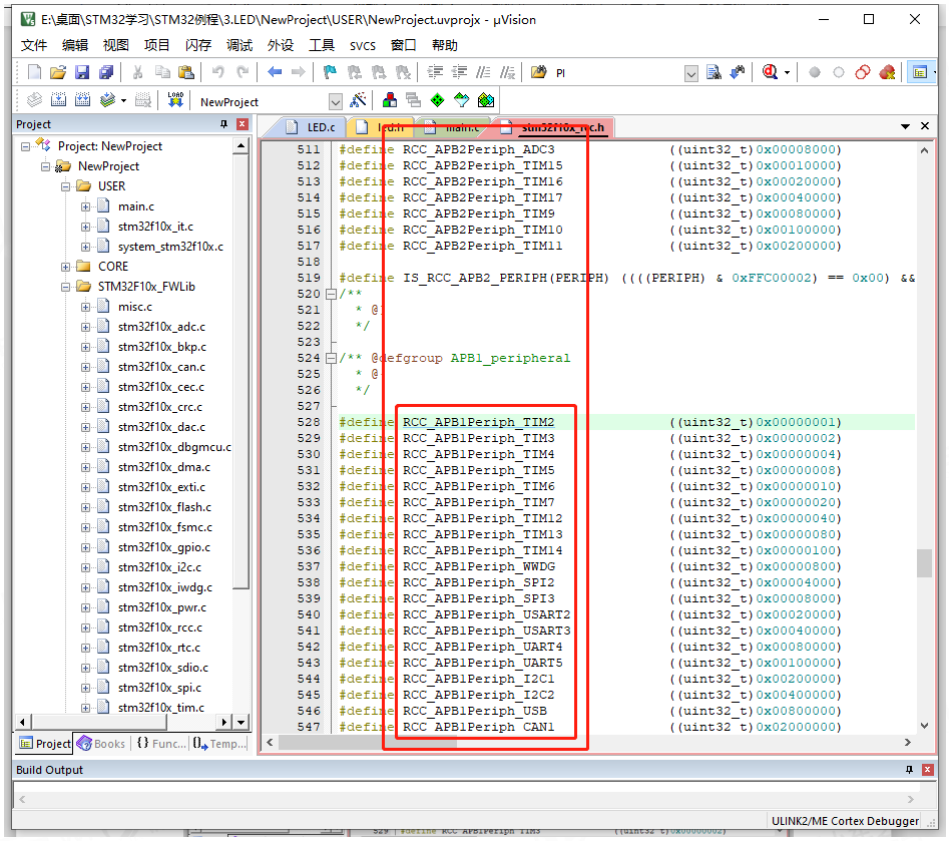

图 2-4-9 挂载在 APB1 时钟下的功能时钟名

即通过【Go To Definition Of ‘xxxx’】我们可以查看函数入口变量其它可能的值。

结构体成员可能的值我们也可以通过在结构体成员变量上右键单击然后选择【Go To Definition Of ‘xxxx’】进行查看，如 GPIO_InitStructure.GPIO_Mode 可能的值：在【GPIO_Mode_Out_PP】上右键单击然后选择【Go To Definition Of ‘xxxx’】。

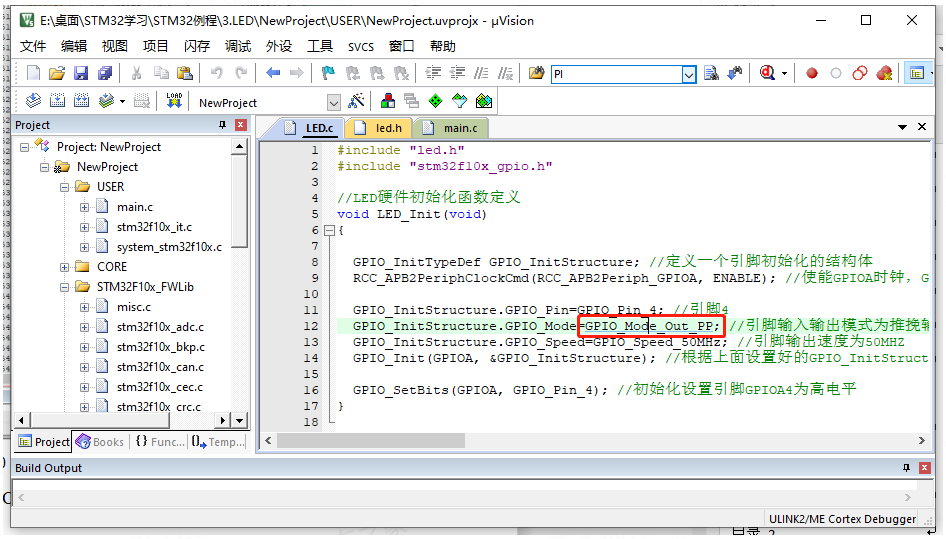

图 2-4-10 在【GPIO_Mode_Out_PP】上右键单击然后选择【Go To Definition Of ‘xxxx’】

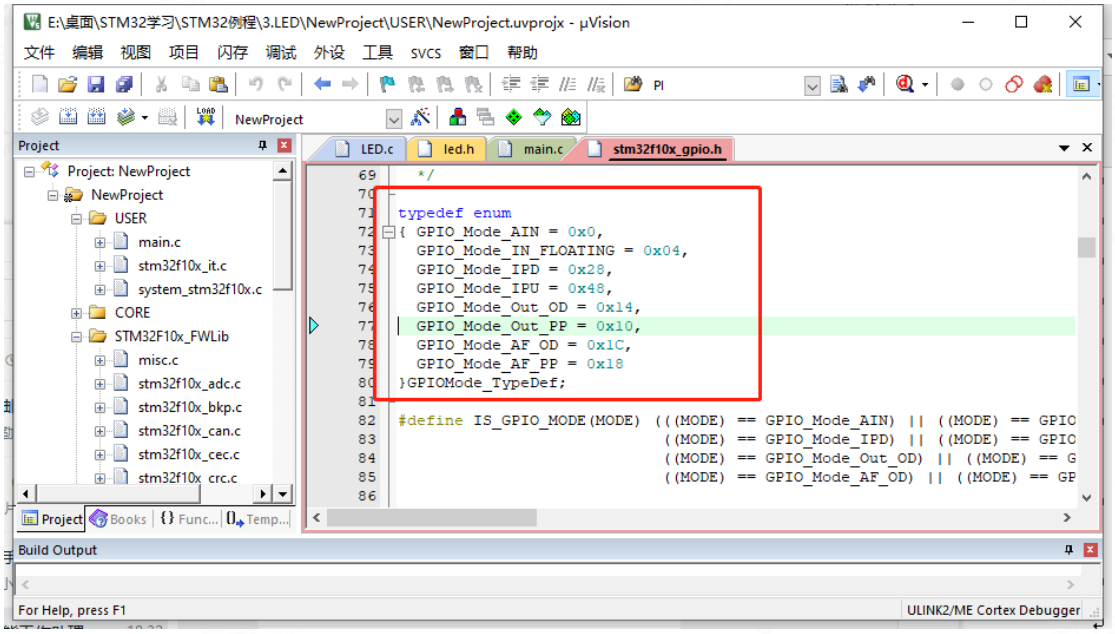

图 2-4-11 GPIO_InitStructure.GPIO_Mode 可能的值

2.5 编写主函数

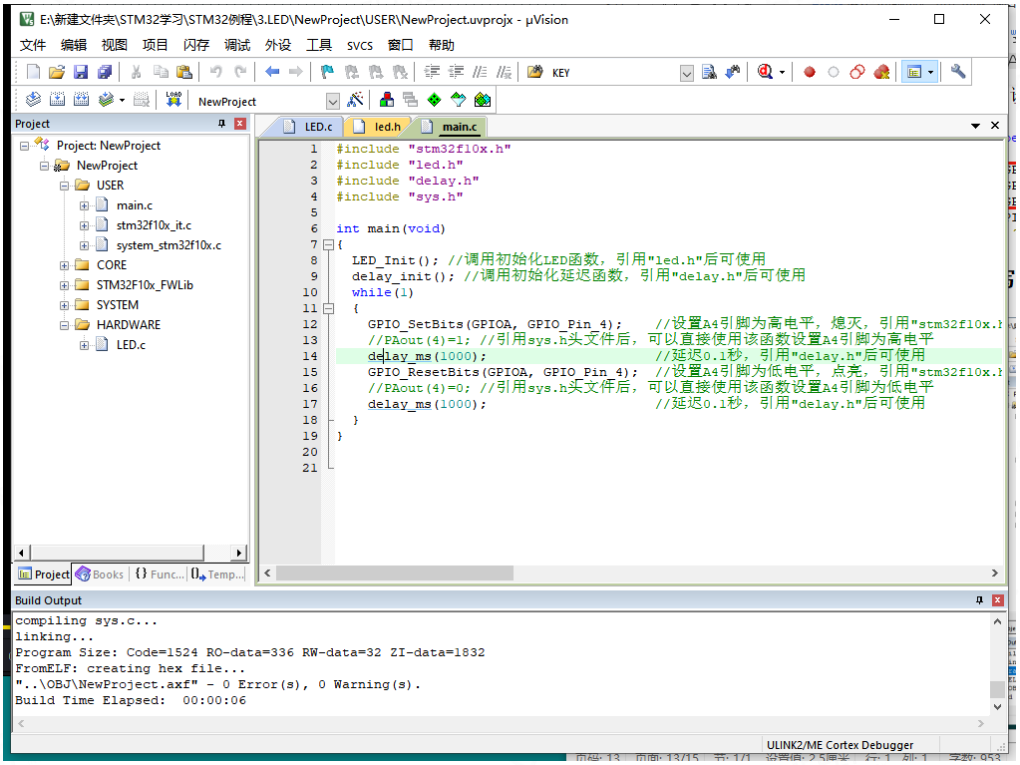

图 2-5-1 主函数

```
#include "stm32f10x_gpio.h"
#include "led.h"
#include "delay.h"
```

```
#include "sys.h"
```

由于使用了 stm32f10x_gpio.h、led.h、delay.h 三个库的函数，所以都要进行引用。

最终只有放在 main 函数里的函数才会被执行。

```
int main(void)
{
    LED_Init(); //调用初始化 LED 函数，引用"led.h"后可使用
    delay_init(); //调用初始化延迟函数，引用"delay.h"后可使用
```

在使用一些功能前，必须进行相应初始化，如接下来要使用延迟函数 delay，那么就要调用函数 delay_init();进行初始化；接下来要控制 LED 外设，那么调用 LED 初始化函数 LED_Init(); 进行初始化。

```
  while(1)
    {
      GPIO_SetBits(GPIOA, GPIO_Pin_4);     //GPIOA_PIN4 输出高电平，LED 灯熄灭，引
用"stm32f10x_gpio.h"后可使用
        //PAout(4)=1; //引用 sys.h 头文件后，可以直接使用该宏定义操作
IO(GPIOA_PIN4)输出高电平
        delay_ms(1000);                      //延迟 1 秒，引用"delay.h"后可使用
        GPIO_ResetBits(GPIOA, GPIO_Pin_4);  //设置 A4 引脚为低电平，点亮，引用
"stm32f10x.h"后可使用
        //PAout(4)=0; //引用 sys.h 头文件后，可以直接使用该宏定义操作
IO(GPIOA_PIN4)输出低电平
        delay_ms(1000);                      //延迟 1 秒，引用"delay.h"后可使用
    }}
```

2.6 上传程序到 STM32F103C8T6 核心板

用 USB 数据线连接 STM32F103C8T6 核心板与电脑。

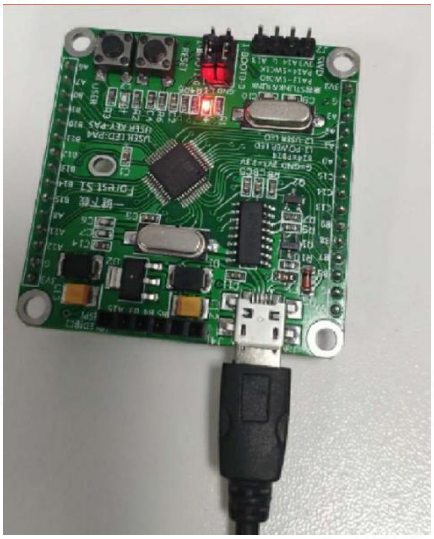

图 2-6-1 USB 连接电脑

双击打开软件【FlyMcu.exe】。该软件可以在我们提供的例程文件夹或者压缩包【Keil uvision5 MDK 版.zip】中找到。

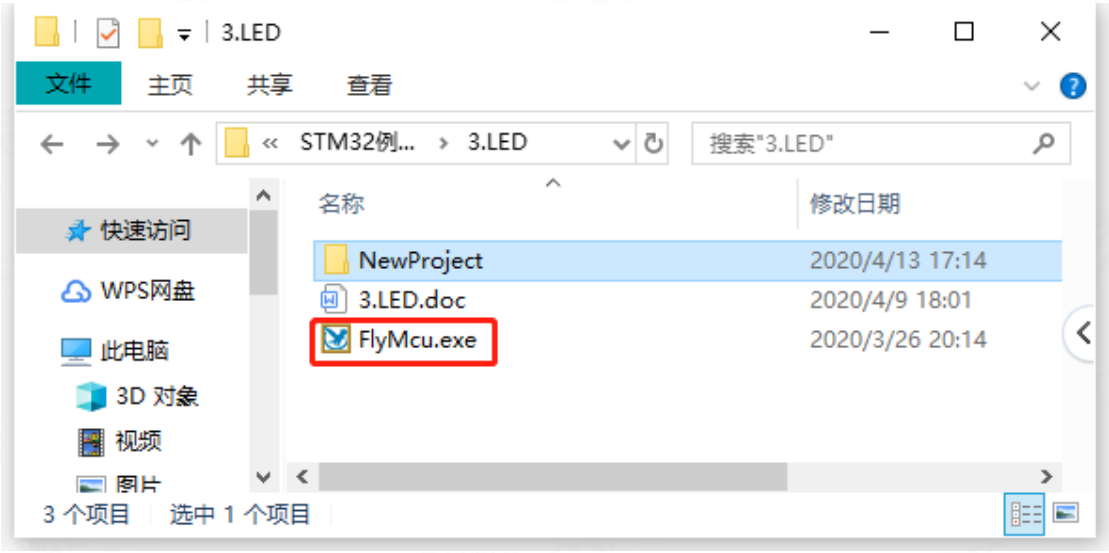

图 2-6-2 双击打开软件

点击【搜索串口(V)】，选择您刚连接核心板的 USB 串口。

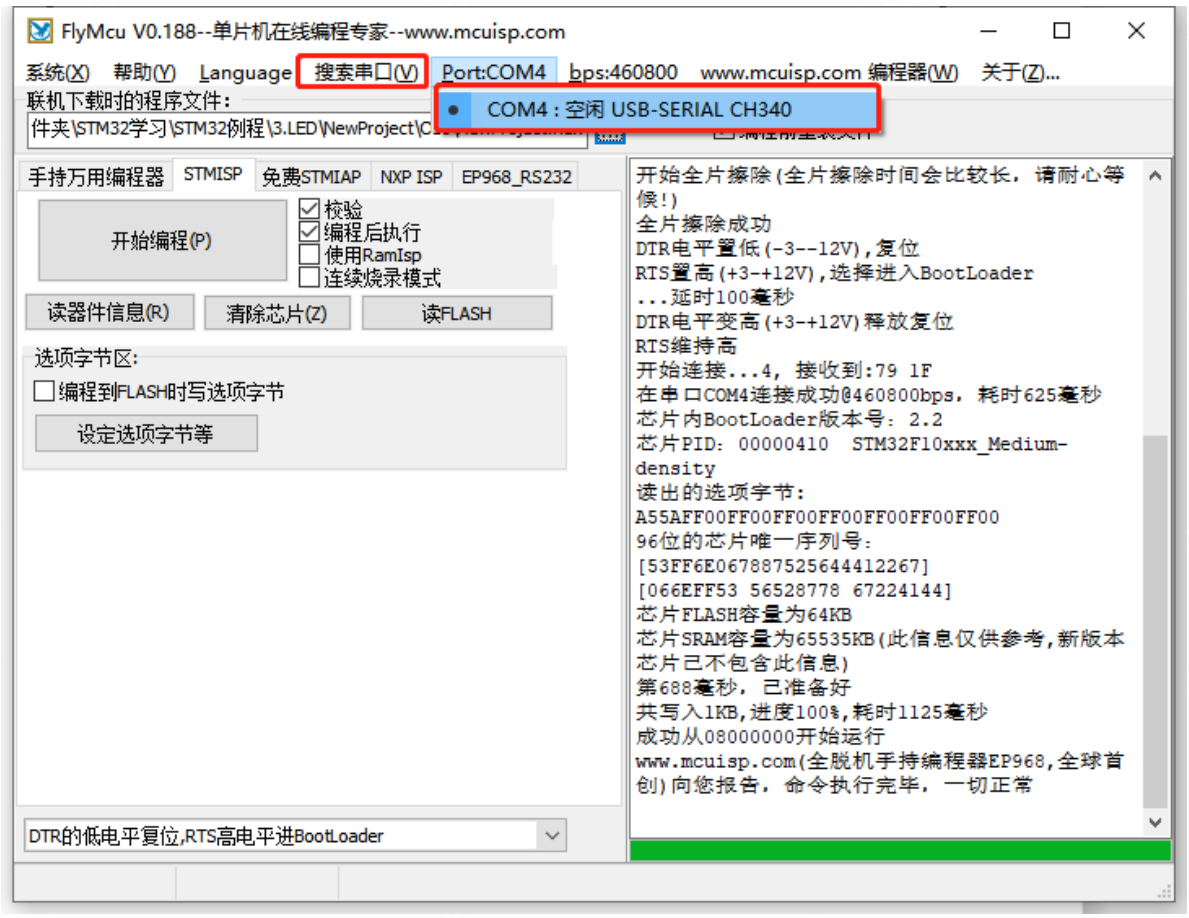

图 2-6-3 选择串口

勾选 1、2 处，在 3 处选择【DTR 的低电平复位,RTS 高电平进 BootLoader】，点击【...】以选择要上传的文件。

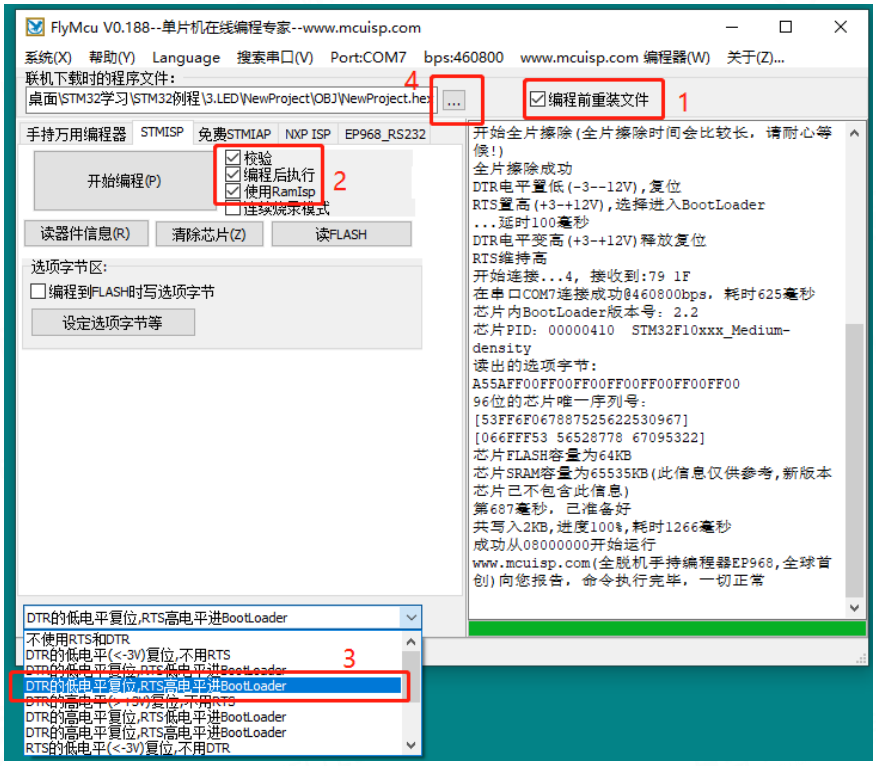

图 2-6-4 依次点击选择 1、2、3、4

选择【OBJ】文件夹下的文件【NewProject.hex】，点击打开。我们上传到核心板的文件就是.hex 文件。

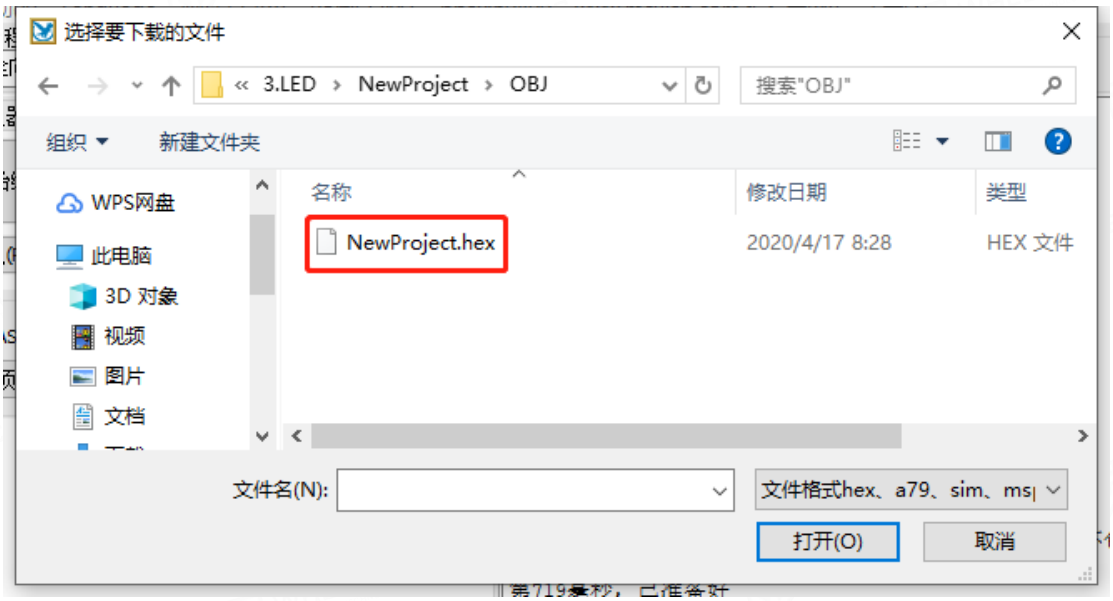

图 2-6-5 选择打开文件【NewProject.hex】

点击【开始编程(P)】，上传完成。

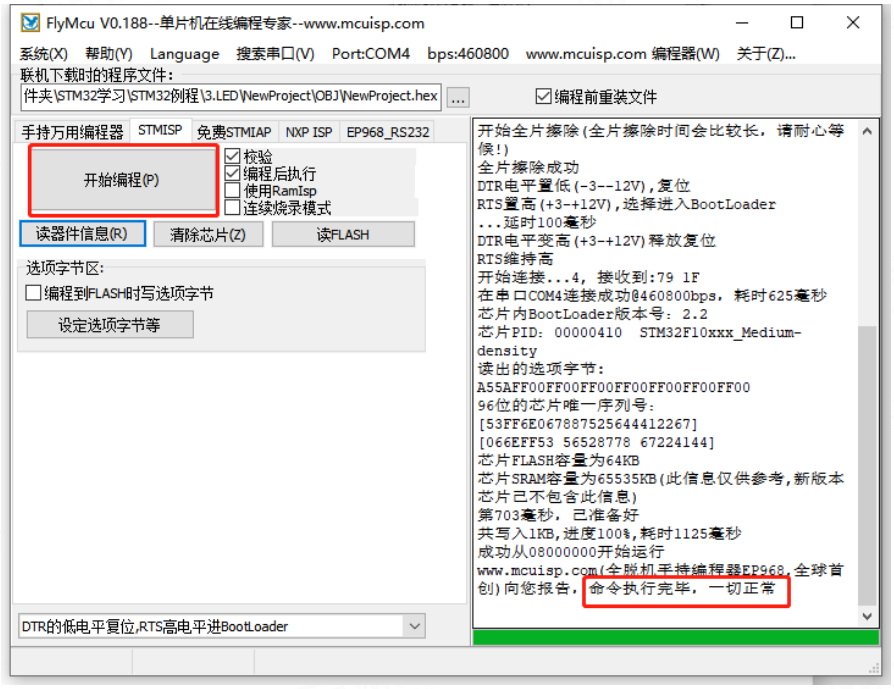

图 2-6-6 点击【开始编程】

2.7 实现思路及效果

(1) 编写头文件，声明初始化 IO 函数，编写源文件，基于官方的 GPIO 库函数定义 GPIO 初始化函数：

首先使能 GPIO 对应时钟，然后在引脚初始化结构体中定义好要初始化的引脚、引脚输出模式、引脚输出速度，最后根据结构体设置对应 GPIO.

