实验 5

EXPERIMENT 5

GPIO 设备编程—输出实验（固态库点亮 LED 灯）

5.1 实验目的

- 掌握引脚连接模块的配置；
- 掌握固件库的使用；
- 熟悉代码的调试。

5.2 实验设备

1. 硬件

(1) PC 一台；
(2) STM32F429IGT6 核心板一块；
(3) DAP 仿真器一个；
(4) LED 灯 3 个；
(5) 导线若干根；
(6) 限流电阻若干只；
(7) 面包板一块。

2. 软件

(1) Windows 7/8/10 系统；
(2) Keil μVision5 集成开发环境。

5.3 实验内容

本实验基于 Keil μVision5 集成开发环境，通过使用固件库，建立自己的工程模板。配置完引脚连接模块后，便可将编写好的一个最基本、最简单的真正实用的工程进行编译，并可将程序烧入开发板，驱动 ARM Cortex-M4 芯片，点亮 LED 灯，以达到本次实验的目的。

5.4 实验预习

- 了解 GPIO 的控制方法；
- 学习固件库的使用方法；
- 仔细阅读 Keil 相关资料，了解 Keil 工程编辑和调试的内容。

5.5 实验原理

实验 1 通过 STM32 的官方网站下载了 STM32F4 的固件库，在实验 4 中用到了一个启动文件 startup_stm32f429_439xx.s，通过该汇编程序初始化开发板，进入 main 函数。在实验 4 中，通过对 GPIO 端口对应的寄存器的配置，完成了对该端口上连接的 LED 灯的点亮与熄灭。在使用寄存器开发应用时，工程师需要对每个控制的寄存器都非常熟悉，并且在修改寄存器相应位时，需手工写入特定参数。开发者查看硬件手册中寄存器的说明，根据说明对照设置，以这样的方式配置寄存器时容易出错，并且代码不好理解，也不便于维护。

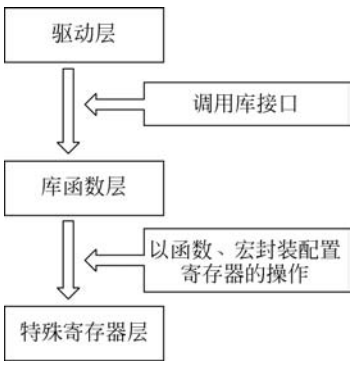

图 5.1 库开发方式

根据寄存器的地址特点，解决此问题的最好方法是使用软件库，而且芯片开发商也提供了这样的软件库，例如，ST 公司提供了 STM32 的标准函数库（也即固件库），这一固件库提供了通过函数设置访问寄存器的方法。

固件库是寄存器与用户驱动层之间的代码，向下处理与寄存器直接相关的配置，向上为用户提供配置寄存器的接口，如图 5.1 所示。

对于应用层需要操作的寄存器，以函数、宏定义的形式封装后，对这类寄存器的操作就可以通过调用这些接口函数完成对硬件寄存器的配置。与直接配置寄存器相比，只要用户配置好被调用函数的参数，就可以方便地使用了，这种配置通常都不会出错。

5.5.1 GPIO 寄存器的数据结构

回忆一下，在实验 4 中，每个 GPIO 组的 10 个寄存器的地址，是相对于其基地址连续的。比如 GPIOB 的基地址是 0x40020400，紧随它的是 10 个寄存器的地址，也就是说，这 10 个寄存器的地址是 0x40020400～0x40020424，寄存器的地址是基地址加偏移地址，偏移地址是连续递增的，这种方式与结构体中的成员类似。这样自然就会想到，通过结构体就可以表示这些寄存器了，结构体成员的顺序按照寄存器的偏移地址从低到高排列，成员类型与寄存器类型一样，设置成 32 位的无符号型。对寄存器的访问，只要知道基地址，就可通过类型转换成该结构体类型，那么对寄存器的访问就是对结构体成员的访问。

在固件库中，库文件 STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\CMSIS\Device\ST\STM32F4xx\Include\stm32xx.h 给出了常用寄存器结构体的定义，比如 GPIO

的 10 个寄存器的结构体的定义如下：

```
1471  typedef struct
1472  {
1473    __IO uint32_t MODER;
1474    __IO uint32_t OTYPER;
1475    __IO uint32_t OSPEEDR;
1476    __IO uint32_t PUPDR;
1477    __IO uint32_t IDR;
1478    __IO uint32_t ODR;
1479    __IO uint16_t BSRRL;
1480    __IO uint16_t BSRRH;
1481    __IO uint32_t LCKR;
1482    __IO uint32_t AFR[2];
1483  } GPIO_TypeDef;
```

其中类型的定义为

```
//volatile 表示易变的变量,防止编译器优化
#define __IO volatile
typedef unsigned int uint32_t;
typedef unsigned short uint16_t;
```

若要访问某个 GPIO，比如 GPIOB，则只需要在 stm32f4xx.h 中设置其首地址就可以了，代码如下：

```
2050  #define PERIPH_BASE           ((uint32_t)0x40000000)
2086  #define APB1PERIPH_BASE       PERIPH_BASE
2208  #define GPIOB_BASE            (AHB1PERIPH_BASE + 0x0400)
2386  #define GPIOB                 ((GPIO_TypeDef *) GPIOB_BASE)
```

定义了访问外设的结构体指针，通过强制把外设的基地址转换成 GPIO_TypeDef 类型的地址，通过结构体指针操作，可访问外设的寄存器。对于 GPIOB 的某个寄存器访问就可以通过结构体的成员访问。例如，清空 GPIOB MODER12，可以通过下面的代码完成：

```
GPIOB->MODER &= ~(0x3 << (2 * 12));
```

对寄存器的操作也可以通过调用接口函数完成。GPIO 的相关操作的头文件为\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\STM32F4xx_StdPeriph_Driver\inc\stm32f4xx_gpio.h。而其他外设相关的接口函数都在文件夹\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\STM32F4xx_StdPeriph_Driver\inc 中的头文件中，大家需要时可以去查找。

5.5.2　GPIO 初始化

由实验 4 可知，为了配置一个端口，需要配置 4 个寄存器，为了方便，可建立一个数据结构专门用于初始化端口，该数据结构在 stm32f4xx_gpio.h 中，代码如下：

```
typedef struct
{
    uint32_t GPIO_Pin;
    GPIOMode_TypeDef GPIO_Mode;
    GPIOSpeed_TypeDef GPIO_Speed;
    GPIOOType_TypeDef GPIO_OType;
    GPIOPuPd_TypeDef GPIO_PuPd;
}GPIO_InitTypeDef;
```

GPIO_Pin 指出是对哪个引脚进行初始化，而其他 4 个成员就是对应的 GPIO 的一个配置寄存器。GPIOMode _ TypeDef、GPIOOType _ TypeDef、GPIOSpeed _ TypeDef 和 GPIOPuPd_TypeDef 都是枚举类型，定义如下：

```
typedef enum
{
    GPIO_Mode_IN      = 0x00, /*!< GPIO Input Mode */
    GPIO_Mode_OUT     = 0x01, /*!< GPIO Output Mode */
    GPIO_Mode_AF      = 0x02, /*!< GPIO Alternate function Mode */
    GPIO_Mode_AN      = 0x03  /*!< GPIO Analog Mode */
}GPIOMode_TypeDef;

typedef enum
{
    GPIO_OType_PP     = 0x00,
    GPIO_OType_OD     = 0x01
}GPIOOType_TypeDef;

typedef enum
{
    GPIO_Low_Speed    = 0x00, /*!< Low speed     */
    GPIO_Medium_Speed = 0x01, /*!< Medium speed */
    GPIO_Fast_Speed   = 0x02, /*!< Fast speed    */
    GPIO_High_Speed   = 0x03  /*!< High speed    */
}GPIOSpeed_TypeDef;

typedef enum
{
    GPIO_PuPd_NOPULL  = 0x00,
    GPIO_PuPd_UP      = 0x01,
    GPIO_PuPd_DOWN    = 0x02
}GPIOPuPd_TypeDef;
```