(2) 在 while 循环中控制 IO 输出高低电平，以及设置合适的延时时间，形成 LED 闪烁的效果。

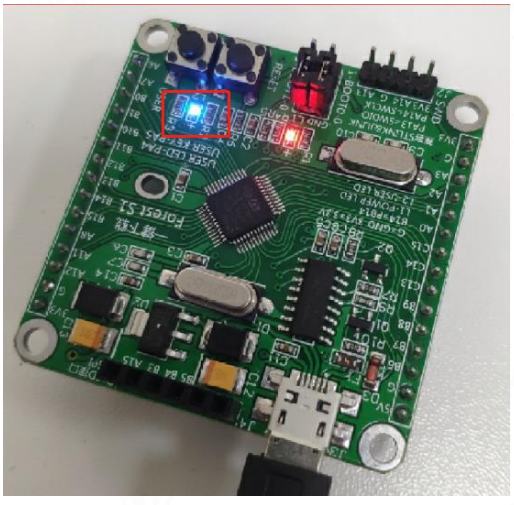

图 2-7-1 实验效果

2.8 本节知识要点

- 硬件外设库函数的创建与使用。
- GPIO 的初始化过程，开启对应时钟→IO 设置→IO 模式→IO 速度→配置 IO。
- GPIO 设置高低输出电平。

3. 按键控制 LED

本次例程使用按键来控制 LED 灯的亮灭。

3.1 相关 IO 介绍

Forest S1 上的按键有 1 个，受引脚 PA5 控制，未按下时受上拉电阻作用，PA5 检测到高电平，按键按下时接地，PA5 检测到低电平。

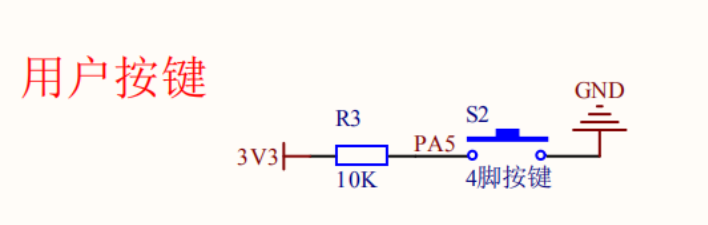

图 3-1-1 四脚按键引脚接线图

Forest S1 上可控制的 LED 灯有 1 个，为蓝色 LED 灯，受引脚 PA4 控制，低电平时点亮。

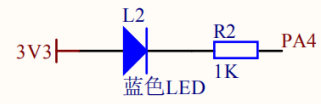

图 3-1-2 蓝色 LED 灯引脚接线图

3.2 为工程重命名

如果每一个新工程都要重新创建的话，那将花费不少时间。我们可以直接复制 LED 例程的工程，因为本次例程为使用按键，我们将文件夹【NewProject】重命名为【Key】。

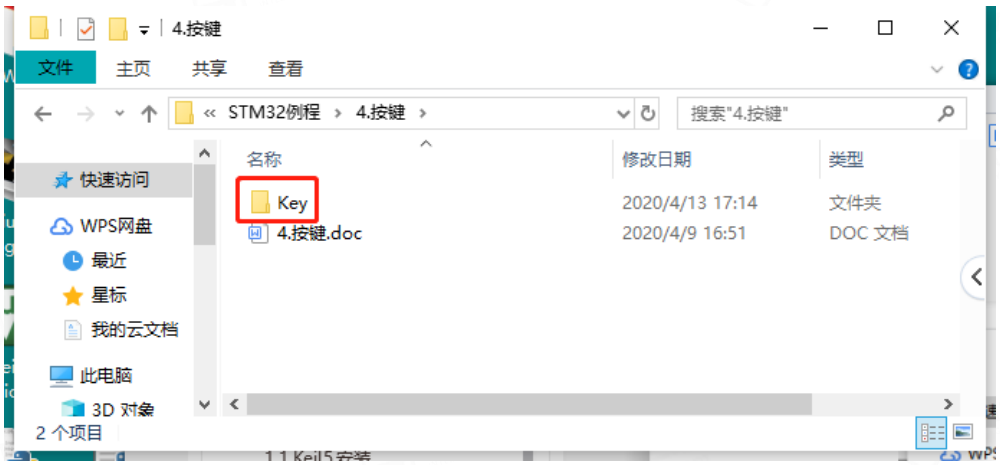

图 3-2-1 重命名文件夹

然后将文件夹【USER】下的文件【NewProject.uvprojx】重命名为【Key.uvprojx】。

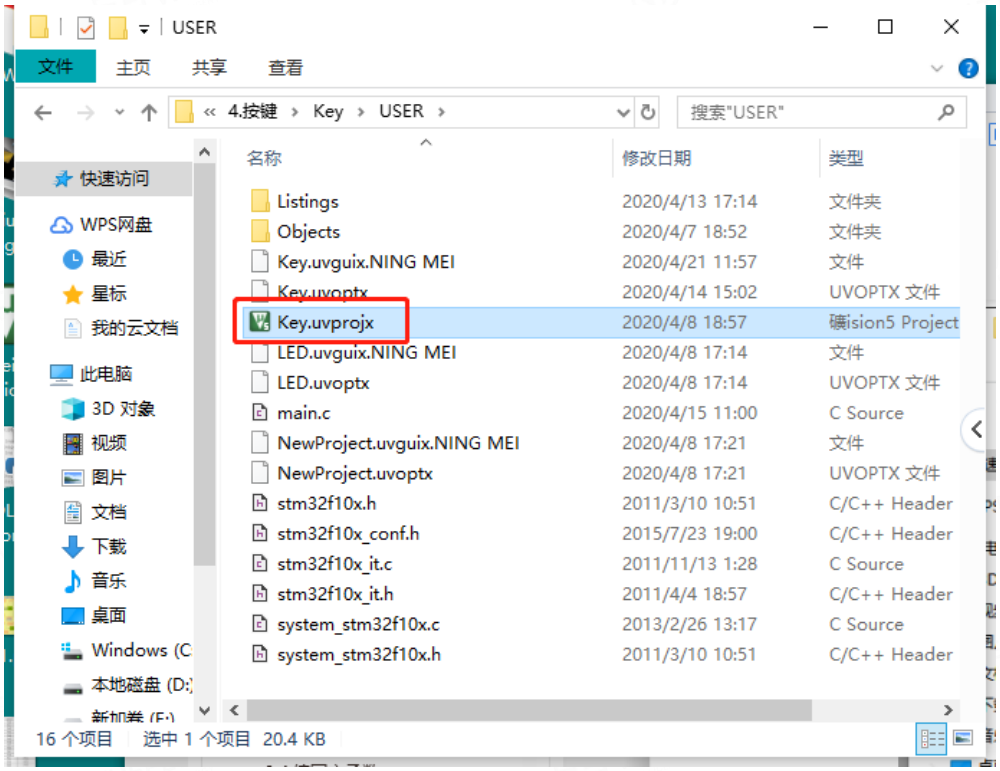

图 3-2-2 重命名文件

双击【Key.uvprojx】打开工程，右键点击【NewProject】，点击【Manage Project Items...】。

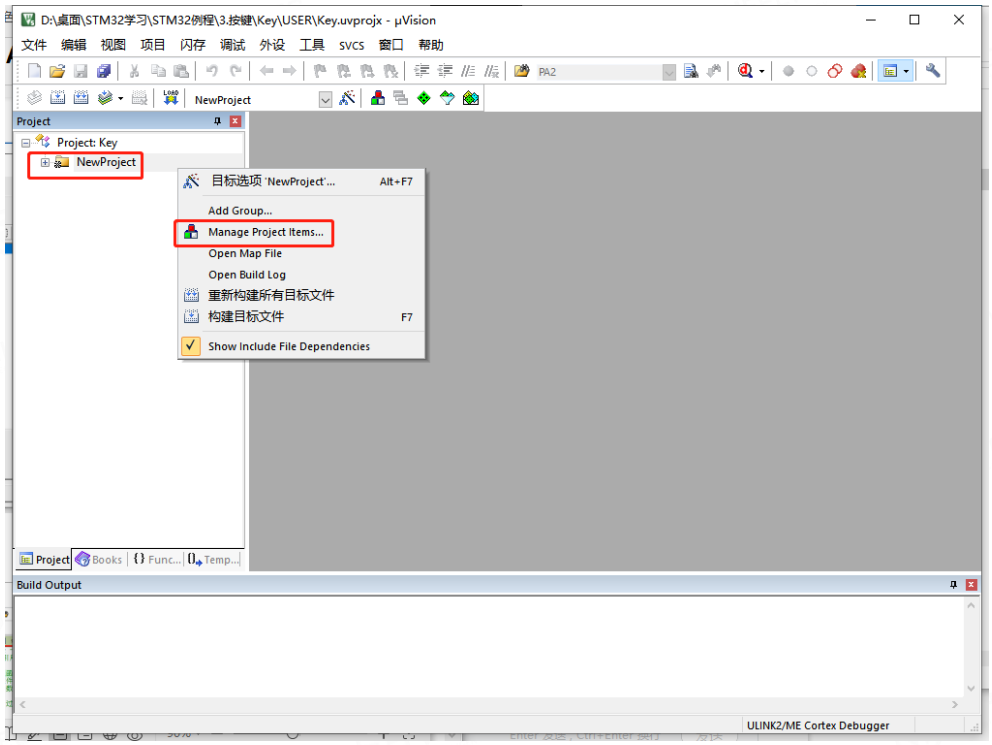

图 3-2-3 选择项目管理器

将【Project Targets】下的【New Project】重命名为【Key】。

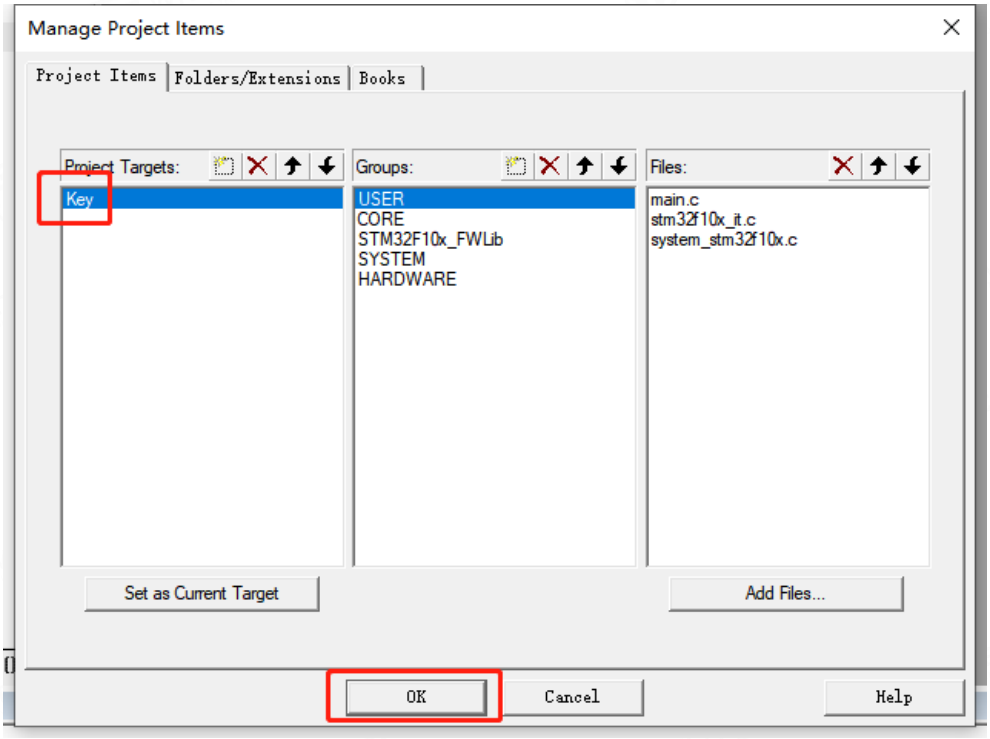

图 3-2-4 重命名项目

则新工程【Key】，创建成功。

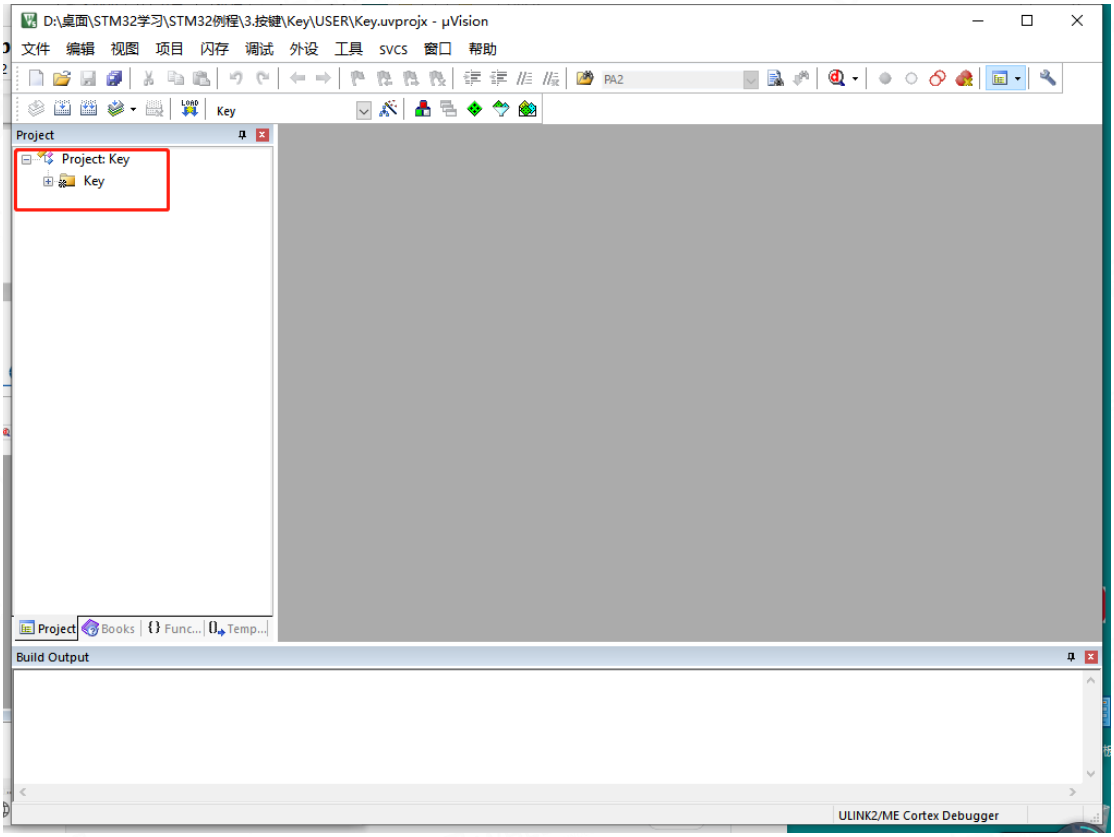

图 3-2-5 工程【Key】

3.3 编写硬件外设库函数

创建按键库函数【KEY.h】、【KEY.c】，并将其添加进工程。

① KEY.h

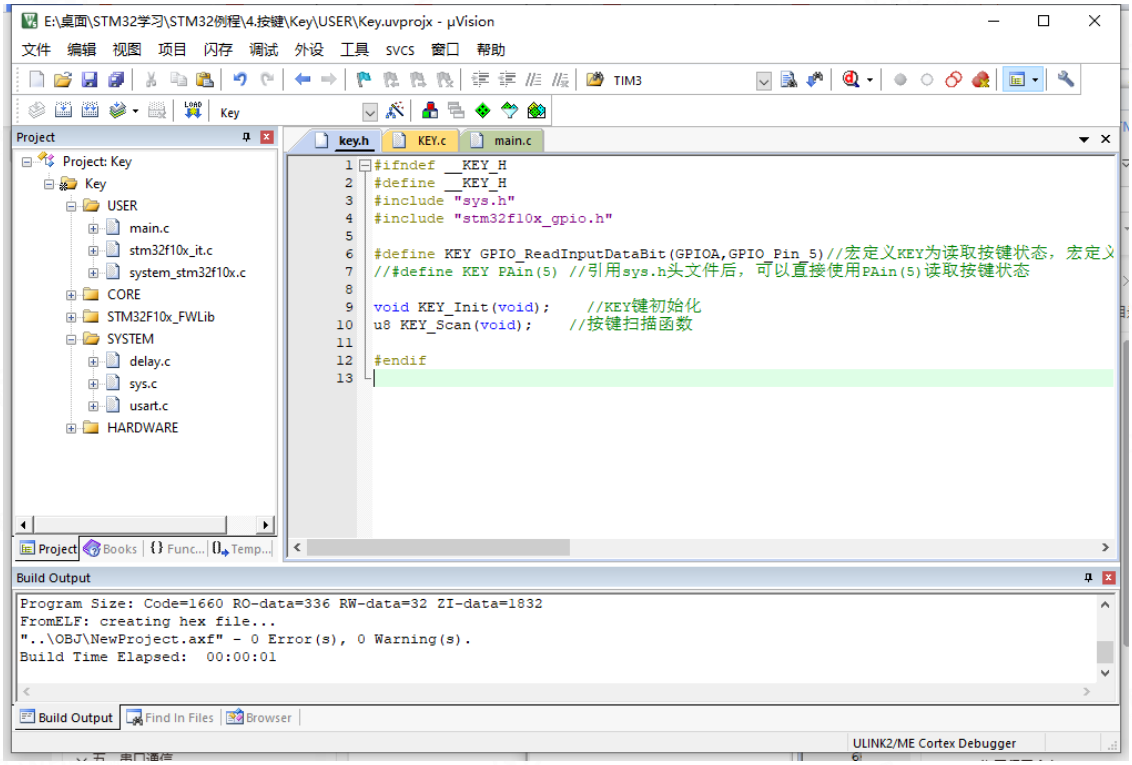

图 3-3-1 KEY.h

```
#ifndef __KEY_H
#define __KEY_H
#include "sys.h"
#include "stm32f10x_gpio.h"

#define KEY GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_5)//宏定义 KEY 为读取按键状态，
宏定义可以提高程序可读性
//#define KEY PAin(5) //引用 sys.h 头文件后，可以直接使用 PAin(5)读取按键状态

void KEY_Init(void);     //KEY 键初始化
u8 KEY_Scan(void);       //按键扫描函数

#endif
```

② KEY.c

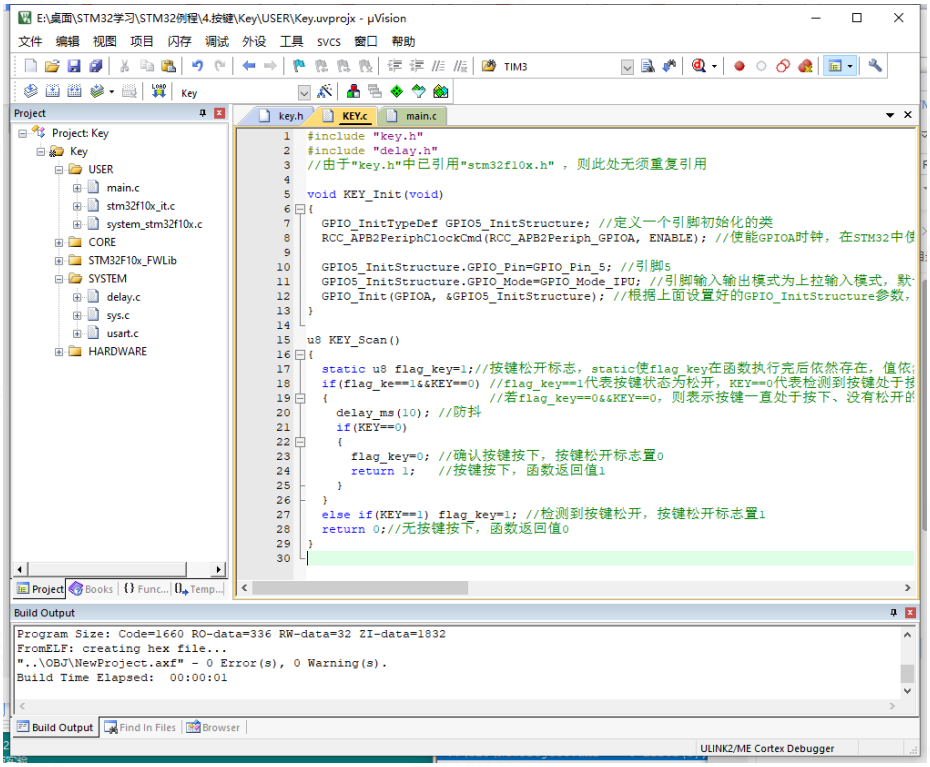

图 3-3-2 KEY.c

```
#include "key.h"
#include "delay.h"
//由于"key.h"中已引用"stm32f10x.h" ，则此处无须重复引用

void KEY_Init(void)
{
  GPIO_InitTypeDef GPIO5_InitStructure; //定义一个引脚初始化的类
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使能 GPIOA 时钟，在
STM32 中使用 IO 口前都要使能对应时钟

    GPIO5_InitStructure.GPIO_Pin=GPIO_Pin_5; //引脚 5
    GPIO5_InitStructure.GPIO_Mode=GPIO_Mode_IPU; //引脚输入输出模式为上拉输入模
式，默认为高电平，外接地后为低电平
    GPIO_Init(GPIOA, &GPIO5_InitStructure); //根据上面设置好的
GPIO_InitStructure 参数，初始化引脚 GPIOA_PIN4
}

u8 KEY_Scan()
{
```

```
    static u8 flag_key=1;//按键松开标志，static 使 flag_key 在函数执行完后依然存在，值依然不变
    if(flag_ke==1&&KEY==0) //flag_key==1 代表按键状态为松开，KEY==0 代表检测到按键处于按下状态，两者结合代表一次按下动作
    {                      //若 flag_key==0&&KEY==0，则表示按键一直处于按下、没有松开的状态
        delay_ms(10); //防抖
        if(KEY==0)
      {
            flag_key=0; //确认按键按下，按键松开标志置 0
        return 1;      //按键按下，函数返回值 1
        }
    }
  else if(KEY==1) flag_key=1; //检测到按键松开，按键松开标志置 1
    return 0;//无按键按下，函数返回值 0
}
```

3.4 编写主函数

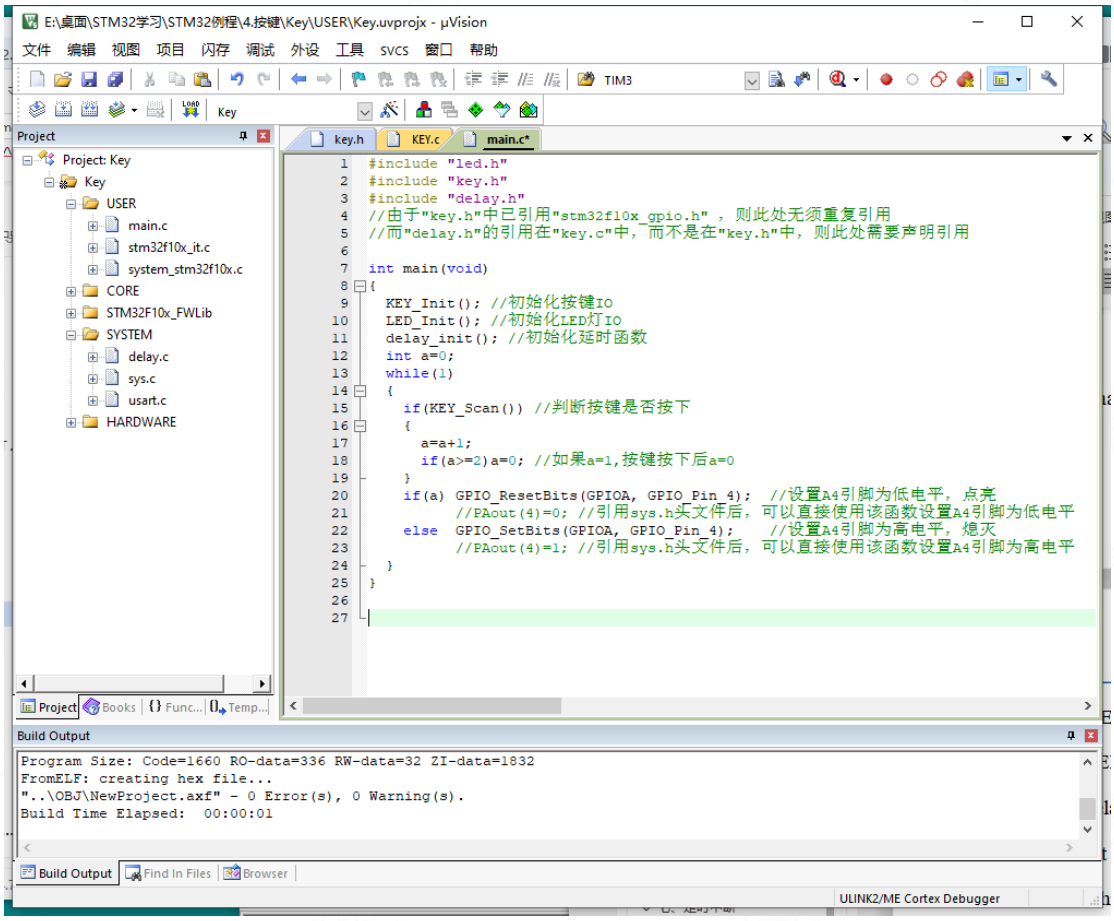

图 3-4-1 主函数

```
#include "led.h"
#include "key.h"
```

```
#include "delay.h"
//由于"key.h"中已引用"stm32f10x_gpio.h" ，则此处无须重复引用
//而"delay.h"的引用在"key.c"中，而不是在"key.h"中，则此处需要声明引用
int main(void)
{
    KEY_Init(); //初始化按键 IO
    LED_Init(); //初始化 LED 灯 IO
    delay_init(); //初始化延时函数
    int a=0;
  while(1)
    {
        if(KEY_Scan()) //判断按键是否按下
        {
            a=a+1;
            if(a>=2)a=0; //如果 a=1,按键按下后 a=0
        }
    if(a) GPIO_ResetBits(GPIOA, GPIO_Pin_4);  //设置 A4 引脚为低电平，点亮
              //PAout(4)=0; //引用 sys.h 头文件后，可以直接使用该函数设置 A4 引脚
为低电平
    else  GPIO_SetBits(GPIOA, GPIO_Pin_4);    //设置 A4 引脚为高电平，熄灭
              //PAout(4)=1; //引用 sys.h 头文件后，可以直接使用该函数设置 A4 引脚
为高电平
    }
}
```

3.5 实现思路及效果

(1) 创建按键库函数，包括按键初始化函数和按键检测函数。

(2) 在主函数中调用按键初始化和 LED 初始化函数。

(3) 在主函数的 while 循环中检测按键按下状态，若检测到按键按下，蓝色 LED 灯改变一次状态。

3.6 本节知识要点

- GPIO 输入模式的使用。

按键（GPIO 输入）初始化与 LED（GPIO 输出）初始化的区别在于 GPIO5_InitStructure.GPIO_Mode 的设置。

- 引脚下降沿状态的检测即按键按下检测的思路。

4. 串口通信

4.1 相关 IO 介绍

Forest S1 上的串口通信引脚有 PA9（UART1_TX）、PA10（UART1_RX）、PA2（UART2_TX）、PA3（UART2_RX）、B10（USART3_TX）、B11（USART3_RX）。

其中 PA9、PA10 已经封装成 USB 接口，可以直接与电脑通信。

本节我们以串口 1（UART1）进行讲解。

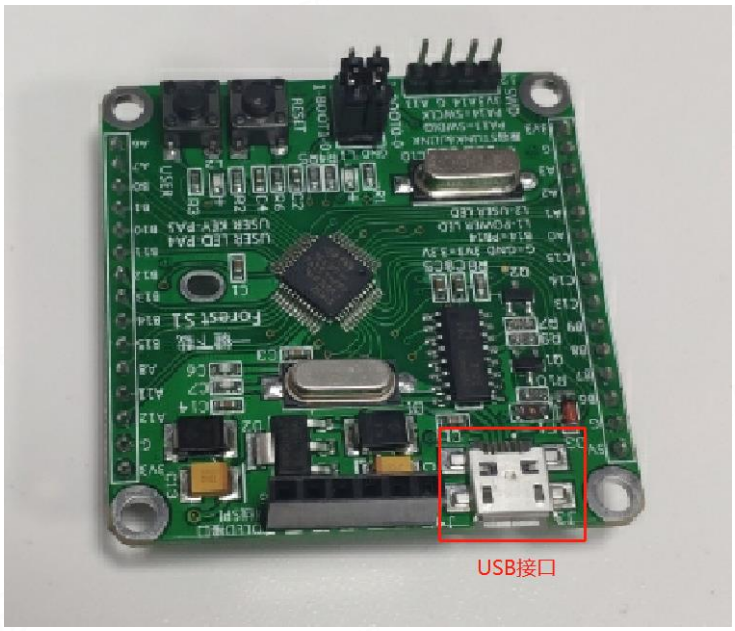

图 4-1-1 USB 接口

4.2 准备工作

我们直接复制工程模板，并重命名为【UART1】，则本节的串口通信工程创建完成。

打开工程【UART1】，移除【SYSTEM】下的【uart.c】。我们该节将自己手动编写一套串口通信的程序，而库文件【uart.c】内已经包含了一套串口通信的程序，两者会发生冲突。

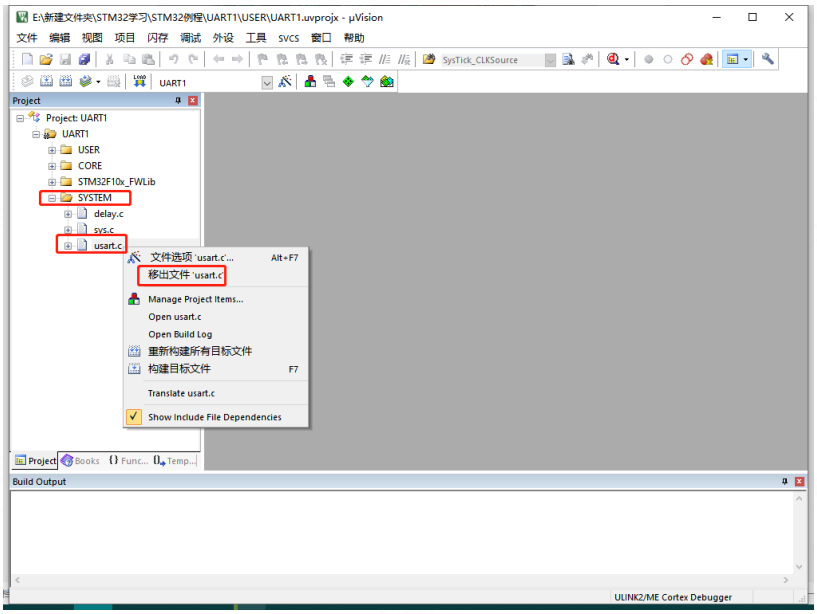

图 4-2-1 移除文件【usart.c】

4.3 编写串口通信 1 初始化函数

本节我们将直接在 main.c 中编写初始化函数，用户若感兴趣可自行编写为库函数形式。

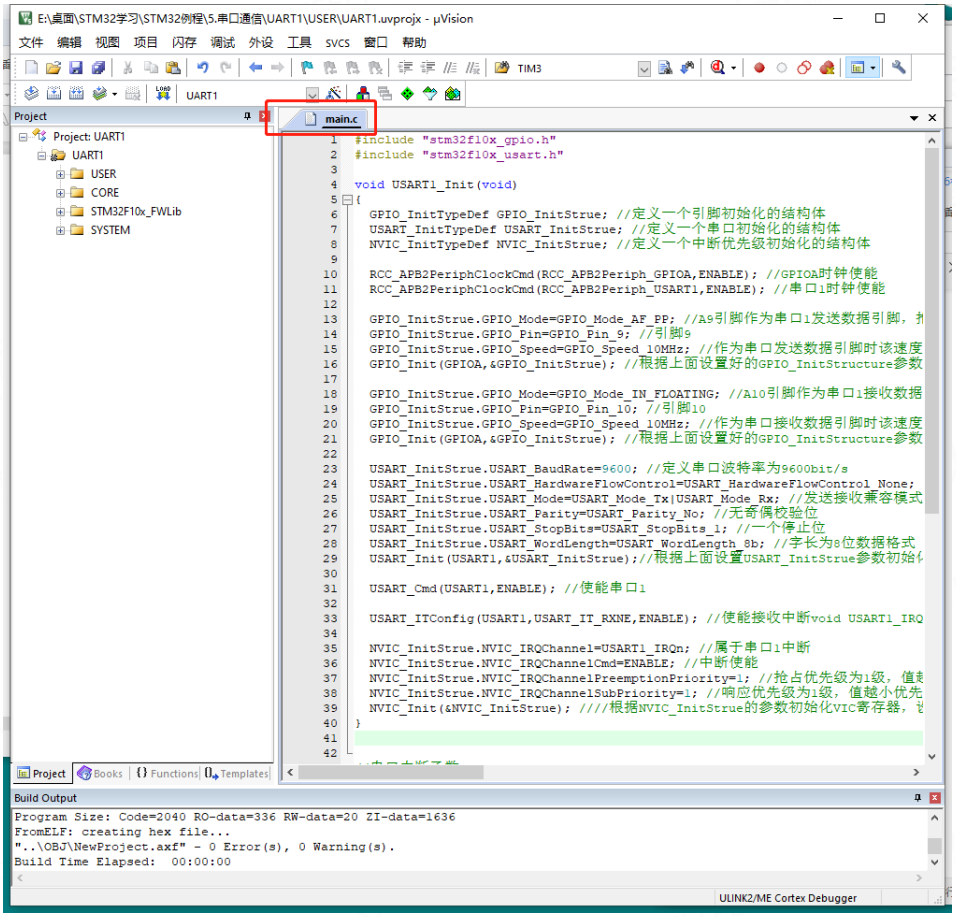

图 4-3-1 初始化函数

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_usart.h"

void USART1_Init(void)
{
```

1) 定义相关结构体：

```
    GPIO_InitTypeDef GPIO_InitStrue; //定义一个引脚初始化的结构体
    USART_InitTypeDef USART_InitStrue; //定义一个串口初始化的结构体
    NVIC_InitTypeDef NVIC_InitStrue; //定义一个中断优先级初始化的结构体
```

2) 使能相关时钟：

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE); //GPIOA 时钟使能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE); //串口 1 时钟使能
```

3) 串口相关 GPIO 设置，串口功能为 A9 引脚的复用功能，所以不能设置为普通的推挽输出，而是设置为推挽复用输出：

```
    GPIO_InitStrue.GPIO_Mode=GPIO_Mode_AF_PP; //A9 引脚作为串口 1 发送数据引脚,推挽复用输出
    GPIO_InitStrue.GPIO_Pin=GPIO_Pin_9; //引脚 9
    GPIO_InitStrue.GPIO_Speed=GPIO_Speed_10MHz; //作为串口发送数据引脚时该速度可以为任意
  GPIO_Init(GPIOA,&GPIO_InitStrue); //根据上面设置好的 GPIO_InitStructure 参数进行初始化

    GPIO_InitStrue.GPIO_Mode=GPIO_Mode_IN_FLOATING; //A10 引脚作为串口 1 接收数据引脚，浮空输入或带上拉输入
    GPIO_InitStrue.GPIO_Pin=GPIO_Pin_10; //引脚 10
    GPIO_InitStrue.GPIO_Speed=GPIO_Speed_10MHz; //作为串口接收数据引脚时该速度可以为任意
  GPIO_Init(GPIOA,&GPIO_InitStrue); //根据上面设置好的 GPIO_InitStructure 参数进行初始化
```

4) 串口功能设置：

```
    USART_InitStrue.USART_BaudRate=9600; //定义串口波特率为 9600bit/s
    USART_InitStrue.USART_HardwareFlowControl=USART_HardwareFlowControl_None; //无硬件数据流控制
    USART_InitStrue.USART_Mode=USART_Mode_Tx|USART_Mode_Rx; //发送接收兼容模式
    USART_InitStrue.USART_Parity=USART_Parity_No; //无奇偶校验位
    USART_InitStrue.USART_StopBits=USART_StopBits_1; //一个停止位
    USART_InitStrue.USART_WordLength=USART_WordLength_8b; //字长为 8 位数据格式
    USART_Init(USART1,&USART_InitStrue);//根据上面设置 USART_InitStrue 参数初始化串口 1
```

5) 使能串口功能：

```
    USART_Cmd(USART1,ENABLE); //使能串口 1
```

6) 使能串口 1 接收中断，定义串口 1 中断服务函数：

```
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
```

7) 如果使能了某个中断，则必须设置该中断对应中断服务函数的优先级：

```
    NVIC_InitStrue.NVIC_IRQChannel=USART1_IRQn; //属于串口 1 中断
    NVIC_InitStrue.NVIC_IRQChannelCmd=ENABLE; //中断使能
    NVIC_InitStrue.NVIC_IRQChannelPreemptionPriority=1;//抢占优先级为 1 级，值越小优先级越高，0 级优先级最高
    NVIC_InitStrue.NVIC_IRQChannelSubPriority=1; //响应优先级为 1 级，值越小优先级越高，0 级优先级最高
    NVIC_Init(&NVIC_InitStrue); ////根据 NVIC_InitStrue 的参数初始化 VIC 寄存器，设置串口 1 中断优先级
}
```

4.4 编写串口通信 1 中断函数

开启串口接收中断函数 void USART1_IRQHandler(void)，即每接收到一个字符即进入此函数一次。开启命令为 USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);，放在初始化函数中。

函数功能为将接收到的数据通过串口发送返回给【串口调试助手】。

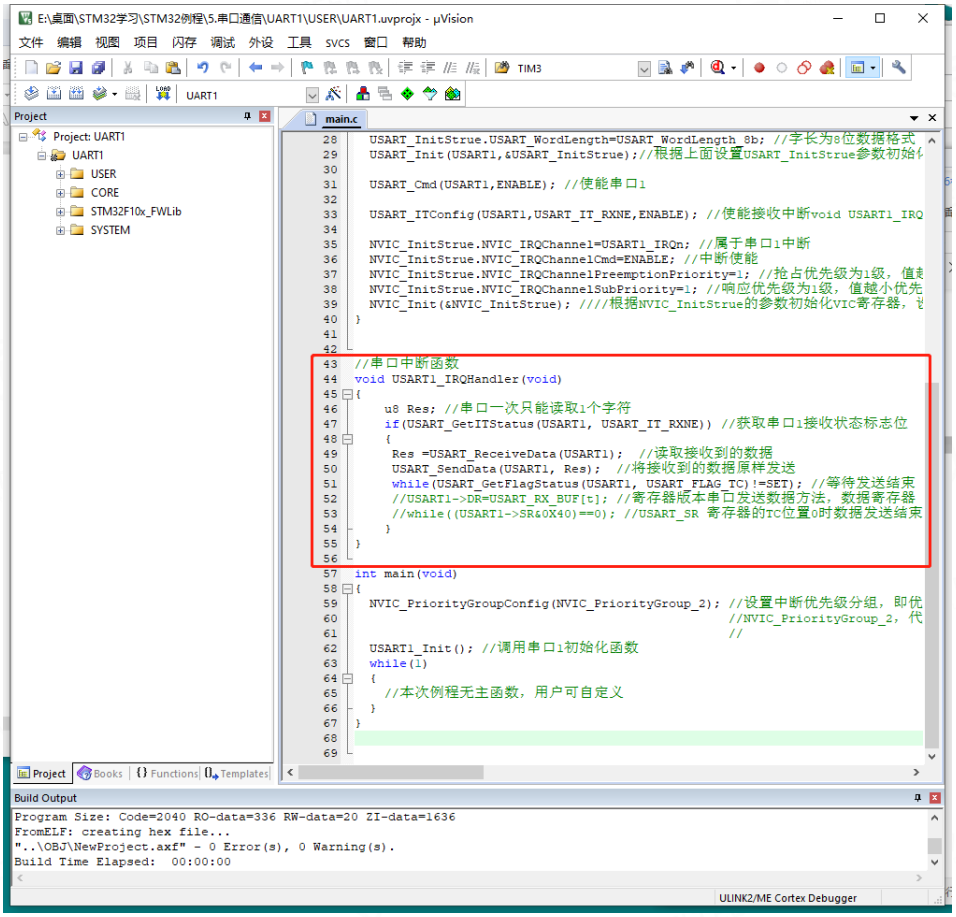

图 4-4-1 中断函数

```
//串口中断函数
void USART1_IRQHandler(void)
{
      u8 Res; //串口一次只能读取 1 个字符
        if(USART_GetITStatus(USART1, USART_IT_RXNE)) //获取串口 1 接收状态标志位，
以确定触发该中断是因为串口 1 接收到了新数据
        {
         Res =USART_ReceiveData(USART1);     //读取接收到的数据
         USART_SendData(USART1, Res);  //将接收到的数据原样发送
         while(USART_GetFlagStatus(USART1, USART_FLAG_TC)!=SET); //等待发送结束
       //USART1->DR=USART_RX_BUF[t]; //寄存器版本串口发送数据方法，数据寄存器
USART_DR 可以存放要发送的数据
         //while((USART1->SR&0X40)==0);//USART_SR 寄存器的 TC 位置 0 时数据发送结
束
    }
}
```