例如，需要配置一个引脚 GPIOB12，就可以通过下面的代码完成：

```
GPIO_InitTypeDef  GPIO_InitStructure
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
GPIO_InitStructure.GPIO_Speed = GPIO_High_Speed;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

解释如下：

(1) 定义一个特殊类型的结构体。

语句 GPIO_InitTypeDef　GPIO_InitStructure 定义了一个 GPIO_InitTypeDef 类型的结构体。

(2) 选择要控制的 GPIO 引脚。

语句 GPIO_InitStructure.GPIO_Pin=GPIO_Pin_12 表示应用第 12 引脚。

(3) 设置引脚模式。

语句 GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT 定义该引脚的模式为输出模式。

(4) 设置推挽模式还是开漏模式。

语句 GPIO_InitStructure.GPIO_OType=GPIO_OType_PP 定义引脚输出类型为推挽模式。

(5) 设置引脚速率。

语句 GPIO_InitStructure.GPIO_Speed=GPIO_High_Speed 表示引脚速率为 100Hz。

(6) 设置上拉/下拉模式。

语句 GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP 表示该引脚为上拉模式。

(7) 调用库函数，初始化 GPIOB12。

初始化语句为“GPIO_Init(GPIOB,&GPIO_InitStructure);”，初始化引脚 GPIOB12。该函数是在 stm32f4_gpio.c 中定义的。另外，为了使这个引脚能够正常工作，还需要设置其时钟。

(8) 配置 GPIO 的外设时钟。

例如，“RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB,ENABLE);”开启了 GPIOB 的时钟，其中函数 RCC_AHB1PeriphClockCmd()是在固件库 stm32f4xx_rcc.c 中。总结一下，GPIO 的寄存器配置主要有两个步骤：配置 GPIO 时钟和配置 GPIO 寄存器。

5.6　实验步骤

5.6.1　硬件连接

本次实验将通过调用固件库，完成 LED 灯的点亮与熄灭，其硬件连接与实验 4 一样，如图 5.2 所示。

5.6.2　实验讲解

在该工程中，用户需要创建 3 个文件，分别为 led.h、led.c 以及 main.c，而在实验 4 中用户编写的 stm32f4xx.h 文件在本实验中不需要自己编写，直接复制固件库中的文件就可以了。

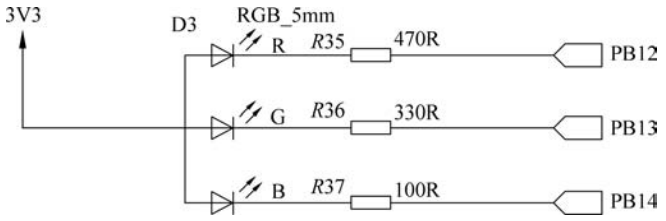

图 5.2　硬件连接

1. led.h

在本实验中是通过固件库来点亮连接在 GPIOB12 引脚上的 LED 灯，在此头文件中对引脚进行了宏定义，以便在引用时更直观，其代码如下：

```
#define LED1_PIN GPIO_Pin_12
#define LED1_GPIO GPIOB
#define LED1_CLK  RCC_AHB1Periph_GPIOB
```

另外，通过一个函数来配置 GPIOB12。在 led.h 头文件中，对这个函数进行了说明，其代码如下：

```
/*配置的函数说明*/
void LED_Config(void);
```

2. led.c

在该文件中完成对配置函数 LED_Config()的定义，对引脚 GPIOB12 的配置在前面已经介绍了，此处不再赘述。应当注意的是，应包含头文件 led.h。

3. main.c

由于在 led.c 文件中有了配置函数，这里就非常简单了，只需要 3 步：

(1) 定义延迟函数。与前一个例子一样，在复制固态函数时，它的模板里有一个 main.h 文件，在这个文件中有对延迟函数的说明，该函数为 TimingDelay_Decrement，对这个函数的定义如下：

```
void TimingDelay_Decrement(void)
{
    int i,j;
    for(i=0;i<5;i++)
    {
        j=5000000;
        while(j>=0) j--;
    }
}
```

(2) 配置 GPIOB12 引脚。由于在 led.c 中定义了配置函数，在 main 函数中只需要调用就可以了：

```
LED_Config( );
```

(3) 点亮、熄灭 LED。通过一个 while(1)循环完成。这里调用库函数完成 LED 灯的点亮与熄灭,这两个函数在 stm32f4xx_gpio.c 文件中:

```
GPIO_SetBits(LED1_GPIO, LED1_PIN)          /* 点亮 LED1 灯 */
GPIO_ResetBits(LED1_GPIO, LED1_PIN)        /* 熄灭 LED1 灯 */
```

5.6.3　创建工程

1. 准备工作

在创建工程之前,需要建立一个文件夹 ex5_LED,并在其下新建 6 个文件夹,分别命名为 CMSIS、CORE、Driver、INIT、Project 和 USR,如图 5.3 所示。

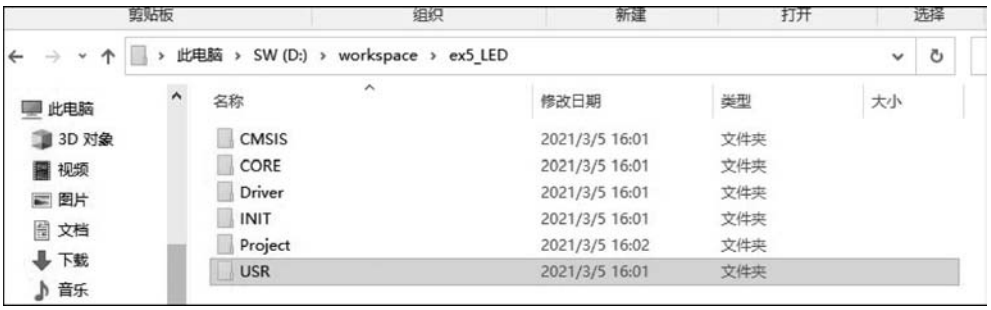

图 5.3　创建文件夹

启动 Keil μVision5,选择 Project→New μVision Project,会弹出一个文件选项,将新建的工程文件保存在之前建立的 ex5_LED\Project 文件夹下,并取名为 led,建立方法与实验 4 一样,单击“保存”按钮。

2. 添加固件库文件

将固件库中的各类所需的文件添加到我们建立的工程文件夹中。添加方法就是直接复制,具体的复制源和目的位置如表 5.1 所示。

表 5.1　复制固件库的文件

工程文件夹	复制文件	文件说明	固件库的位置
INIT	startup_stm32f429_439xx.s	初始化文件	…\en.stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm
CORE	stm32f4xx.h system_stm32f4xx.h	外设寄存器定义 用于系统初始化	…\en.stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\CMSIS\Device\ST\STM32F4xx\Include
	system_stm32f4xx.c stm32f4xx_conf.h	用于配置系统时钟	…\en.stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates

续表

工程文件夹	复制文件	文件说明	固件库的位置
CMSIS	include	内核相关的固件库	…\en. stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\CMSIS\Include
Driver	inc 文件夹	外设.h 文件	…\en. stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\STM32F4xx_StdPeriph_Driver\inc
	src 文件夹	外设对应的.c 文件	…\en. stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Libraries\STM32F4xx_StdPeriph_Driver\src
USR	stm32f4xx_it.c stm32f4xx_it.h main.c main.h	用户编写的程序和中断服务函数	…\en. stm32f4_dsp_stdperiph_lib\STM32F4xx_DSP_StdPeriph_Lib_V1.8.0\Project\STM32F4xx_StdPeriph_Templates

从表 5.1 中可以看出，这里主要将 libraries、project 文件夹的文件添加到工程文件夹中。自此，固件库的文件已经添加完成。下面将这些文件添加到工程中。

3. 创建文件

虽然将固件库中的 main.c 添加到了 USR 中，但需要重写里面的内容，main.c 代码如下：