4.5 编写主函数

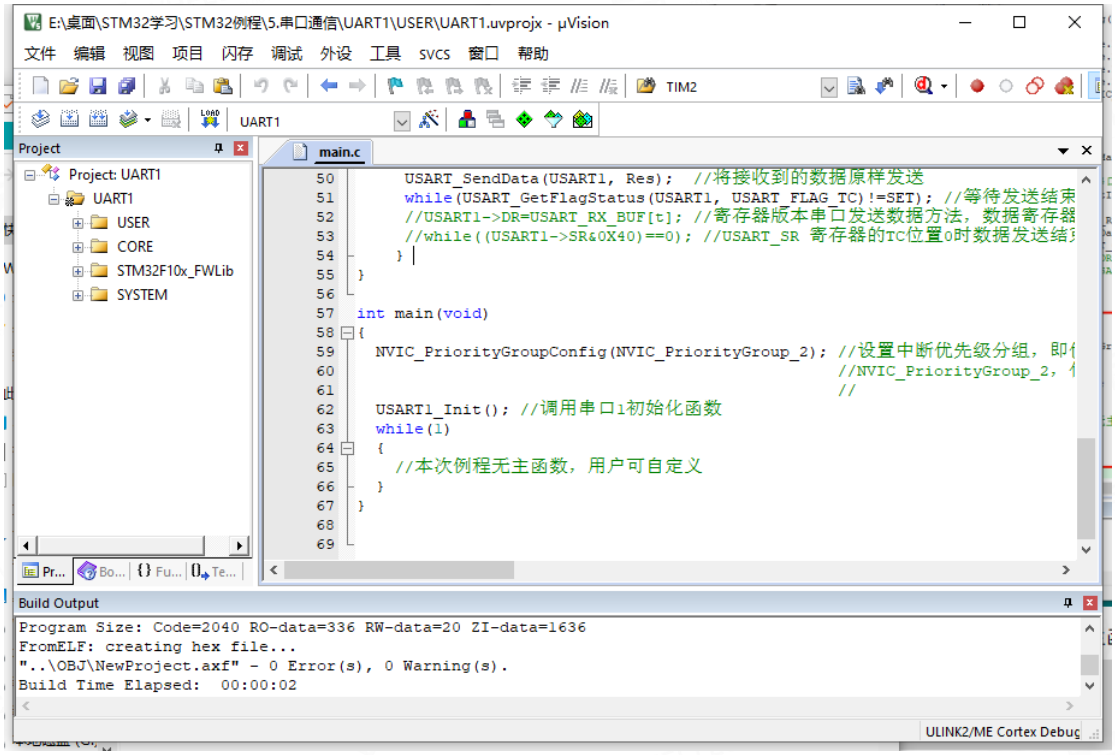

图 4-5-1 主函数

```
int main(void)
{
```

1) 中断优先级分组设置后，当有多个中断同时触发时，可以根据中断优先级分组设置确定优先处理哪个中断，本节只使用了一个串口 1 接收中断，用处不

大，但是当使用的中断变多后，中断优先级分组就很有必要了：

```
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优先级分组，即优先级分级个数，NVIC_PriorityGroup_2，代表抢占优先级位数位 2，可以分[0，1，2，3]四级优先级响应优先级位数位 2，可以分[0，1，2，3]四级优先级
    USART1_Init(); //调用串口 1 初始化函数
    while(1)
    {//本次例程无主函数，用户可自定义}
}
```

4.6 实现思路、串口调试助手软件的使用及程序效果

① 实现思路

(1) 创建串口初始化函数，在串口初始化函数中完成

a. CPIO 时钟使能、串口时钟使能，

b. GPIO 复用功能的初始化，

c. 串口通信参数的初始化及使能，

d. 中断优先级的初始化及使能。

(2) 创建串口中断服务函数，在串口中断服务函数中完成串口通信数据的接收和发送。

(3) 在主函数中设置中断优先级分组和调用串口初始化函数。

② 串口调试助手软件的使用

双击打开【串口调试助手[WHEELTEC].exe】。该软件可以在文件夹【软件资料】或者我们提供的例程文件夹中找到。

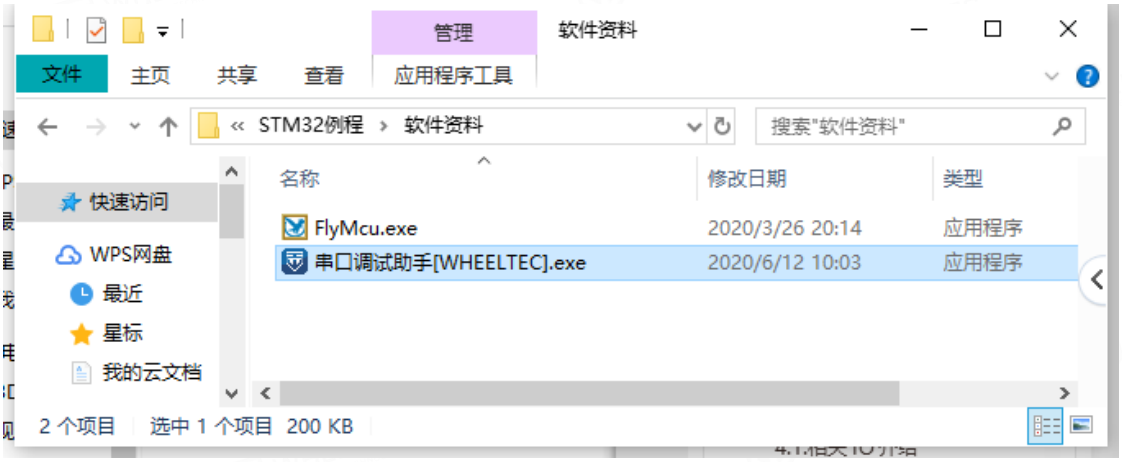

图 4-6-1 s 双击双击打开软件

在 1 处设置波特率等参数（对应串口初始化程序，如下图），在 2 处点击【打开串口】，当显示为【关闭串口】即代表串口已打开（此时无法上传程序，若要上传程序请点击【关闭串口】）。

```
USART_InitStrue.USART_BaudRate=9600; //定义串口波特率为9600bit/s
USART_InitStrue.USART_HardwareFlowControl=USART_HardwareFlowControl_None; //
USART_InitStrue.USART_Mode=USART_Mode_Tx|USART_Mode_Rx; //发送接收兼容模式
USART_InitStrue.USART_Parity=USART_Parity_No; //无奇偶校验位
USART_InitStrue.USART_StopBits=USART_StopBits_1; //一个停止位
USART_InitStrue.USART_WordLength=USART_WordLength_8b; //字长为8位数据格式
```

图 4-6-2 程序的串口通信设置

图 4-6-3 串口调试助手要设置的地方

点击 1 处【发送】，串口调试助手向 Forest S1 发送 2 处文字【WAITHOME】，在 3 处显示 Forest S1 发送过来的数据【WAITHOME】。

图 4-6-4 点击【发送】

③ 程序效果

程序效果即为串口调试助手向 Forest S1 发送什么数据，Forest S1 就向串口调试助手发送什么数据。

4.7 本节知识要点

- GPIO 的串口功能初始化过程，CPIO、串口时钟开启→GPIO 复用功能设置→串口通信参数的初始化及使能→中断优先级的初始化及使能。
- 串口接收发送数据功能的使用。
- 串口调试助手的使用。

5. 外部中断

5.1 相关 IO 介绍

STM32 的每个 IO 都可以作为外部中断输入。本节以 PA5 即按键引脚作为外部中断输入引脚为例进行讲解，控制效果依然是按键控制 LED 灯。

Forest S1 上的按键有 1 个，受引脚 PA5 控制，未按下时受上拉电阻作用，PA5 检测到高电平，按键按下时接地，PA5 检测到低电平。

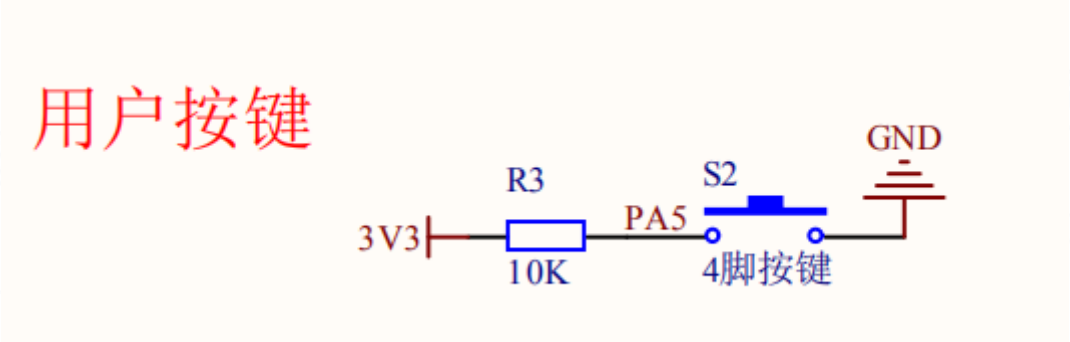

图 5-1-1 四脚按键引脚接线图

Forest S1 上可控制的 LED 灯有 1 个，为蓝色 LED 灯，受引脚 PA4 控制，低电平时点亮。

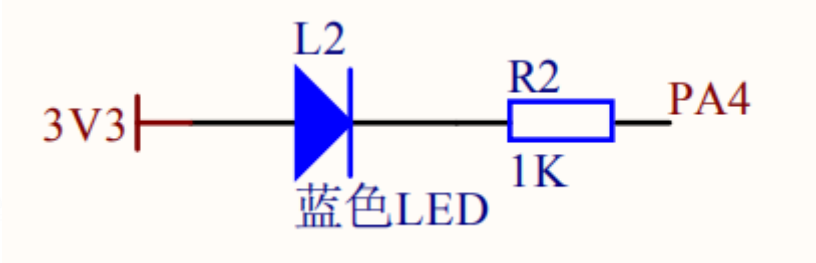

图 5-1-2 LED 引脚接线图

5.2 创建工程与外部中断库文件

我们将第四节的按键控制 LED 工程复制出来，并重命名为【EXTI】。创建库文件【EXTI.h】、【EXTI.c】，并加入到工程中。具体过程见第四节第 2 小节和第三节第 2、3 小节。

5.3 编写外部中断库函数

① EXTI.h

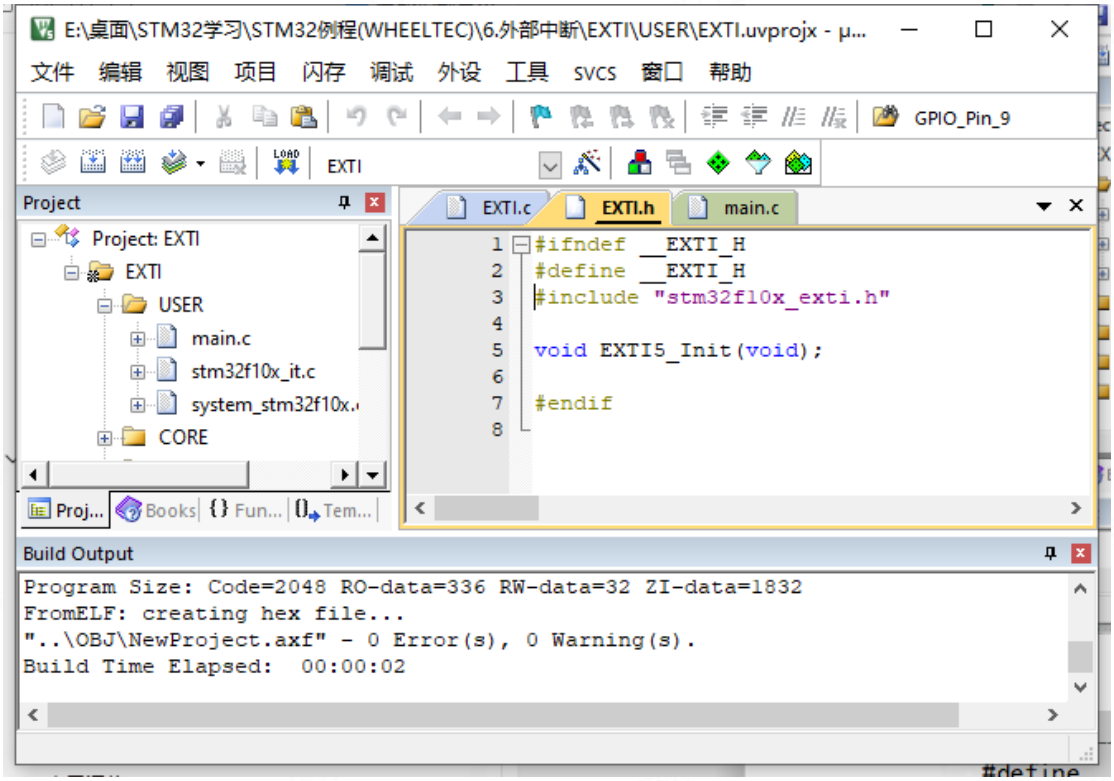

图 5-3-1 EXTI.h

```
#ifndef __EXTI_H
#define __EXTI_H
#include "stm32f10x_exti.h"

void EXTI5_Init(void);

#endif
```

② EXTI.c

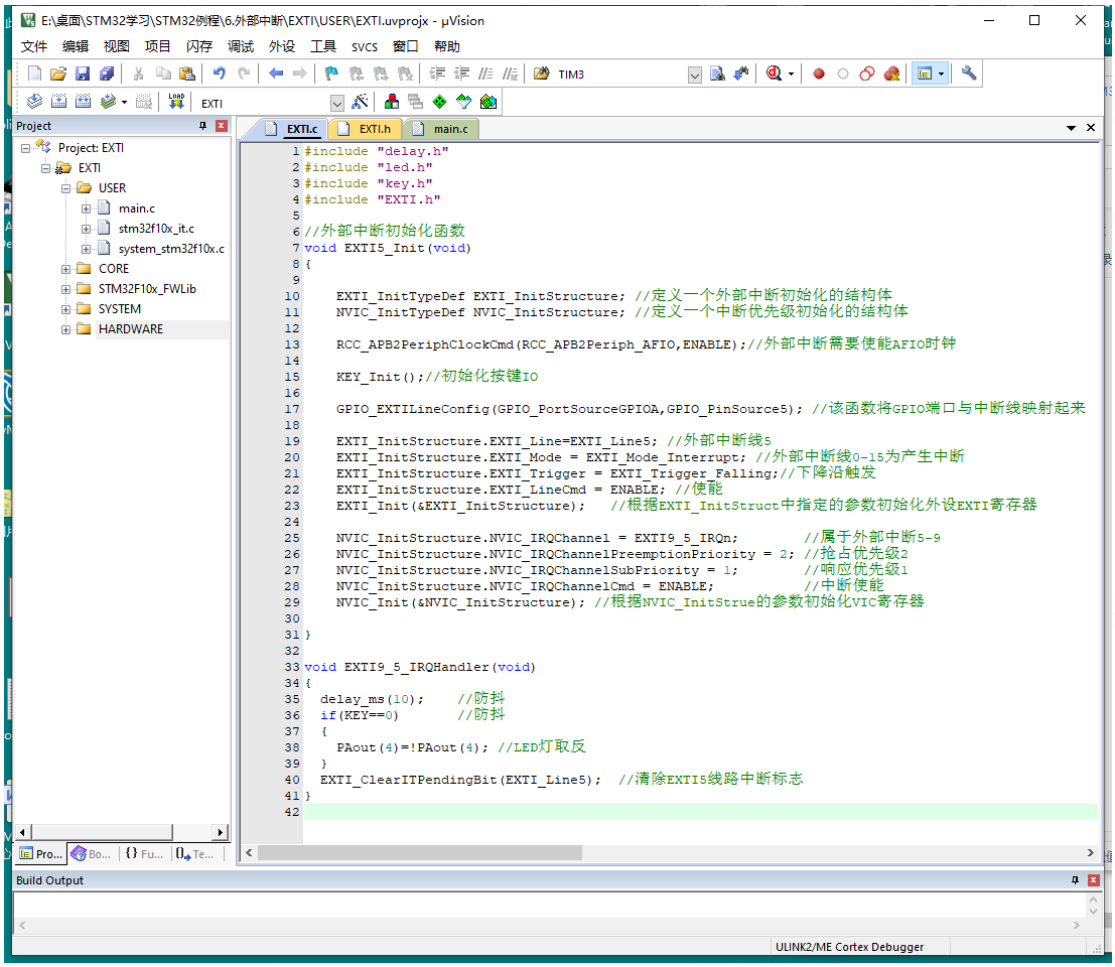

图 5-3-2 EXTI.c

```
#include "delay.h"
#include "led.h"
#include "key.h"
#include "EXTI.h"
//外部中断初始化函数
void EXTI5_Init(void)
{
```

1) 定义相关结构体：

```
    EXTI_InitTypeDef EXTI_InitStructure; //定义一个外部中断初始化的结构体
    NVIC_InitTypeDef NVIC_InitStructure; //定义一个中断优先级初始化的结构体
```

2) 使能外部中断时钟：

```
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//外部中断需要使能 AFIO 时钟
```

3) 初始化 GPIO：

```
    KEY_Init();//初始化按键 IO
```

4) 将 GPIO 端口与中断线映射起来，PA5 对应中断线 5：

```
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,GPIO_PinSource5);
```

5) 设置中断线 5 的属性：

```
    EXTI_InitStructure.EXTI_Line=EXTI_Line5; //外部中断线 5
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //外部中断线 0-15 为只能触发中断，而不能触发事件
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;//下降沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE; //使能
    EXTI_Init(&EXTI_InitStructure);      //根据 EXTI_InitStruct 中指定的参数初始化外设 EXTI 寄存器
```

6) 设置中断服务函数的优先级并使能，中断线 5-9 使用同一个中断服务函数：

```
        NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;              //属于外部中断 5-9
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;    //抢占优先级 2
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;                //响应优先级 1
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                      //中断使能
    NVIC_Init(&NVIC_InitStructure); //根据 NVIC_InitStrue 的参数初始化 VIC 寄存器
}
```

7) 中断线 5-9 的中断服务函数：

```
void EXTI9_5_IRQHandler(void)
{
  delay_ms(10);     //防抖
    if(KEY==0)        //防抖
    {      PAout(4)=!PAout(4); //LED 灯取反}
    EXTI_ClearITPendingBit(EXTI_Line5);  //清除 EXTI5 线路中断标志
}
```

5.4 编写主函数

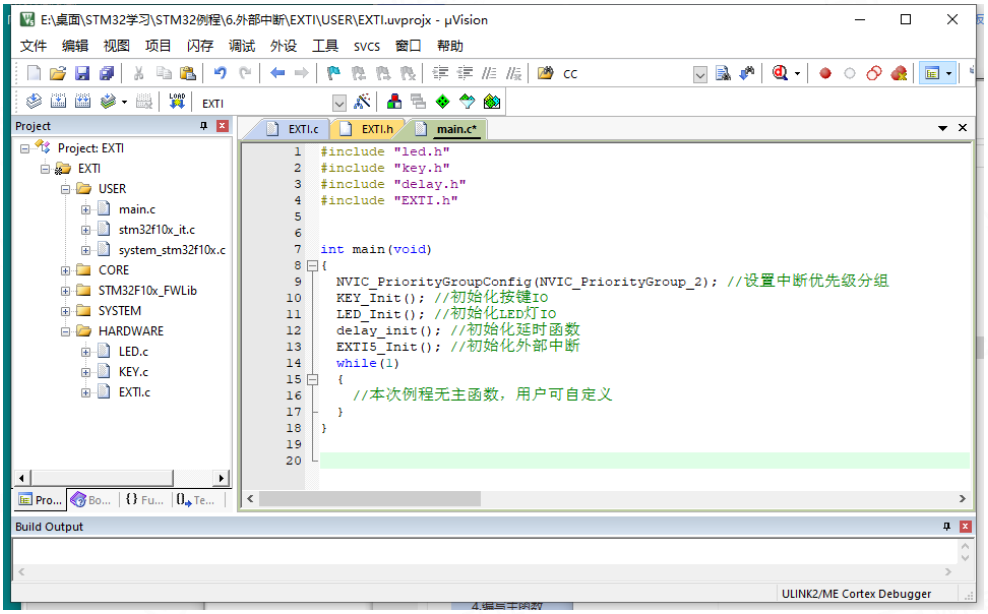

图 5-4-1 主函数

```
#include "led.h"
#include "key.h"
#include "delay.h"
#include "EXTI.h"
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优先级分组
    KEY_Init(); //初始化按键 IO
    LED_Init(); //初始化 LED 灯 IO
    delay_init(); //初始化延时函数
    EXTI5_Init(); //初始化外部中断
    while(1)
    {
      //本次例程无主函数，用户可自定义
    }
}
```

外部中断设置好后，按键按下即会触发中断服务函数并执行，单片机会暂停主循环 while 任务，优先执行中断服务函数，中断服务函数执行完毕后单片机再回到主循环 while 继续执行暂停的任务。

5.5 实现思路及效果

(1) 创建外部中断初始化函数，在外部中断初始化函数中完成

a. AFIO 时钟(外部中断时钟)使能，

b. 按键 GPIO 的初始化直接调用第 4 节编写的函数 KEY_Init()，

c. 完成 GPIO 端口与中断线的映射，

d. 外部中断参数的初始化，

e. 中断优先级的初始化及使能。

(2) 创建外部中断服务函数，在外部中断服务函数中按键按下的检测和 LED 灯状态取反操作。

(3) 在主函数中设置中断优先级分组和调用相关初始化初始化函数。

实现效果为按键按下一次，LED 灯状态改变一次。

5.6 本节知识要点

- 外部中断初始化过程，AFIO 时钟开启→相关 GPIO 初始化→完成 GPIO 端口与中断线的映射→外部中断的初始化→中断优先级初始化。
- 外部中断服务函数的使用。

6. 定时中断

6.1 相关 IO 介绍

本例程使用定时中断控制 LED 亮灭。

Forest S1 上可控制的 LED 灯有 1 个，为蓝色 LED 灯，受引脚 PA4 控制，低电平时点亮。

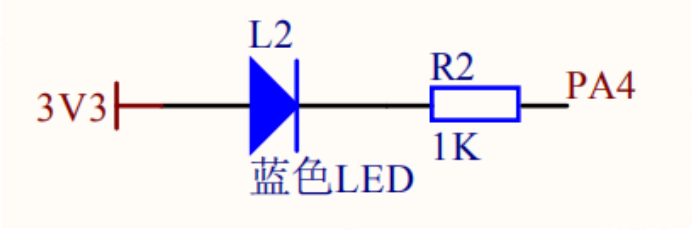

图 6-1-1 蓝色 LED 灯引脚接线图

6.2 创建工程与定时中断库文件

我们将第三节的点亮 LED 工程复制出来，并重命名为【TIMER】。创建库文件【TIMER.h】、【TIMER.c】，并加入到工程中。具体过程见第四节第 2 小节和第三节第 2、3 小节。