```
#include "stm32f4xx.h"
#include "./led/led.h"
#include "main.h"

void TimingDelay_Decrement(void)
{
    int i,j;
    for(i=0;i<5;i++)
    {
        j=5000000;
        while (j>=0 ) j--;
    }
}

int main()
{
    LED_Config();
    while(1)
    {
        GPIO_SetBits(LED1_GPIO, LED1_PIN);
        TimingDelay_Decrement( );
        GPIO_ResetBits(LED1_GPIO, LED1_PIN);
```

```
            TimingDelay_Decrement( );
        }
    }
```

LED灯是外设,我们需要在USR文件夹下创建文件夹led,然后选择File→New,创建一个文件,这时就会在编辑窗口出现一个名为Text1的编辑窗口,如图5.4所示。

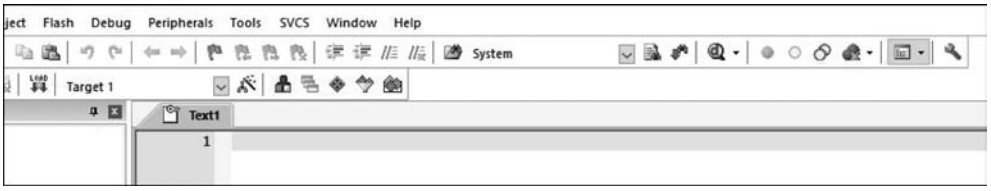

图5.4　新建文件窗口

在此编辑窗口,写入代码:

```
#include "stm32f4xx.h"

/*引脚定义*/

/*LED*/
#define LED1_PIN GPIO_Pin_12
#define LED1_GPIO GPIOB
#define LED1_CLK RCC_AHB1Periph_GPIOB

/*初始化函数说明*/
void LED_Config(void);
```

输入完代码后,单击“保存”按钮,这时会弹出保存文件对话框,请选择led文件夹,输入文件名(文件名需要带扩展名)。由于这里建立的是库文件,所以起名为led.h。

依此方法建立led.c文件,并保存在led文件夹中,该文件代码为:

```
/*LED*/
#include "./led/led.h"

void LED_Config()
{
GPIO_InitTypeDef  GPIO_InitStructure;

RCC_AHB1PeriphClockCmd(LED1_CLK, ENABLE);

    GPIO_InitStructure.GPIO_Pin = LED1_PIN ;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_High_Speed;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

至此，文件创建完成。

4. 添加文件到工程

右击 Target1，选择 Manage Project Items 按钮，然后单击 Groups 下的 New(Insert) 按钮，在工程中添加 5 个组，并依次将它们重命名为 INIT、CORE、CMSIS、Driver、USR，这个 Groups 名可以与建立的文件夹名不一样，如图 5.5 所示。

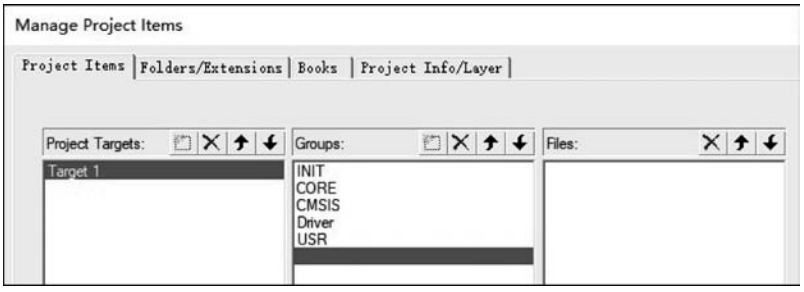

图 5.5　添加工程文件夹

新建文件夹后，分别在对应的文件夹里添加本次实验所需的.c、.h 或者.s 文件。

单击 INIT 组，单击 Add Files 按钮添加文件，添加文件夹 ex5-LED\INIT 下的.s 启动文件，注意，默认类型是.c 文件，将类型变为.s 就可以看到 INIT 文件夹下的启动文件，如图 5.6 所示。

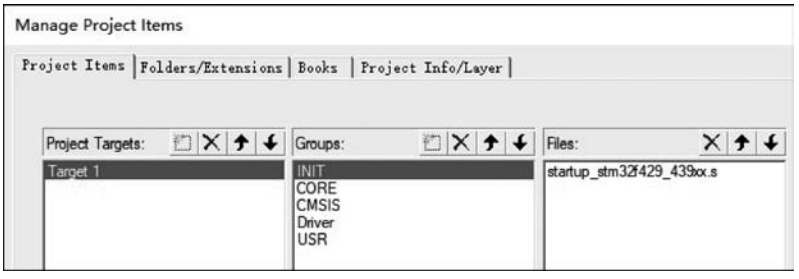

图 5.6　在 INIT 组添加启动文件

单击 CORE 组，单击 Add Files 按钮添加文件，添加文件夹 ex5-LED\CORE 中的一个.c 文件，如图 5.7 所示。

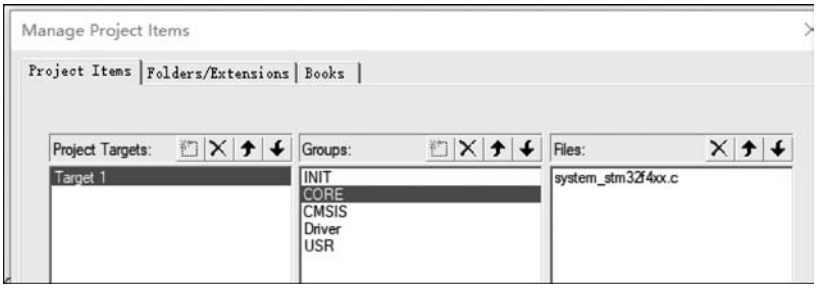

图 5.7　在 CORE 组添加文件

单击 Driver 组，单击 Add Files 按钮添加文件，添加文件夹 ex5-LED\Driver\src 中的所

有.c 文件,如图 5.8 所示。

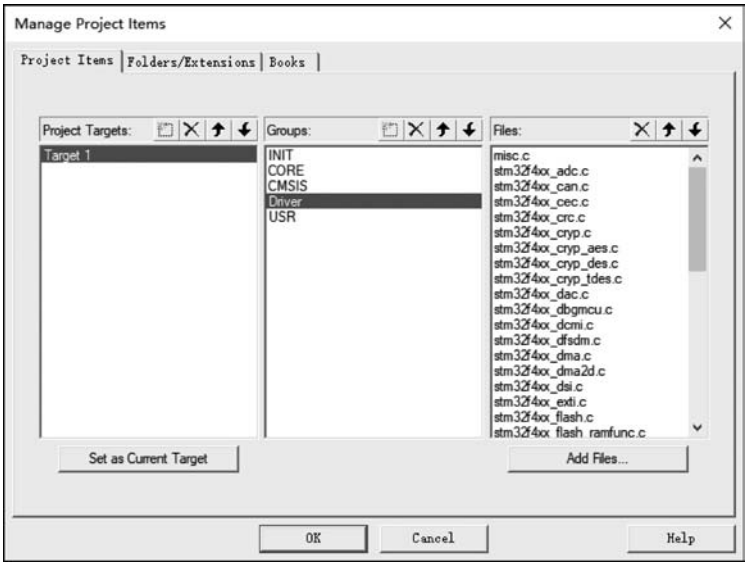

图 5.8　在 Driver 组添加文件

单击 USR 组,单击 Add Files 按钮添加文件,添加文件夹 ex5-LED\USR 中的 2 个.c 文件,以及 ex5-LED\USR\led 中的 led.c 文件,如图 5.9 所示。