6.3 编写定时中断库函数

① TIMER.h

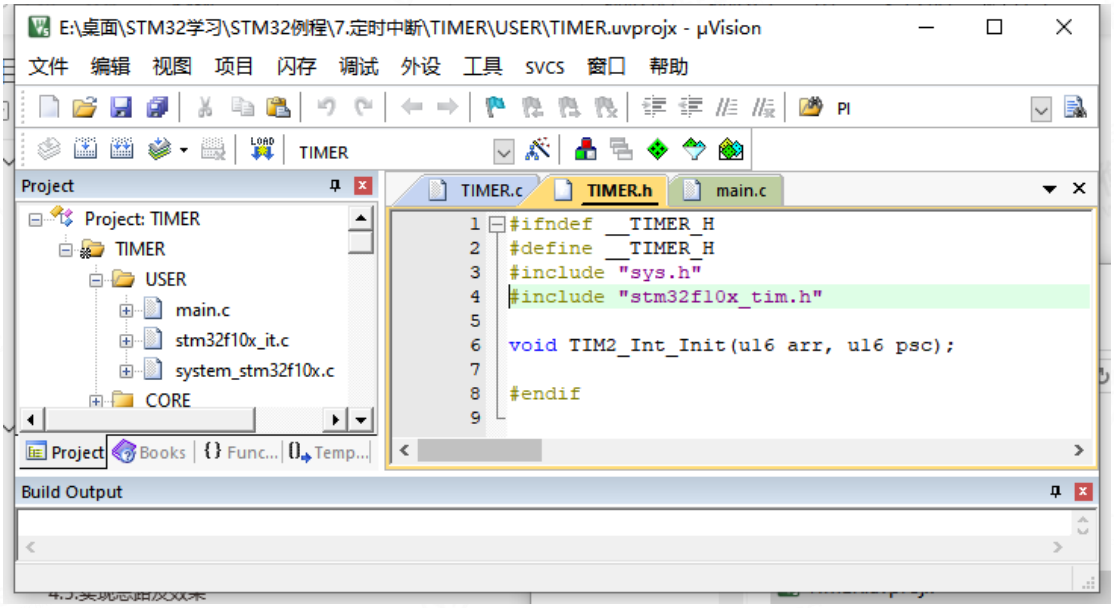

图 6-3-1 TIMER.h

```
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"
#include "stm32f10x_tim.h"

void TIM2_Int_Init(u16 arr, u16 psc);

#endif
```

② TIMER.c

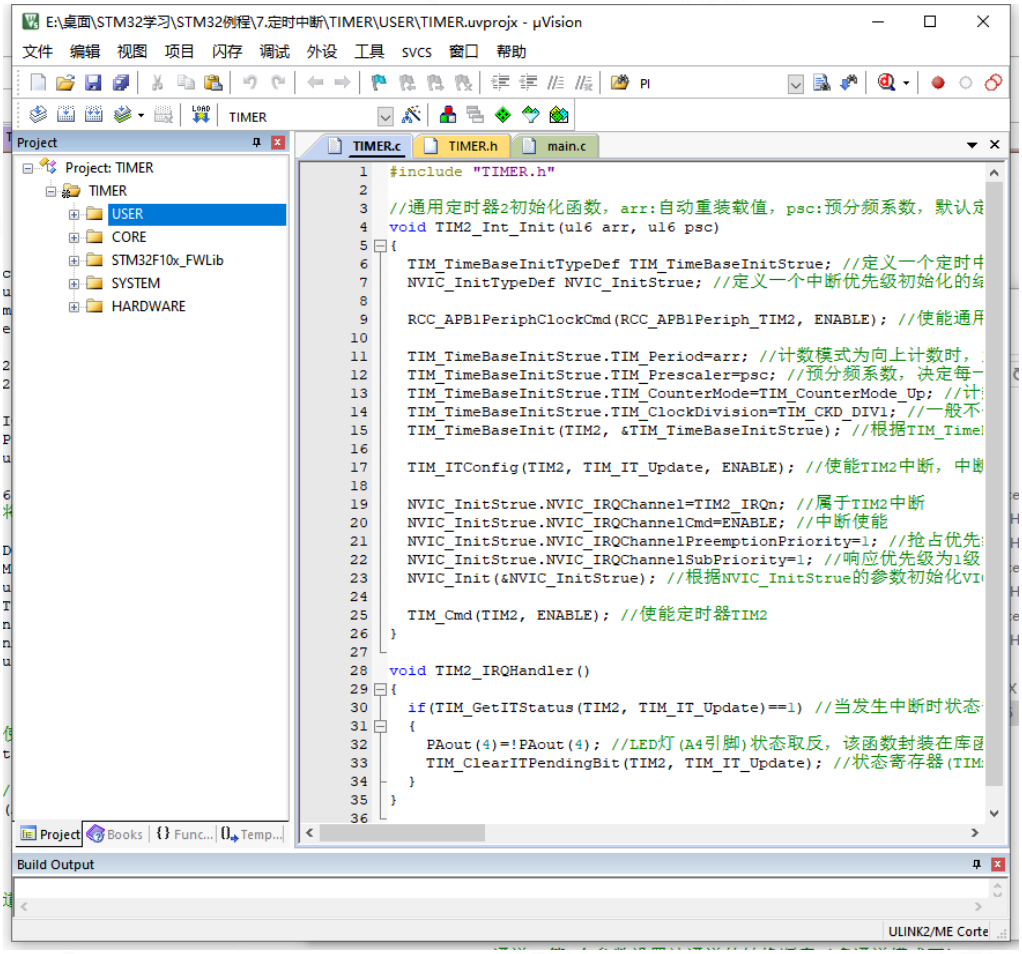

图 6-3-2 TIMER.c

```
#include "TIMER.h"
//通用定时器 2 初始化函数，arr:自动重装载值，psc:预分频系数，默认定时时钟为 72MHZ
时，两者共同决定定时中断时间
void TIM2_Int_Init(u16 arr, u16 psc)
{
```

1) 定义相关结构体：

```
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseInitStrue; //定义一个定时器初始化的结
构体
    NVIC_InitTypeDef NVIC_InitStrue; //定义一个中断优先级初始化的结构体
```

2) 使能定时器时钟：

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); //使能通用定时器 2 时钟
```

3) 设置并初始化定时器 TIM2：

```
    TIM_TimeBaseInitStrue.TIM_Period=arr; //计数模式为向上计数时，定时器从 0 开始计数，计数超过到 arr 时触发定时中断服务函数
    TIM_TimeBaseInitStrue.TIM_Prescaler=psc; //预分频系数，决定每一个计数的时长
    TIM_TimeBaseInitStrue.TIM_CounterMode=TIM_CounterMode_Up; //计数模式：向上计数
    TIM_TimeBaseInitStrue.TIM_ClockDivision=TIM_CKD_DIV1; //一般不使用，默认 TIM_CKD_DIV1
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStrue); //根据 TIM_TimeBaseInitStrue 的参数初始化定时器 TIM2
```

4) 使能定时器中断：

```
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); //使能 TIM2 中断，中断模式为更新中断：TIM_IT_Update
```

5) 定时器中断优先级设置：

```
    NVIC_InitStrue.NVIC_IRQChannel=TIM2_IRQn; //属于 TIM2 中断
    NVIC_InitStrue.NVIC_IRQChannelCmd=ENABLE; //中断使能
    NVIC_InitStrue.NVIC_IRQChannelPreemptionPriority=1; //抢占优先级为 1 级，值越小优先级越高，0 级优先级最高
    NVIC_InitStrue.NVIC_IRQChannelSubPriority=1; //响应优先级为 1 级，值越小优先级越高，0 级优先级最高
    NVIC_Init(&NVIC_InitStrue); //根据 NVIC_InitStrue 的参数初始化 VIC 寄存器，设置 TIM2 中断
```

6) 使能定时器：

```
    TIM_Cmd(TIM2, ENABLE); //使能定时器 TIM2
}
```

7) 定时器中断服务函数

```
void TIM2_IRQHandler()
{
  if(TIM_GetITStatus(TIM2, TIM_IT_Update)==1) //当发生中断时状态寄存器(TIMx_SR)的 bit0 会被硬件置 1，即定时器 2 的中断标志置 1
    {
      PAout(4)=!PAout(4); //LED 灯(A4 引脚)状态取反，该函数封装在库函数"sys.h"中
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update); //状态寄存器(TIMx_SR)的 bit0 置 0，即定时器 2 的中断标志置 0
    }
}
```

6.4 编写主函数

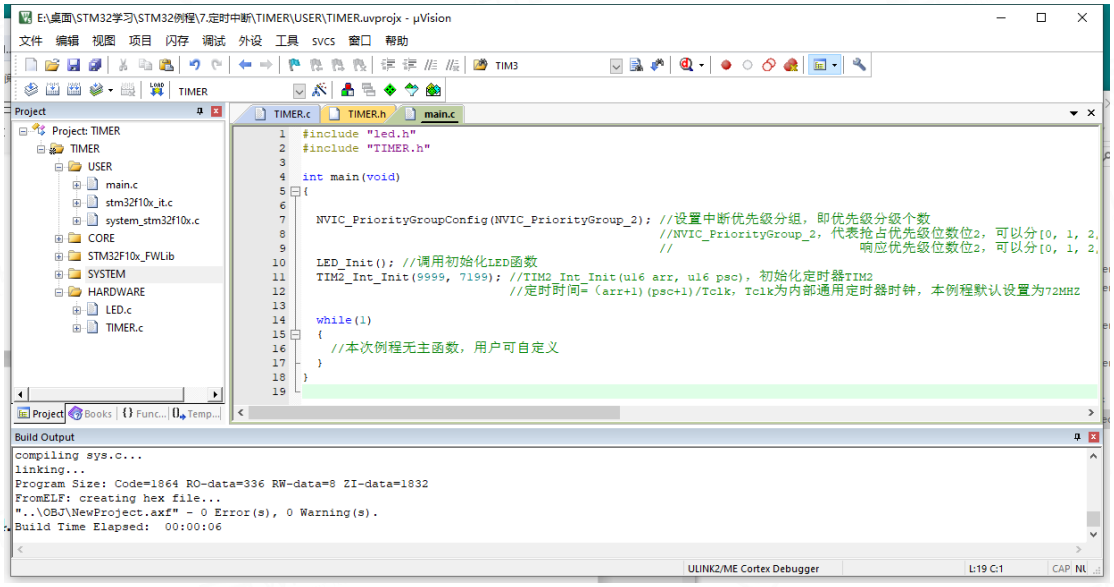

图 6-4-1 主函数

```
#include "led.h"
#include "TIMER.h"

int main(void)
{
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置中断优先级分组，即优先级分级个数
                                                 //NVIC_PriorityGroup_2,代表抢占优先级位数位 2，可以分[0，1，2，3]四级优先级
                                                 //响应优先级位数位 2，可以分[0，1，2，3]四级优先级
    LED_Init();//调用初始化 LED 函数，因为我们在定时器中断服务函数中进行了 LED 控制
  TIM2_Int_Init(9999,7199);//TIM2_Int_Init(u16 arr,u16 psc),初始化定时器 TIM2，定时时间=（arr+1)(psc+1)/Tclk，Tclk 为内部通用定时器时钟，本例程默认设置为 72MHZ

  while(1)
    {
        //本次例程无主函数，用户可自定义
    }
}
```

定时中断设置好后，定时时间满后，会触发对应定时器中断服务函数，单片机会暂停主循环 while 任务，优先执行对应定时器定时中断服务函数，执行完毕后单片机再回到主循环 while 继续执行暂停的任务。

6.5 实现思路及效果

(1) 创建定时器初始化函数，在定时器初始化函数中完成

a. 定时器时钟使能，

b. 定时器参数的初始化及定时中断的使能，

c. 中断优先级的初始化及使能，

d. 使能定时器。

(2) 创建定时中断服务函数，在定时中断服务函数中完成 LED 灯状态取反操作。

(3) 在主函数中设置中断优先级分组和调用相关初始化初始化函数。

实现效果为每隔 1 秒，LED 灯状态改变一次。

6.6 本节知识要点

- 定时中断初始化过程，定时器时钟开启→定时器的初始化→中断优先级初始化。
- 定时中断服务函数的使用。

7. 定时器的 PWM 输出模式

7.1 相关 IO 介绍

本例程使用定时器 PWM 输出控制 LED 灯的亮度。STM32 的定时器除了 TIM6 和 7，其他的定时器都可以用来产生 PWM 输出。

我们以 Forest S1 上的 A0(TIM2_CHN1)引脚为例做 PWM 输出例程。用户需自行 DIY 外接 LED 灯，LED 灯正极连接 A0，LED 灯负极连接 GND。

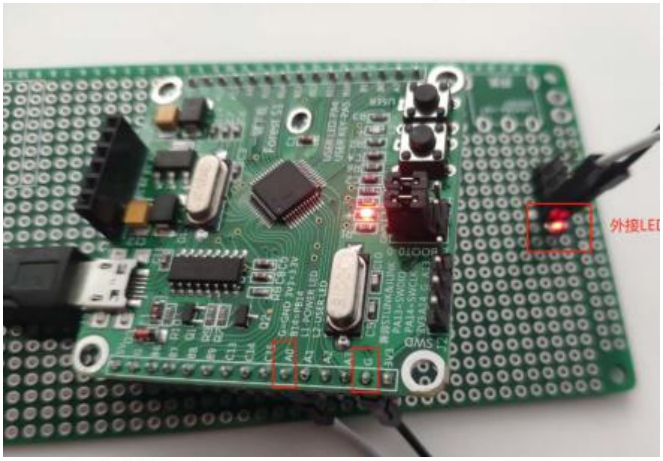

图 7-1-1 外接 LED 灯

7.2 编写 PWM 输出库函数

在上节定时中断工程的基础上添加库函数。

① TIMER.h

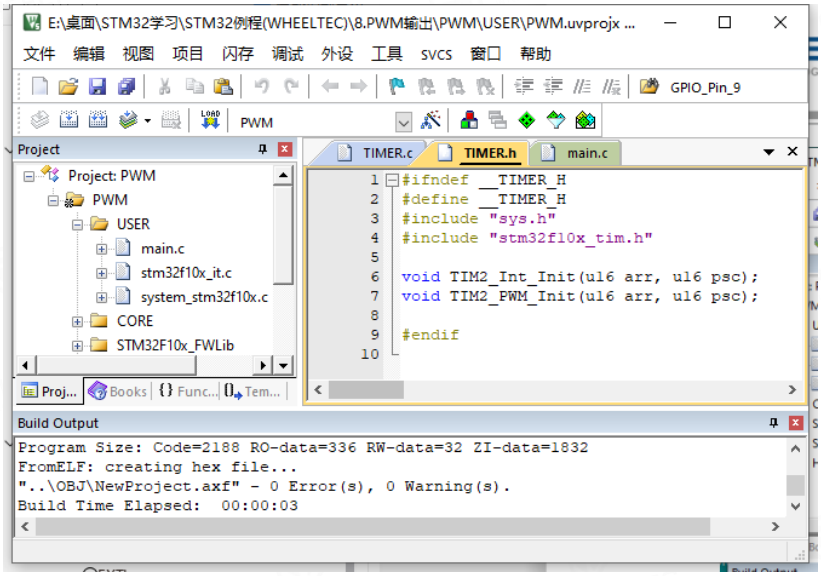

图 7-2-1 TIMER.h

```
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"
#include "stm32f10x_tim.h"

void TIM2_Int_Init(u16 arr, u16 psc);
void TIM2_PWM_Init(u16 arr, u16 psc);

#endif
```

② TIMER.c

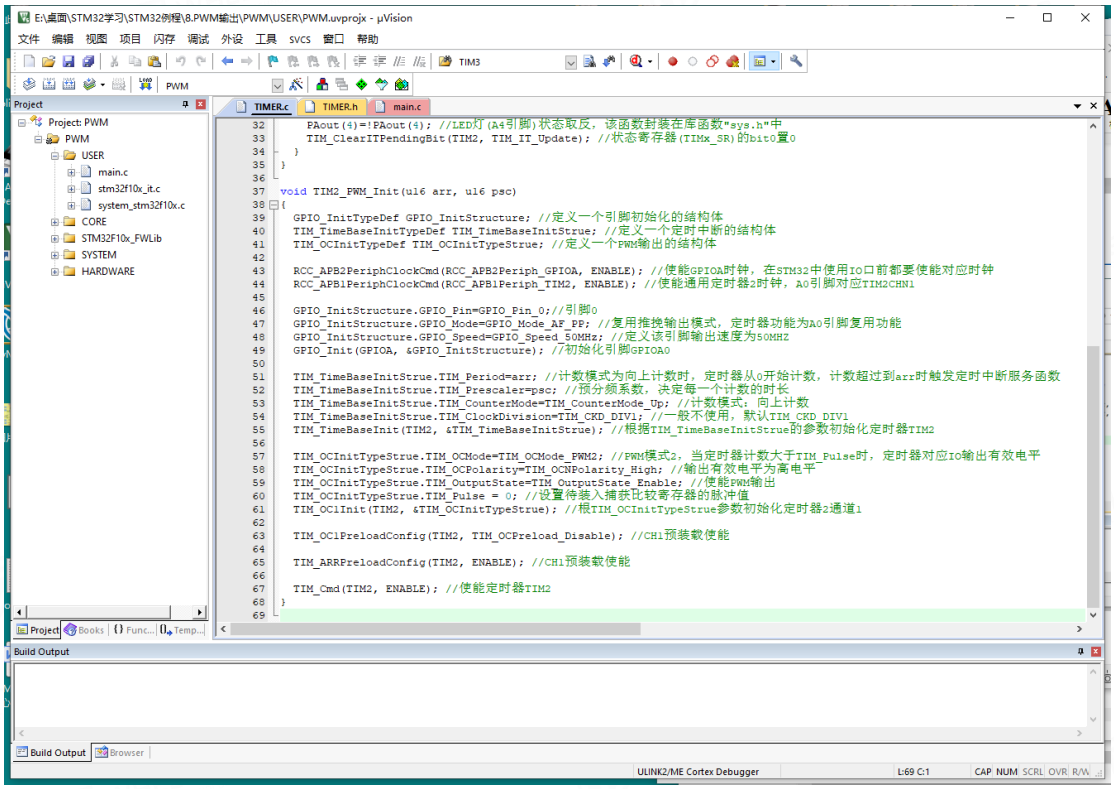

图 7-2-2 TIMER.c

```
void TIM2_PWM_Init(u16 arr, u16 psc)
{
```

1) 定义相关结构体：

```
    GPIO_InitTypeDef GPIO_InitStructure;     //定义一个引脚初始化的结构体
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStrue;     //定义一个定时中断的结构体
    TIM_OCInitTypeDef TIM_OCInitTypeStrue;    //定义一个 PWM 输出的结构体
```

2) 使能相关时钟：

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使能 GPIOA 时钟，在 STM32 中使用 IO 口前都要使能对应时钟
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); //使能通用定时器 2 时钟，A0 引脚对应 TIM2CHN1
```

3) 初始化 GPIO：

```
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_0;//引脚 0
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;//复用推挽输出模式，定时器功能为 A0 引脚复用功能
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;  //定义该引脚输出速度为 50MHZ
        GPIO_Init(GPIOA, &GPIO_InitStructure); //初始化引脚 GPIOA0
```

4) 设置并初始化定时器 TIM2：

```
    TIM_TimeBaseInitStrue.TIM_Period=arr;//计数模式为向上计数时，定时器从 0 开始计数，计数超过到 arr 时触发定时中断服务函数
    TIM_TimeBaseInitStrue.TIM_Prescaler=psc;   //预分频系数，决定每一个计数的时长
    TIM_TimeBaseInitStrue.TIM_CounterMode=TIM_CounterMode_Up;  //计数模式：向上计数
    TIM_TimeBaseInitStrue.TIM_ClockDivision=TIM_CKD_DIV1; //一般不使用，默认 TIM_CKD_DIV1
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStrue); //根据 TIM_TimeBaseInitStrue 的参数初始化定时器 TIM2
```

5) 设置并初始化 PWM：

```
    TIM_OCInitTypeStrue.TIM_OCMode=TIM_OCMode_PWM2; //PWM 模式 2，当定时器计数大于 TIM_Pulse 时，定时器对应 IO 输出有效电平
    TIM_OCInitTypeStrue.TIM_OCPolarity=TIM_OCNPolarity_High; //PWM 输出有效电平为高电平
    TIM_OCInitTypeStrue.TIM_OutputState=TIM_OutputState_Enable; //使能 PWM 输出
    TIM_OCInitTypeStrue.TIM_Pulse = 0; //设置待装入捕获比较寄存器的脉冲值
    TIM_OC1Init(TIM2, &TIM_OCInitTypeStrue); //根 TIM_OCInitTypeStrue 参数初始化定时器 2 通道 1
```

6) 预装载设置：

```
    TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable); //OC1 预装载使能，使能后改变 TIM_Pulse 的值立刻生效，不使能则下个周期生效
    TIM_ARRPreloadConfig(TIM2, ENABLE); //ARRP 预装载使能，改变 TIM_Period 的值立刻生效，不使能则下个周期生效
```

7) 使能定时器 2：

```
    TIM_Cmd(TIM2, ENABLE); //使能定时器 TIM2
}
```

7.3 编写主函数

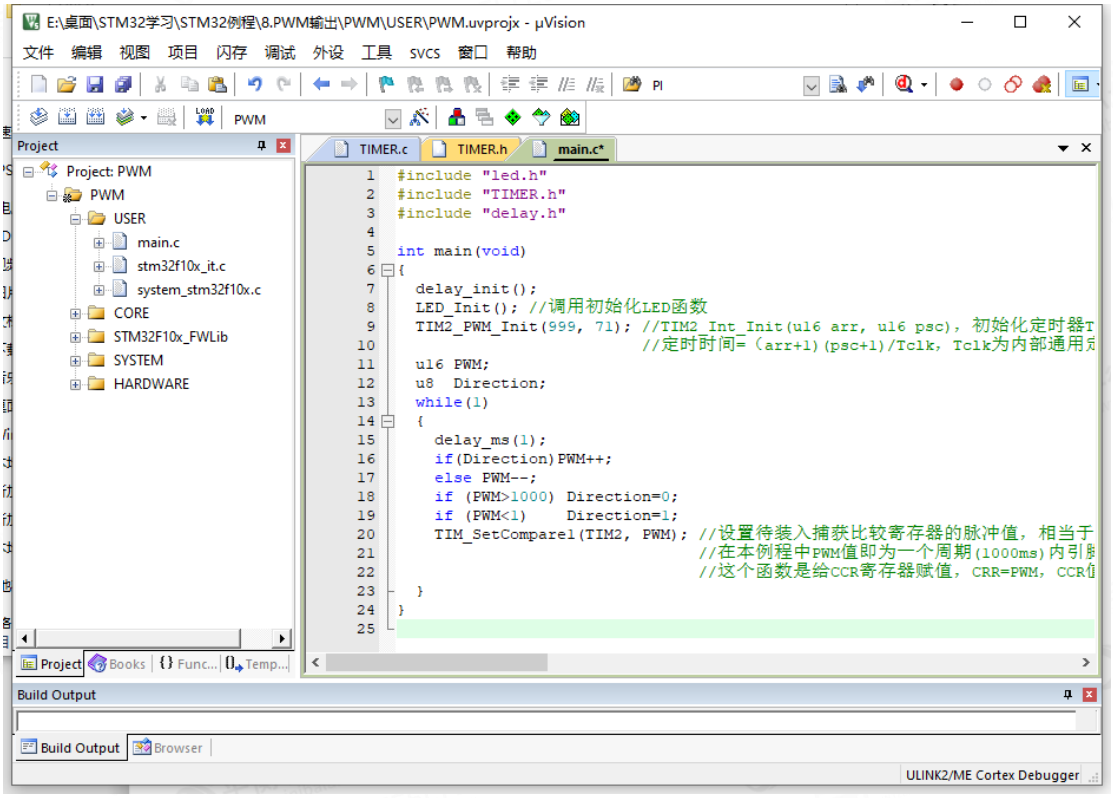

图 7-3-1 主函数

```
#include "led.h"
#include "TIMER.h"
#include "delay.h"
int main(void)
{
```

1) 调用相关初始化函数：

```
    delay_init();
    LED_Init(); //调用初始化 LED 函数
  TIM2_PWM_Init(999, 71); //TIM2_Int_Init(u16 arr, u16 psc)，初始化定时器 TIM2。
定时时间=（arr+1)(psc+1)/Tclk，Tclk 为内部通用定时器时钟，本例程默认设置为 72MHZ
    u16 PWM;//用于设置 PWM 输出值
    u8  Direction;//用于设置 PWM 值变化方向
```

2) 主循环，PWM 值循环变大变小，实现效果为 LED 循环变暗变亮：

```
  while(1)
    {
        delay_ms(1);
        if(Direction)PWM++;
        else PWM--;
        if (PWM>1000) Direction=0;
        if (PWM<1)    Direction=1;
```

```
        TIM_SetCompare1(TIM2, PWM); //设置待装入捕获比较寄存器的脉冲值，相当于不
断设置 TIM_Pulse，在本例程中 PWM 值即为一个周期(1000ms)内引脚 A0 输出的低电平时长
(单位 ms)。这个函数是给 CCR 寄存器赋值，CRR=PWM，CCR 值/ARR 值为低电平在一个周期内
的占比
    }   }
```

7.4 实现思路及效果

(1) 创建定时器初始化函数，在定时器初始化函数中完成

a. 定时器时钟及 GPIOA 时钟使能，

b. 定时器参数的初始化，

c. 定时器 PWM 输出参数的初始化及使能，

d. 使能定时器。

(2) 在主函数中调用相关初始化初始化函数。

(3) 在 while 循环中不断设置 PWM 值以调节 LED 灯亮度。

实现效果 LED 灯循环变亮和变暗。

7.5 本节知识要点

定时器 PWM 输出的初始化过程，定时器时钟及相关 GPIO 时钟开启→定时器的初始化→PWM 输出初始化。

小知识：同一定时器，不同 PWM 通道，定时器只需初始化一次，改定时器所有 PWM 通道使能后都可使用（记得初始化对应引脚），不同通道可以设置不同 PWM 值。

注意：本节没有使用定时中断功能。

8. PWM 输出控制舵机

8.1 相关 IO 介绍

本节我们以上一级 PWM 输出工程为基础继续编写程序。

接线：Forest S1 上的 A0(TIM2_CHN1)引脚连接舵机的控制端（一般为橙色线），舵机电源线（一般为红色线）连接 5V 接口，舵机地线（一般为棕色线）连接 GND 接口。

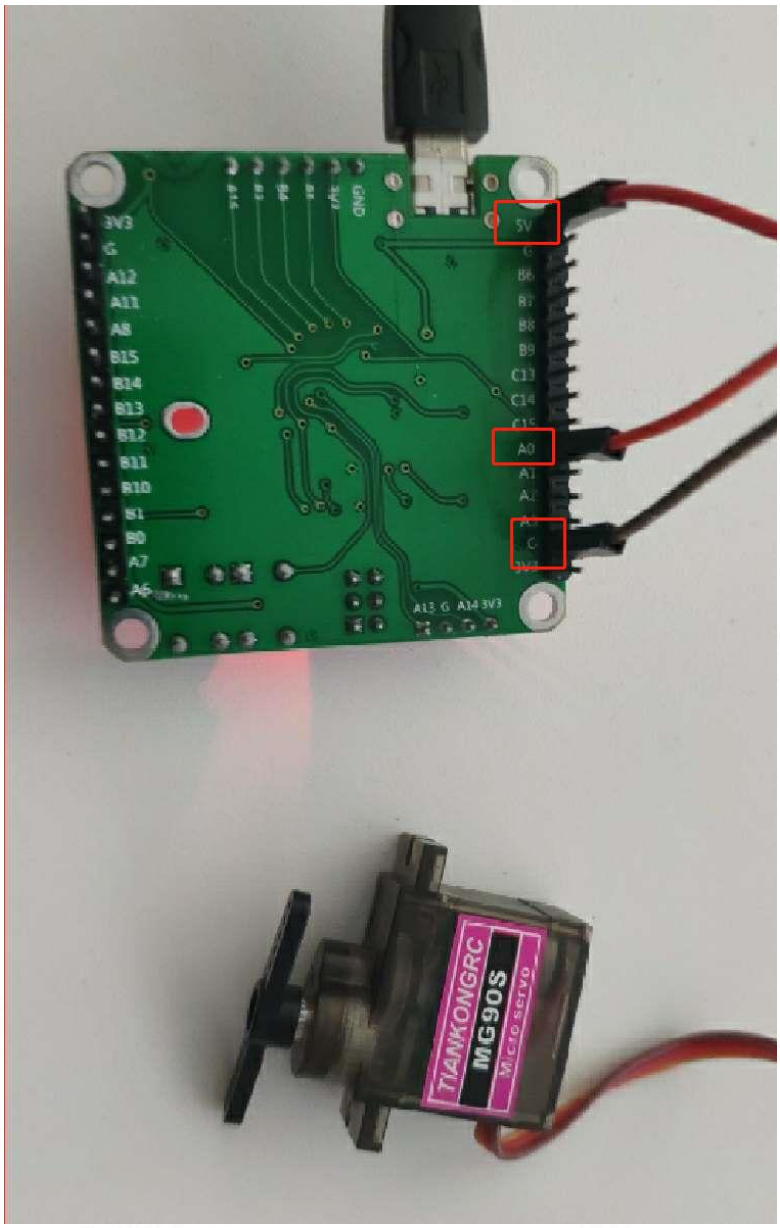

图 8-1-1 舵机与 STM32 接线

8.2 修改 PWM 库函数

修改 TIMER.c 文件里的定时器 2PWM 模式初始化函数，设置 PWM 模式为 PWM 模式 1，即当定时器计数小于 TIM_Pulse 时，定时器对应 IO 输出有效电平（本程序中有效电平设置为高电平）。

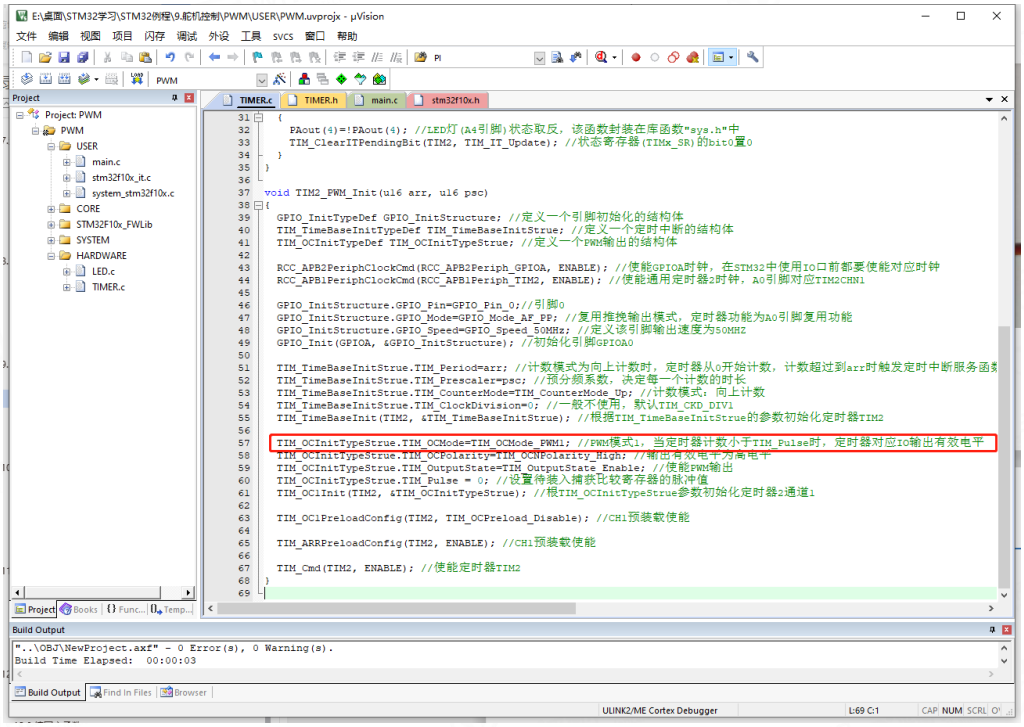

图 8-2-1 程序修改

8.3 编写主函数

PWM 控制方面我们直接使用上一节的编写函数

void TIM2_PWM_Init(u16 arr, u16 psc)。

① 舵机控制原理

舵机控制的原理是向舵机控制端输入指定 PWM 值，舵机则固定在对应 PWM 值的角度。一般舵机的控制信号 PWM 为 50HZ，高电平 0.5~2.5ms(占空比 2.5%~12.5%)对应舵机 0~180°，设高电平时间为 t，那么对应的角度

$$Angle = \frac{t-0.5}{2.5-0.5} \times 180°,$$

即高电平时间 t=0.5ms 对应 0°，

即高电平时间 t=1.0ms 对应 45°，

即高电平时间 t=1.5ms 对应 90°，

即高电平时间 t=2.0ms 对应 135°，

即高电平时间 t=2.5ms 对应 180°。

② 定时器 PWM 输出参数的设置

void TIM2_PWM_Init(u16 arr, u16 psc)。

这里已知主频 72M，要获得 50HZ 的 PWM 波频率，则有

$50=\frac{72\times10^6}{(arr+1)\times(psc+1)}$。

为方便使用，先定满占空比(arr 值—溢出系数)，假定 arr=9999，则有

$psc=\frac{72\times10^6}{50\times(9999+1)}-1=143$，每一计数代表的时间 $t_1=\frac{143+1}{72\times10^6}=2\times10^{-3}ms$。

TIM_SetCompare1(TIM2, PWM)这个函数是给 CCR 寄存器赋值，CRR=PWM。

一个 PWM 周期内的高电平时间 $t=t_1\times PWM$，则

$Angle=\frac{t-0.5}{2.5-0.5}\times180°=\frac{PWM\times2\times10^{-3}-0.5}{2.5-0.5}\times180°$。

$PWM=\frac{Angle\times(2.5-0.5)}{180°\times2\times10^{-3}}+\frac{0.5}{2\times10^{-3}}=\frac{Angle\times2}{180°\times2\times10^{-3}}+250$，Angle 取值范围 0 度到 180 度，则 PWM 值范围为[250, 1250]。

③ 主函数

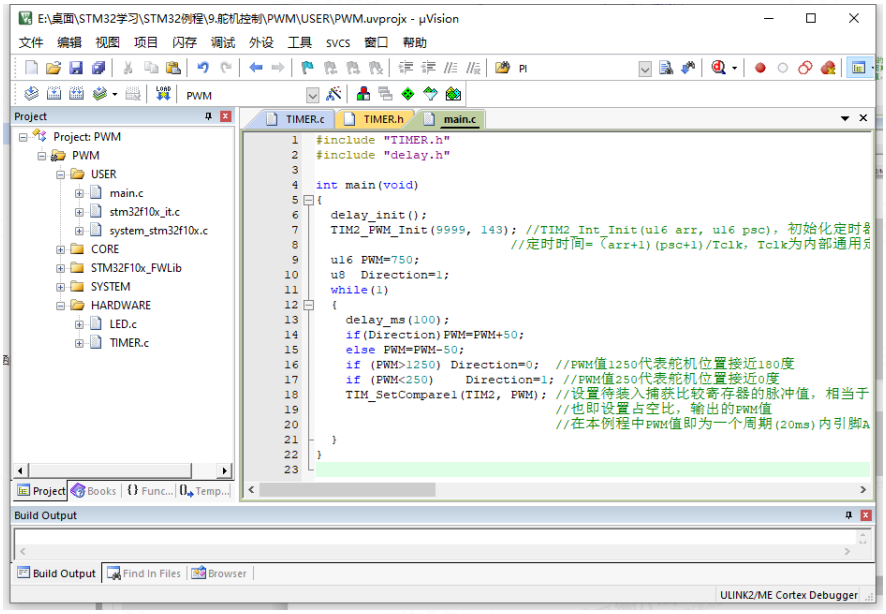

图 8-3-1 主函数

```
#include "TIMER.h"
#include "delay.h"

int main(void)
{
```

1) 调用相关初始化函数：

```
    delay_init();
  TIM2_PWM_Init(9999, 143); //TIM2_Int_Init(u16 arr, u16 psc), 初始化定时器 TIM2
                            //定时时间= (arr+1)(psc+1)/Tclk, Tclk 为内部通用定时
器时钟, 本例程默认设置为 72MHZ
    u16 PWM=750;//舵机中间角度对应 PWM 值
    u8  Direction=1;//PWM 值变化方向
```

2) 主循环，PWM 值循环变大变小，实现效果为舵机在 0-180 度内来回旋转：

```
  while(1)
    {
        delay_ms(100);
        if(Direction)PWM=PWM+50;
        else PWM=PWM-50;
        if (PWM>1250) Direction=0;  //PWM 值 1250 代表舵机位置接近 180 度
        if (PWM<250)    Direction=1; //PWM 值 250 代表舵机位置接近 0 度
        TIM_SetCompare1(TIM2, PWM); //设置待装入捕获比较寄存器的脉冲值, 相当于不
断设置 TIM_Pulse,也即设置占空比,输出的 PWM 值。在本例程中 PWM 值即为一个周期(20ms)
内引脚 A0 输出的高电平时长(单位 2^-3ms)
    }
}
```

8.4 实现思路及效果

(1) 在主函数中根据舵机频率计算好 arr、psc 后调用定时器 PWM 输出初始化函数。

(2) 在 while 循环中不断设置 PWM 值以改变舵机角度。

实现效果为舵机从 0° 到 180° ，再从 180° 到 0° 循环旋转。如果出现舵机转向角度过小问题，可以更换电流更大的电源给舵机供电，STM32F103C8T6 的 5v 引脚电流较小，容易出现难以驱动的问题。

8.5 本节知识要点

根据舵机需要设置定时器 PWM 输出参数，以实现对舵机的控制。

9. 定时器的输入捕获模式

9.1 相关 IO 介绍

本节以捕获 GPIOB0（TIM3_CH3）引脚高电平时间长度为例讲解定时器输入捕获。本节需要用户 DIY 电路进行实验，如下图所示，按键一端接 B0 引脚，按键另一端接 3.3V 引脚（因为 Forest S1 板子上自带的用户按键引脚 A5 没有定时器功能）。按键按下时 B0 引脚就会接通高电平。

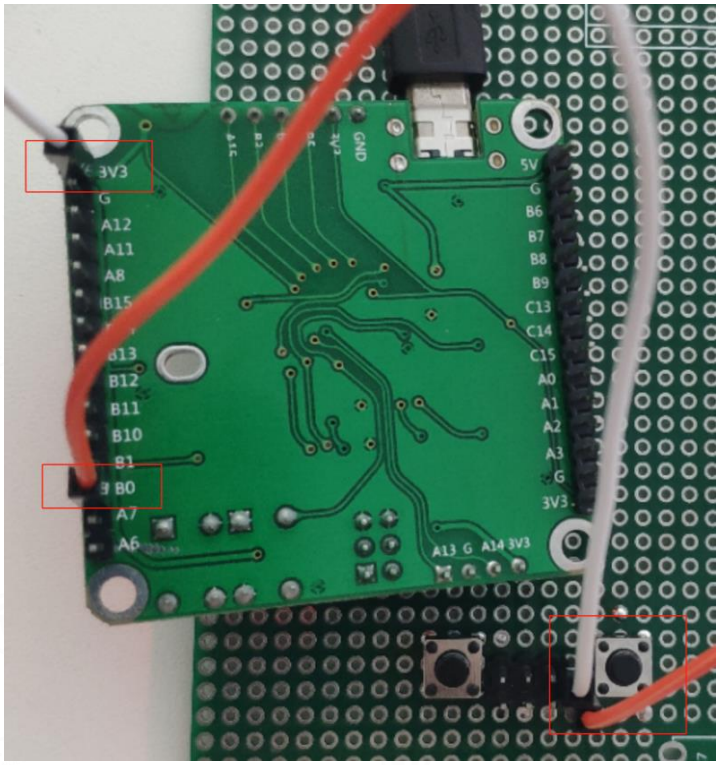

图 9-1-1 STM32 与四脚按键接线

9.2 编写定时器输入捕获库函数

本节直接使用上一节 PWM 输出控制舵机的工程，并将其重命名为【Capture】，不进行重命名也可以，不会影响程序运行。

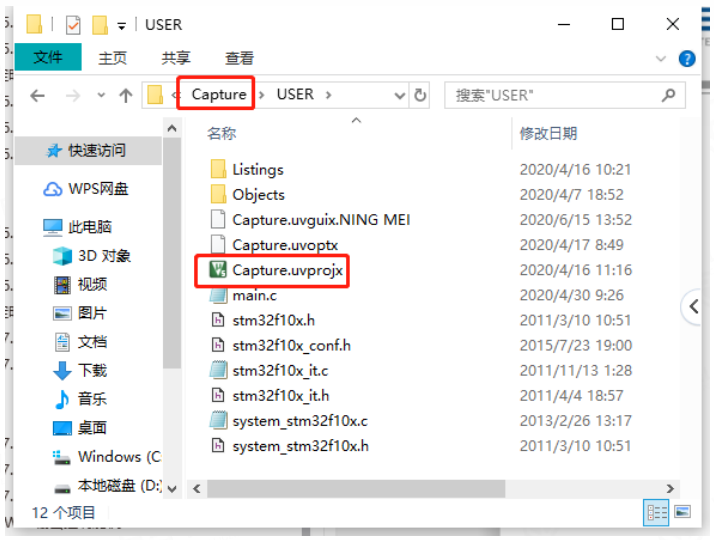

图 9-2-1 重命名工程

① TIMER.h

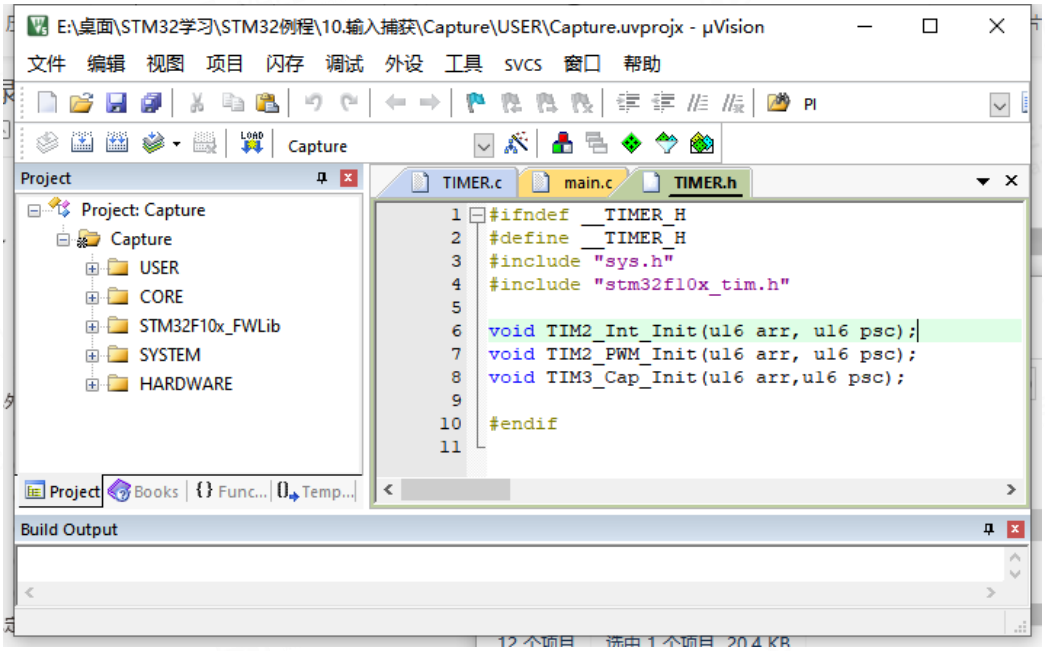

图 9-2-2 TIMER.h

```
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"
#include "stm32f10x_tim.h"

void TIM2_Int_Init(u16 arr, u16 psc);
void TIM2_PWM_Init(u16 arr, u16 psc);
void TIM3_Cap_Init(u16 arr,u16 psc);

#endif
```

② TIMER.c

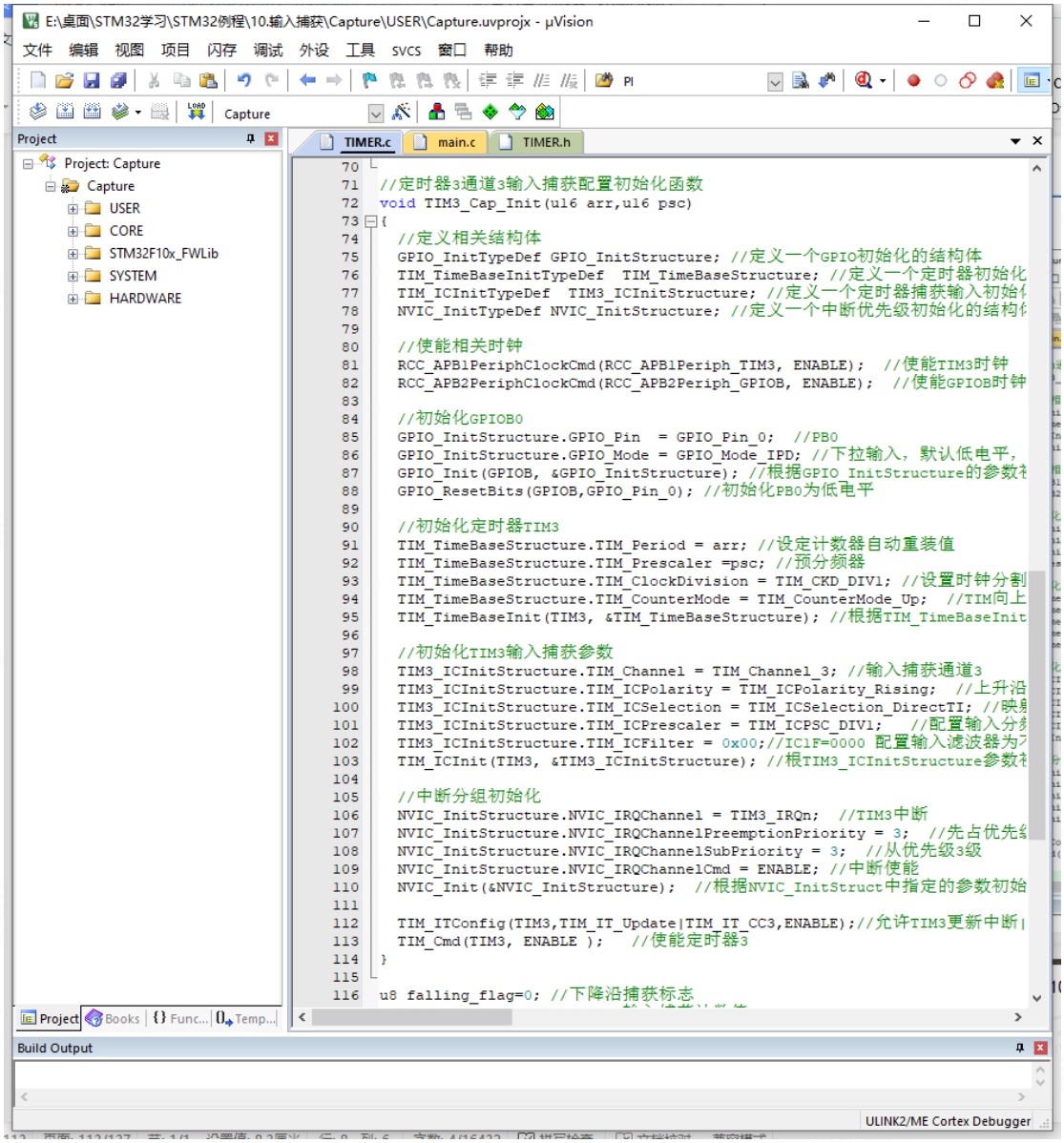

图 9-2-3 TIMER.c

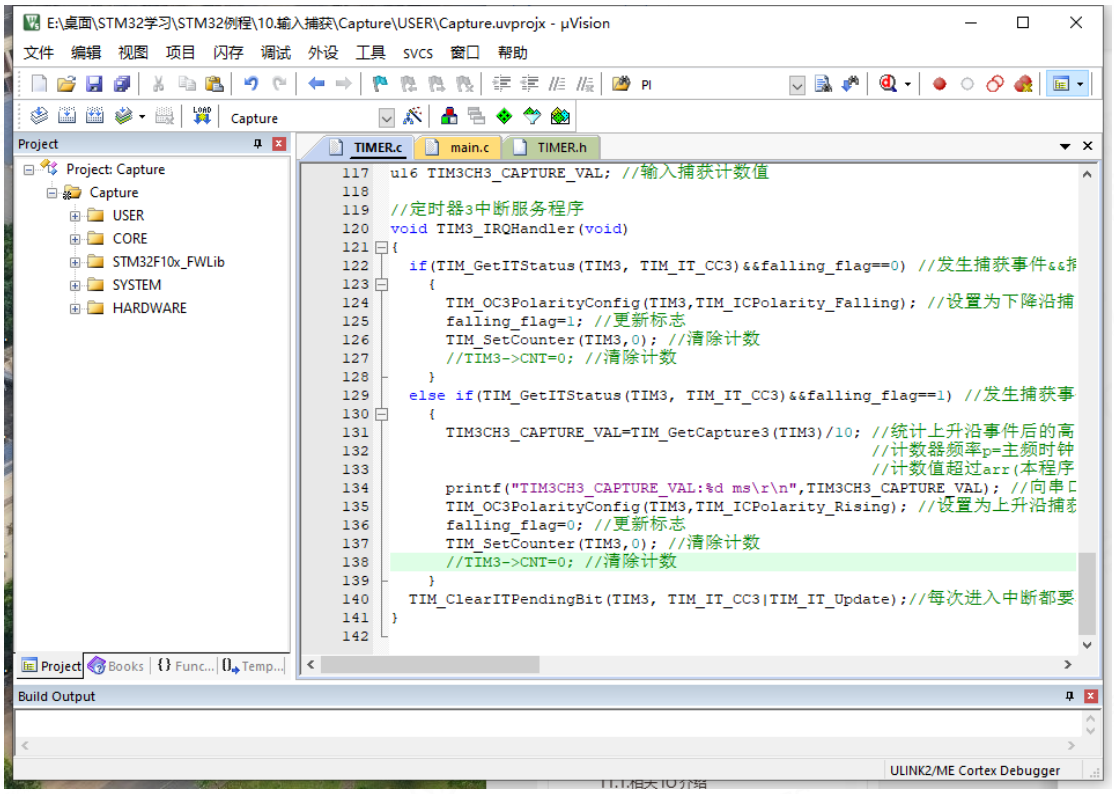

图 9-2-4 TIMER.c

```
//定时器 3 通道 3 输入捕获配置初始化函数
void TIM3_Cap_Init(u16 arr,u16 psc)
{
```

1) 定义相关结构体：

```
    GPIO_InitTypeDef GPIO_InitStructure; //定义一个 GPIO 初始化的结构体
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure; //定义一个定时器初始化的结构体
    TIM_ICInitTypeDef  TIM3_ICInitStructure; //定义一个定时器捕获输入初始化的结构体
    NVIC_InitTypeDef NVIC_InitStructure; //定义一个中断优先级初始化的结构体
```

2) 使能相关时钟：

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);    //使能 TIM3 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);  //使能 GPIOB 时钟
```

3) 初始化 GPIO：

```
    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_0;  //PB0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //下拉输入，默认低电平，可以被外部电压拉高为高电平
    GPIO_Init(GPIOB, &GPIO_InitStructure); //根据 GPIO_InitStructure 的参数初始化 GPIOB0
    GPIO_ResetBits(GPIOB,GPIO_Pin_0); //初始化 PB0 为低电平
```

4) 设置并初始化定时器 TIM3：

```
    TIM_TimeBaseStructure.TIM_Period = arr; //设定计数器自动重装值
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //预分频器
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //设置时钟分割:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM 向上计数模式
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //根据 TIM_TimeBaseInitStruct 的参数初始化定时器 TIM3
```

5) 初始化 TIM3 输入捕获参数：

```
    TIM3_ICInitStructure.TIM_Channel = TIM_Channel_3; //输入捕获通道 3
  TIM3_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;  //上升沿捕获
    TIM3_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI; //映射到 TI3 上
    TIM3_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;   //配置输入分频为不分频，分频决定几个捕获事件触发中断服务函数
    TIM3_ICInitStructure.TIM_ICFilter = 0x00;//IC1F=0000 配置输入滤波器为不滤波
    TIM_ICInit(TIM3, &TIM3_ICInitStructure); //根 TIM3_ICInitStructure 参数初始化定时器 TIM3 输入捕获
```

6) 定时器中断优先级设置：

```
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;  //TIM3 中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;  //先占优先级 3 级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;  //从优先级 3 级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //中断使能
    NVIC_Init(&NVIC_InitStructure);  //根据 NVIC_InitStruct 中指定的参数初始化外设 NVIC 寄存器
```

7) 使能定时器中断：

```
    TIM_ITConfig(TIM3,TIM_IT_Update|TIM_IT_CC3,ENABLE);//允许 TIM3 更新中断|允许 TIM3 通道 3 捕获中断
```

8) 使能定时器：

```
  TIM_Cmd(TIM3, ENABLE );   //使能定时器 3
}
```

9) 定义相关变量：

```
u8 falling_flag=0; //下降沿捕获标志
u16 TIM3CH3_CAPTURE_VAL; //输入捕获计数值
```

10) 定时器 3 中断服务程序，计算按键按下的持续时间：

```
void TIM3_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM3, TIM_IT_CC3)&&falling_flag==0) //发生捕获事件&&捕获
```

```
事件为上升沿（按键按下，B0 接通高电平）
        {
            TIM_OC3PolarityConfig(TIM3,TIM_ICPolarity_Falling); //设置为下降沿
捕获
            falling_flag=1; //更新标志
            TIM_SetCounter(TIM3,0); //清除计数
            //TIM3->CNT=0; //清除计数
        }
  else if(TIM_GetITStatus(TIM3, TIM_IT_CC3)&&falling_flag==1) //发生捕获事件&&
捕获事件为下降沿
    {
        TIM3CH3_CAPTURE_VAL=TIM_GetCapture3(TIM3)/10; //统计上升沿事件后的高电
平持续时间（按键长按时间）,计数器频率 p=主频时钟(72M)/预分频系数
(psc)=72*10^6/7200=10Khz，即 0.0001s=0.1ms 一次计数，计数值超过 arr(本程序设置为
65535)将重新计数

            printf("TIM3CH3_CAPTURE_VAL:%d ms\r\n",TIM3CH3_CAPTURE_VAL); //向串
口调试助手发送统计时间，注意时间上限为 6.5535 秒，超过 6.5535 秒会重新计数
      TIM_OC3PolarityConfig(TIM3,TIM_ICPolarity_Rising); //设置为上升沿捕获
            falling_flag=0; //更新标志
      TIM_SetCounter(TIM3,0);    //清除计数
            //TIM3->CNT=0; //清除计数
      }
  TIM_ClearITPendingBit(TIM3, TIM_IT_CC3|TIM_IT_Update);//每次进入中断都要清空
中断标志，否则主函数将无法正常执行，而一直进入中断服务函数
}
```

9.3 编写主函数

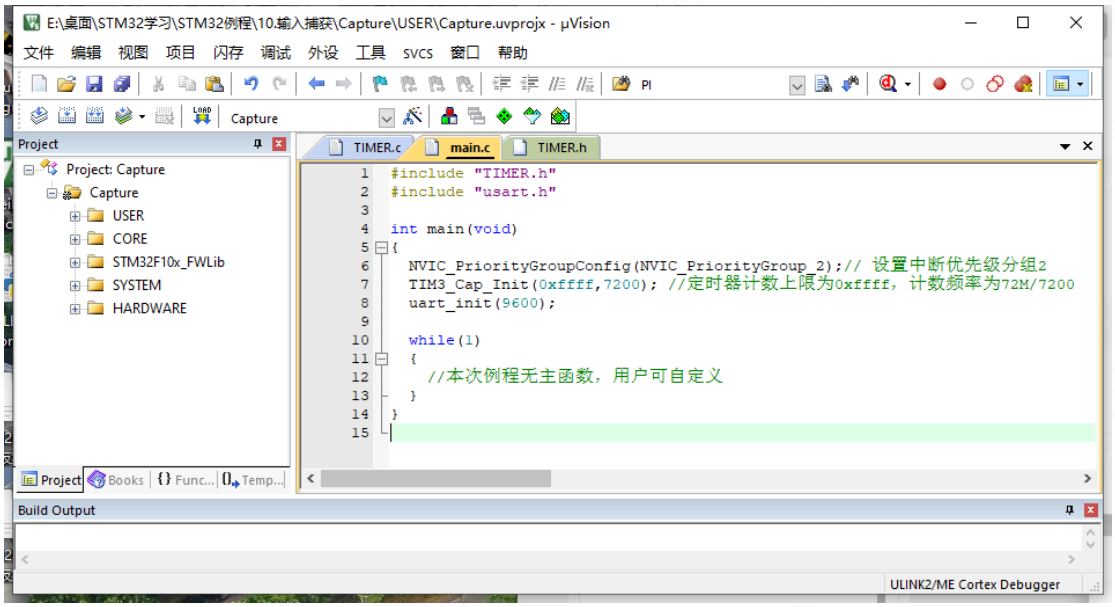

图 9-3-1 主函数

```
#include "TIMER.h"
#include "usart.h"

int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);// 设置中断优先级分组 2
  TIM3_Cap_Init(0xffff,7200);    //定时器计数上限为 0xffff，计数频率为 72M/7200
    uart_init(9600);

  while(1)
    {
    //本次例程无主函数，用户可自定义
    }
}
```

9.4 实现思路及效果

(1) 编写定时器输入捕获初始化函数。

初始化 GPIO、定时器时钟→初始化 GPIO 引脚→初始化定时器→初始化定时器输入捕获→初始化中断分组→使能定时中断、输入捕获中断→使能定时器。

(2) 编写定时器中断服务函数。

按键按下产生上升沿，激活定时器输入捕获中断，进入中断服务函数，定时器重新计数，设置为下降沿捕获；按键松开产生下降沿，进入中断服务函数，读取计数值，设置为上升沿捕获。

(3) 在主函数中定义相关初始化函数。

实现效果为按下按键并松开后，串口调试助手会显示按键按下的时长（单位 ms，上限 59.652 秒）。

图 9-1 实现效果

9.5 本节知识要点

- 定时器输入捕获参数的初始化：选择输入捕获通道 3→选择捕获模式→选择映射方式→配置预分频→配置滤波。
- 重新设置捕获模式函数、获取计数值函数和清除计数值函数的使用。

10. 定时器的编码器模式

10.1 相关 IO 介绍

在 STM32F103C8T6 中，编码器使用的是定时器接口，定时器 1，2，3，4 有编码器的功能。同时只有 CH1 和 CH2 可以进行编码器模式。

本节以 B6（TIM4CH1）、B7（TIM4CH2）为例进行讲解。

接线如下表 10-1、图 10-1-1 所示：

表 10-1

STM32	GND	5V	B6	B7	GND	3.3V
电机	电机线-	编码器 5V	编码器 A	编码器 B	编码器 GND	电机线+

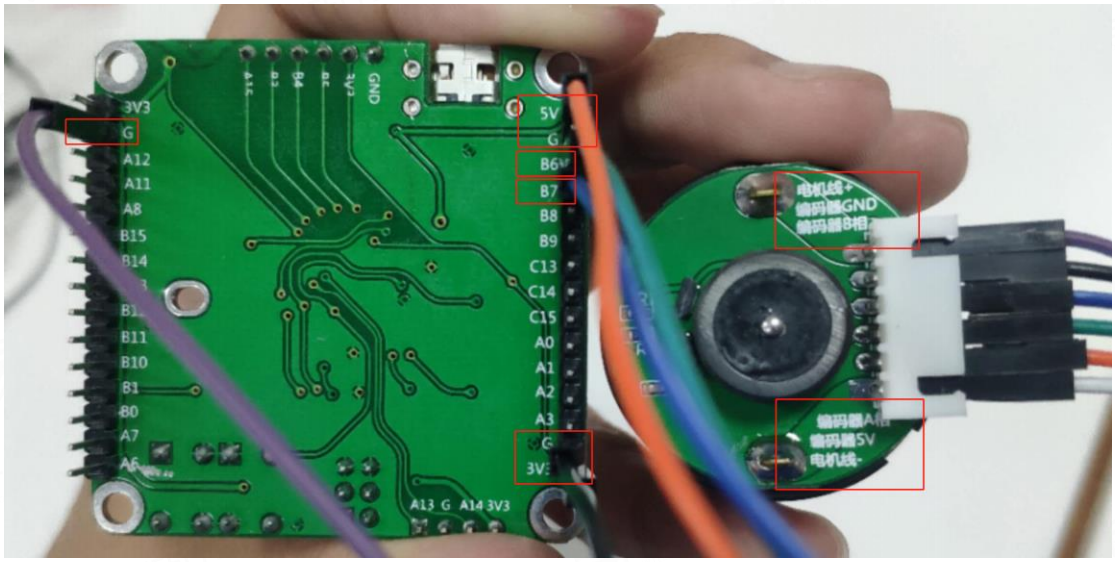

图 10-1-1 STM32 与编码器接线

10.2 编写编码器库函数

创建库函数并进入工程的过程之前的章节已阐述多次，本节不在重复。

① ENCODER.h

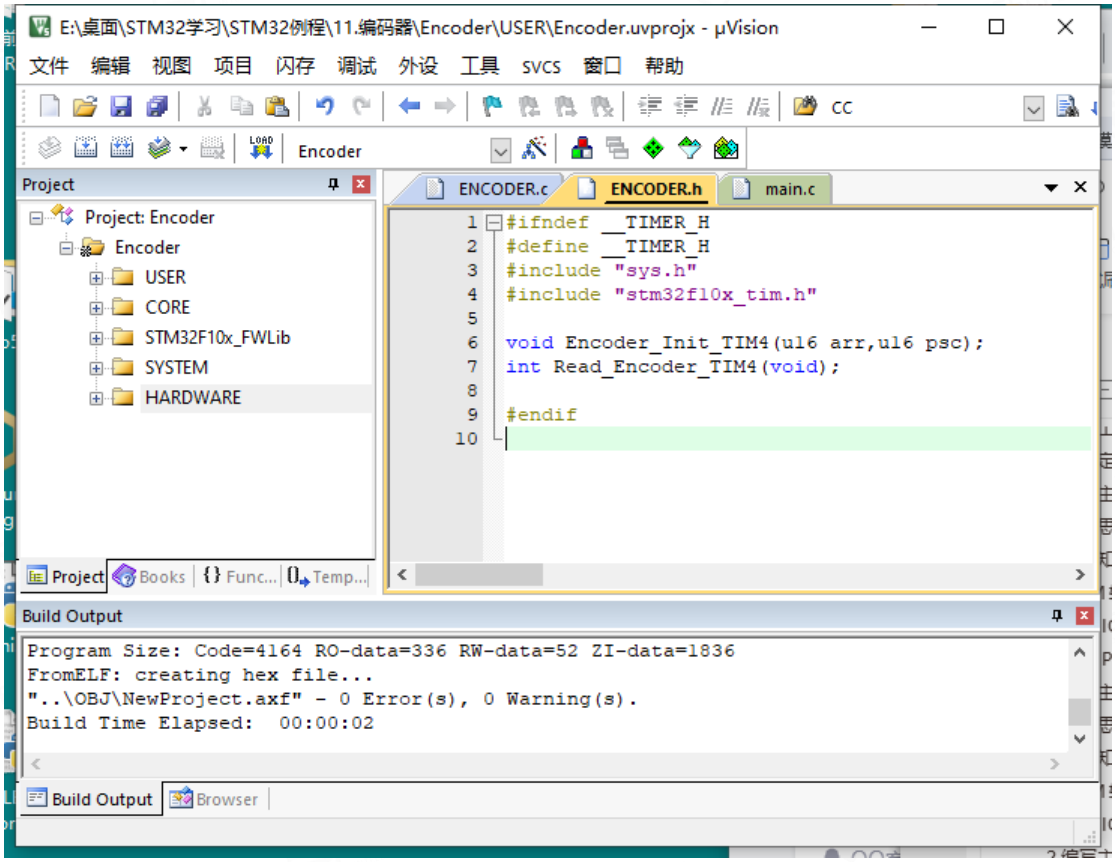

图 10-2-1 TIMER.h

```
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"
#include "stm32f10x_tim.h"

void Encoder_Init_TIM4(u16 arr,u16 psc);
int Read_Encoder_TIM4(void);

#endif
```

② ENCODER.c

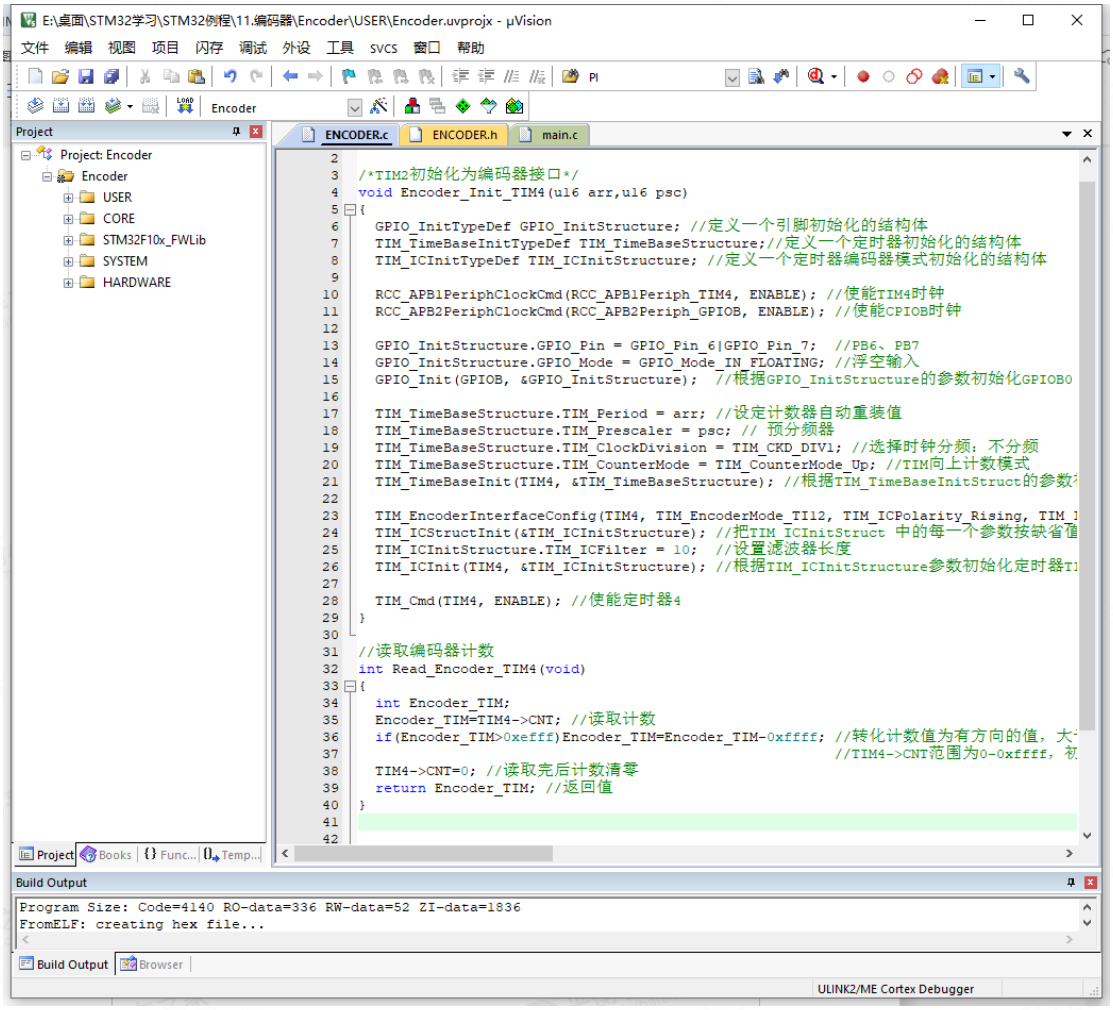

图 10-2-2 TIMER.c

```
#include "ENCODER.h"

/*TIM2 初始化为编码器接口*/
void Encoder_Init_TIM4(u16 arr,u16 psc)
{
```

1) 定义相关结构体：

```
    GPIO_InitTypeDef GPIO_InitStructure; //定义一个引脚初始化的结构体
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;//定义一个定时器初始化的结构体
  TIM_ICInitTypeDef TIM_ICInitStructure; //定义一个定时器编码器模式初始化的结构体
```

2) 使能相关时钟：

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); //使能 TIM4 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //使能 GPIOB 时钟
```

3) 初始化 GPIO：

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;     //PB6、PB7
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //浮空输入
    GPIO_Init(GPIOB, &GPIO_InitStructure);  //根据 GPIO_InitStructure 的参数初始化 GPIOB0
```

4) 设置并初始化定时器 TIM4：

```
    TIM_TimeBaseStructure.TIM_Period = arr; //设定计数器自动重装值
    TIM_TimeBaseStructure.TIM_Prescaler = psc; // 预分频器
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //选择时钟分频：不分频
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM 向上计数模式
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure); //根据 TIM_TimeBaseInitStruct 的参数初始化定时器 TIM4
```

5) 设置并初始化定时器编码器：

```
   TIM_EncoderInterfaceConfig(TIM4, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);//使用编码器模式 3：CH1、CH2 同时计数，四分频
    TIM_ICStructInit(&TIM_ICInitStructure); //把 TIM_ICInitStruct 中的每一个参数按缺省值填入
    TIM_ICInitStructure.TIM_ICFilter = 10;  //设置滤波器长度
    TIM_ICInit(TIM4, &TIM_ICInitStructure); //根 TIM_ICInitStructure 参数初始化定时器 TIM4 编码器模式
```

6) 使能定时器：

```
    TIM_Cmd(TIM4, ENABLE); //使能定时器 4
}
```

7) 读取编码器计数函数，以一定周期调用该函数，读取后清零计数：

```
int Read_Encoder_TIM4(void)
{
    int Encoder_TIM;
    Encoder_TIM=TIM4->CNT; //读取计数
    if(Encoder_TIM>0xefff)Encoder_TIM=Encoder_TIM-0xffff; //转化计数值为有方向的值，大于 0 正转，小于 0 反转。
                                                          //TIM4->CNT 范围为 0-0xffff，初值为 0。
    TIM4->CNT=0; //读取完后计数清零
    return Encoder_TIM; //返回值
}
```

10.3 编写主函数

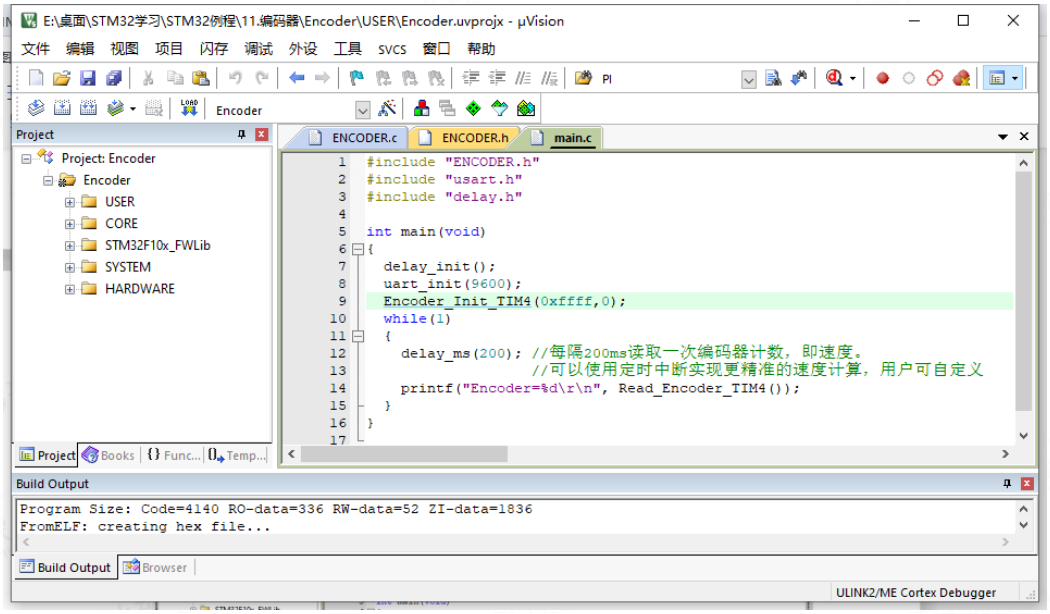

图 10-3-1 主函数

```
#include "ENCODER.h"
#include "usart.h"
#include "delay.h"
int main(void)
{
```

1) 调用相关初始化函数：

```
    delay_init();
    uart_init(9600);
    Encoder_Init_TIM4(0xffff,0);
```

2) 在主循环中以一定周期调用读取编码器计数并串口发送该计数：

```
  while(1)
    {
        delay_ms(200); //每隔 200ms 读取一次编码器计数，即速度。可以使用定时中断实现更精准的速度计算，用户可自定义
    printf("Encoder=%d\r\n", Read_Encoder_TIM4());
    }
}
```

10.4 实验思路及效果

(1) 编写编码器初始化函数

使能 GPIO、定时器时钟→初始化 GPIO(两个)引脚→初始化定时器→初始化定时器编码器→使能定时器。

(2) 编写读取编码器计数值函数

读取编码器计数并判断方向正反，读取完后将计数值清零。

(3) 编写主函数

调用相关初始化函数，在 while 循环中以一定周期读取并向串口调试助手发送编码器计数值（即电机速度），每次读取编码器计数会在读取函数中将编码器计数清零。

(4) 打开串口调试助手，调波特率为 9600（对应主程序的函数调用 uart_init(9600);），打开串口。

图 10-2 实验效果

实验效果为串口调试助手显示编码器计数，计数大小与方向随电机速度与方向变化，电机速度变化可以通过调节电机电源实现，电机速度方向可以通过反接电源实现。

图 10-3 电机电源接口

10.5 本节知识要点

- 定时器编码器模式的初始化：选择编码器模式→编码器模式结构体默认初始化→设置滤波器长度→初始化。
- 编码器计数的读取与置 0。

11. ADC 采集

11.1 相关 IO 介绍

STM32F103C8T6 上有 1 个 ADC 控制器，每个 ADC 控制器有 16 个通道。

本节以 GPIOA0（ADC1 通道 0）引脚为例进行讲解。接线如下图 11-1-1 所示，A0 接 3V3，用来测量 3V3 引脚的实际电压（有条件可以接其它电压值，注意不能超过 3.3V）。

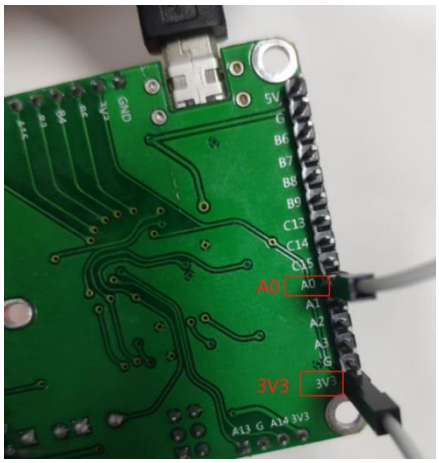

图 11-1-1 ADC 控制引脚接线图

11.2 编写 ADC 采集库函数

① ADC.h

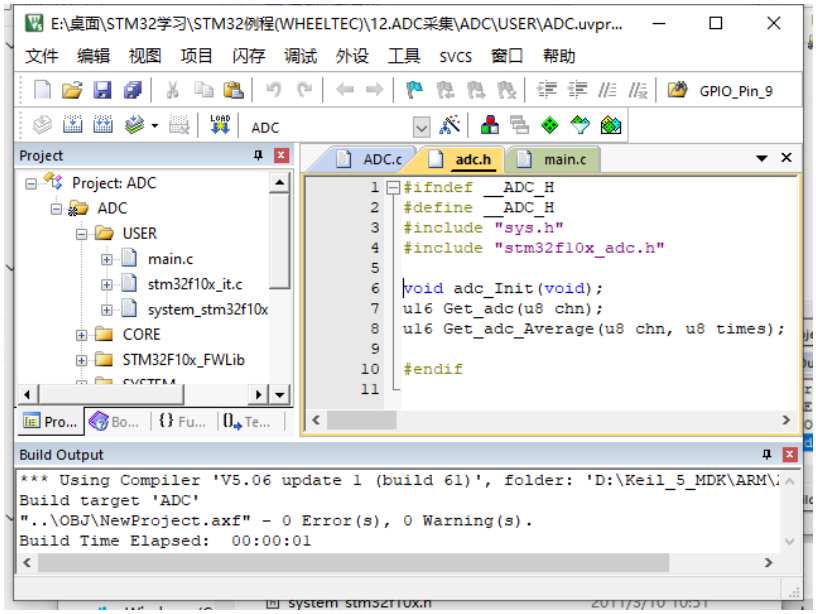

图 11-2-1 ADC.h

```
#ifndef __ADC_H
#define __ADC_H
#include "sys.h"
#include "stm32f10x_adc.h"

void adc_Init(void);
u16 Get_adc(u8 chn);
u16 Get_adc_Average(u8 chn, u8 times);

#endif
```

② ADC.c

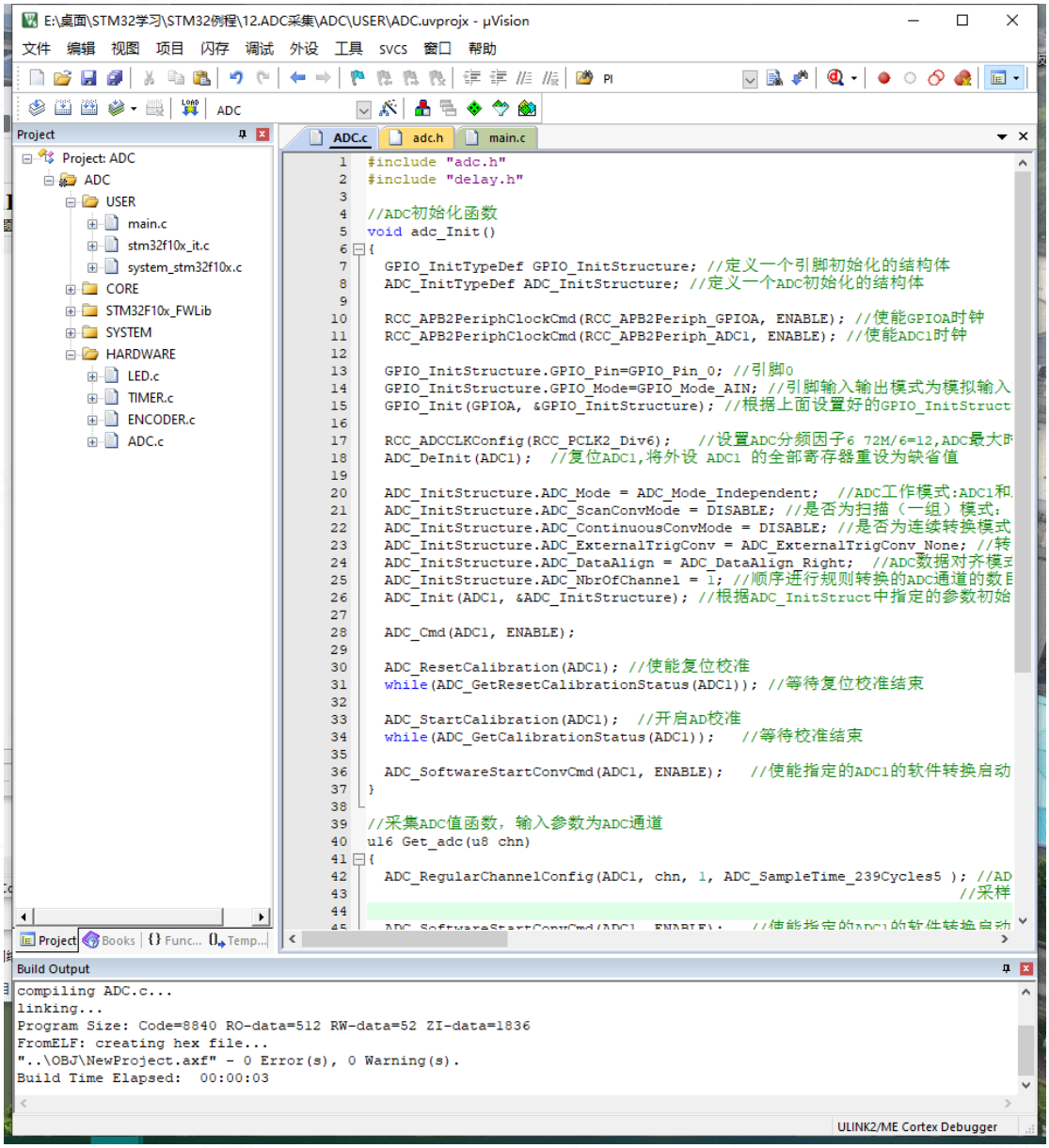

图 11-2-2 ADC.c

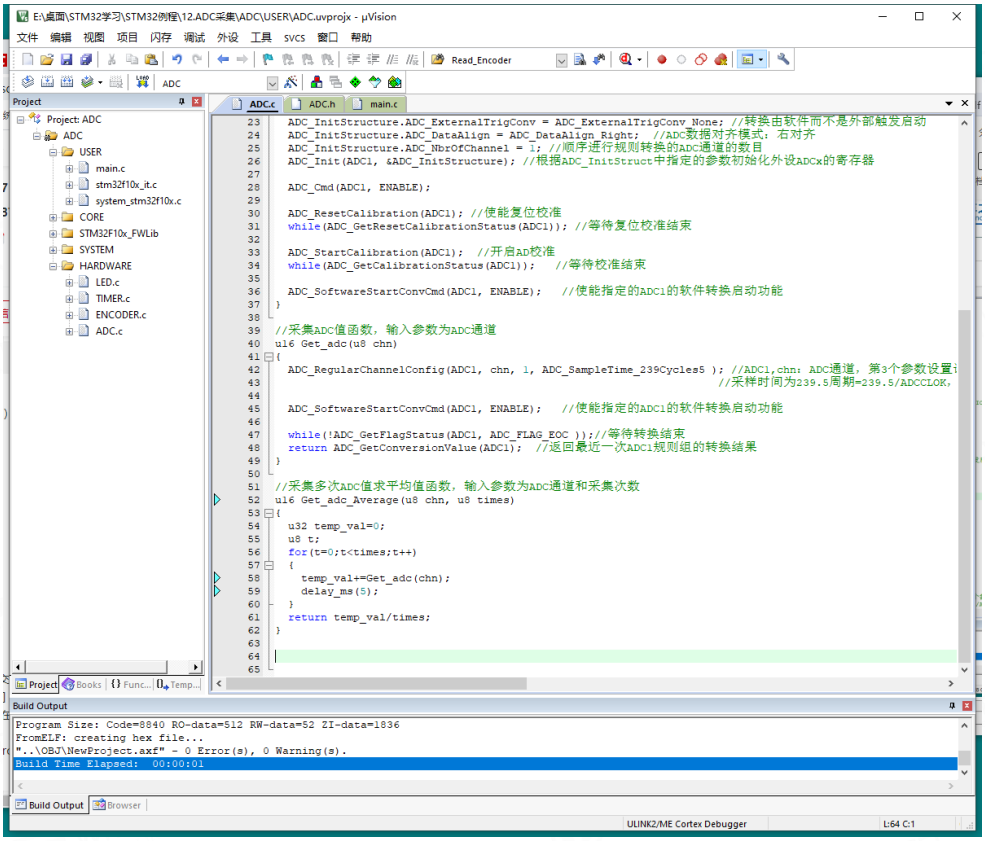

图 11-2-3 ADC.c

```
#include "adc.h"
#include "delay.h"
//ADC 初始化函数
void adc_Init()
{
```

1) 定义相关结构体：

```
  GPIO_InitTypeDef GPIO_InitStructure; //定义一个引脚初始化的结构体
  ADC_InitTypeDef ADC_InitStructure; //定义一个 ADC 初始化的结构体
```

2) 使能相关时钟：

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //使能 GPIOA 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE); //使能 ADC1 时钟
```

3) 初始化 GPIO：

```
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_0; //引脚 0
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AIN; //引脚输入输出模式为模拟输入模
式
    GPIO_Init(GPIOA, &GPIO_InitStructure); //根据上面设置好的 GPIO_InitStructure
参数，初始化引脚 GPIOA_PIN0
```

4) 设置并初始化 ADC：

```
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);    //设置 ADC 分频因子 6 72M/6=12,ADC 最大时
间不能超过 14M
```

```
    ADC_DeInit(ADC1);  //复位 ADC1,将外设 ADC1 的全部寄存器重设为缺省值
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //ADC 工作模式:ADC1 和
ADC2 工作在独立模式
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;   //是否为扫描(一组)模式:否:
单通道模式
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;//是否为连续转换模式,否:
单次转换模式
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //转换
由软件而不是外部触发启动
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //ADC 数据对齐模式:
右对齐
    ADC_InitStructure.ADC_NbrOfChannel = 1; //顺序进行规则转换的 ADC 通道的数目
    ADC_Init(ADC1, &ADC_InitStructure); //根据 ADC_InitStruct 中指定的参数初始化
外设 ADCx 的寄存器
```

5) 使能 ADC：

```
    ADC_Cmd(ADC1, ENABLE);
```

6) 进行必要的校准：

```
    ADC_ResetCalibration(ADC1); //使能复位校准
    while(ADC_GetResetCalibrationStatus(ADC1)); //等待复位校准结束
    ADC_StartCalibration(ADC1);  //开启 AD 校准
    while(ADC_GetCalibrationStatus(ADC1));   //等待校准结束
```

7) 使能软件转换启动功能，使能后可以直接调用相关函数读取 ADC 值：

```
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);     //使能指定的 ADC1 的软件转换启动
功能
}
```

8) 读取 ADC 值函数，输入参数为 ADC 通道：

```
u16 Get_adc(u8 chn)
{
    ADC_RegularChannelConfig(ADC1, chn, 1, ADC_SampleTime_239Cycles5 );
    //ADC1,chn: ADC 通道, 第 3 个参数设置该通道的转换顺序 (多通道模式下)
//采样时间为 239.5 周期=239.5/ADCCLOK, ADCCLOK=72/6MHZ(6 代表 ADC 初始化时的分频系
数)
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);     //使能指定的 ADC1 的软件转换启动
功能
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
    return ADC_GetConversionValue(ADC1);    //返回最近一次 ADC1 规则组的转换结果
}
```

9) 采集多次 ADC 值求平均值函数，输入参数为 ADC 通道和采集次数：

```
u16 Get_adc_Average(u8 chn, u8 times)
{
  u32 temp_val=0;
    u8 t;
```

```
    for(t=0;t<times;t++)
    {   temp_val+=Get_adc(chn);
        delay_ms(5);  }
    return temp_val/times;
}
```

11.3 编写主函数

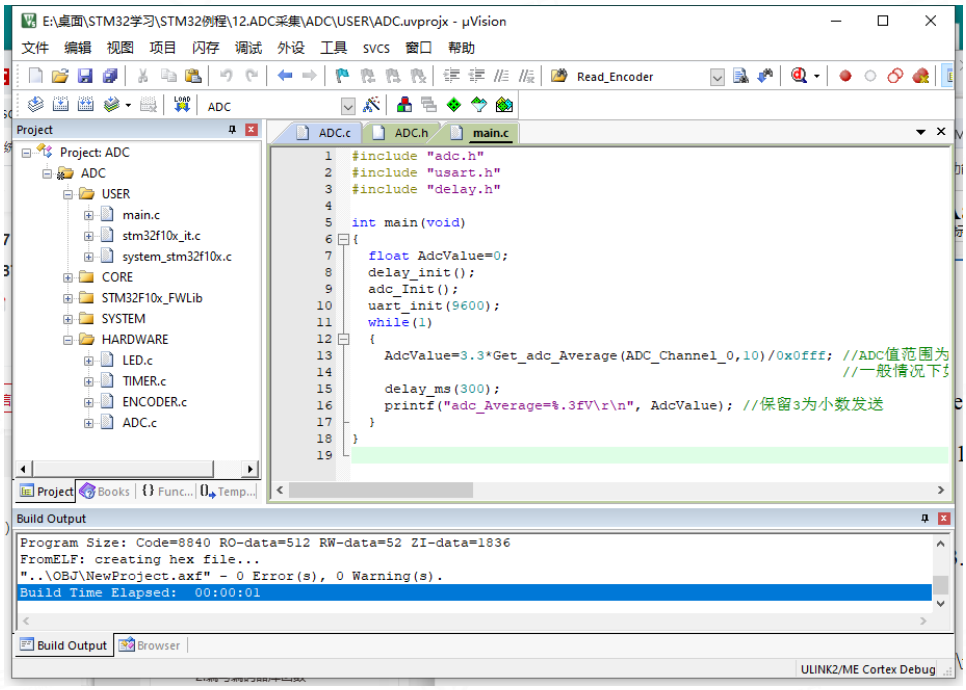

图 11-3-1 主函数

```
#include "adc.h"
#include "usart.h"
#include "delay.h"
int main(void){
```

1) 定义相关变量和调用相关初始化函数：

```
    float AdcValue=0;
    delay_init();
    adc_Init();
    uart_init(9600);
    while(1){
```

2) 读取 ADC 值并串口发送 ADC 值：

```
    AdcValue=3.3*Get_adc_Average(ADC_Channel_0,10)/0x0fff; //ADC 值范围为从
0-2^12=4095（111111111111）。一般情况下如本例程对应电压为 0-3.3V
    delay_ms(300);
    printf("adc_Average=%.3fV\r\n", AdcValue); //保留 3 为小数发送
    }
 }
```

11.4 实验思路及效果

(1) 编写 ADC 初始化函数

使能 GPIO、ADC 时钟→初始化 GPIO 引脚→初始化 ADC→使能 ADC→复位并开启 ADC 校准。

(2) 编写采集 ADC 值函数采集

设置指定 ADC 通道的序号（序号在多通道模式下才有意义）、采样时间→使能软件转换→返回 ADC 值。

(3) 编写采集多次 ADC 并求平均值函数

(4) 编写主函数

调用相关初始化函数，在 while 循环中以一定周期读取并向串口调试助手发送 ADC 值（即 A0 口外接电压值）。

图 11-3 实现效果

11.5 本节知识要点

- ADC 的初始化：选择 ADC 工作模式→选择通道模式→选择转换模式→选择启动触发发送→选择 ADC 值对齐方式→设置所用通道总数目→根据上述参

数初始化 ADC→最后复位并使能 ADC 校准。

- 指定通道参数的设置：序号、采样周期。
- ADC 软件转换功能的开启与 ADC 采集函数的使用。
- ADC 值到实际电压值的转换。