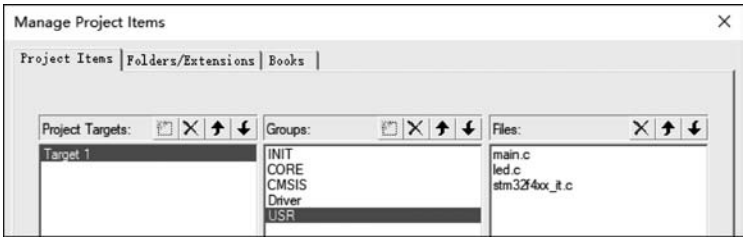

图 5.9　在 USR 组添加文件

至此,全部文件添加完毕,单击 OK 按钮。

大家可能注意到了,CMSIS 文件夹中没有任何文件添加到 CMSIS 组,这是由于这个文件夹都是内核的.h 头文件,没有.c 文件。

5. 配置参数

配置参数的方法与实验 4 完全一样,可以参照实验 4 进行配置,唯一不同的是包含路径需要根据本实验进行设置,如图 5.10 所示。

6. 点亮 LED 灯

单击 按钮编译代码,成功后,单击 按钮将程序下载到开发板,程序下载后,Build Output 选项卡中如果出现 Application running…,则表示程序下载成功;如果没有出现,按复位键试试。如果一切顺利,可以看到 LED 灯按照 main 函数中定义的那样一闪一闪发光,如图 4.17 所示。

7. 注意事项

由于种种复杂原因,在开发过程中总会遇到各种各样的问题,保持一份耐心与毅力坚持学

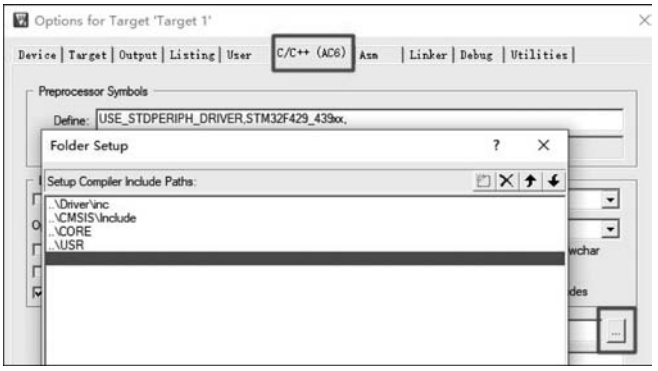

图 5.10　修改包含路径

下去，总会有收获。这里仅说明一下在开发过程中经常遇到的问题及一些解决办法和建议。

（1）在新建一个工程时，工程名一般取英文名，以便于识别。若含有汉字则可能会出现乱码。

（2）在建立工程模板时，由于添加文件比较多且烦琐，所以应保持耐心。分清具体哪一个文件夹下存放哪些文件，又具体为.c、.h、.s 哪一种类型的文件。应按照上面所讲，一步步操作，不要出错。

（3）若芯片是 STM32F429-39xx，则要在 Files 中寻找 stm32f4xx_fsmc.c 文件，单击右上角红色的叉号按钮进行删除；若芯片是 STM32F407-41xx，也要在 Files 中寻找 stm32f4xx_fmc.c 文件，单击右上角红色的叉号按钮进行删除。FMC 文件实现的功能和 FSMC 一样，根据不同的芯片选取一个就可以了。

（4）这里强调最重要的一点，就是添加 C/C++下的宏定义与.h 文件的路径问题。首先，在添加宏定义的时候，一定要弄清是要把“STM32F40_41xxx，USE_STDPERIPH_DRIVER”（针对 STM32F407 系列）或“USE_STDPERIPH_DRIVER，STM32F429_439xx”（针对 STM32F429 系列）这个宏定义添加进去。要记住一个字母，一个符号都不能错，否则会导致程序当中很多函数、定义无法识别，最终导致编译不成功；其次，在添加.h 文件的路径时，一定要把文件夹添加到你使用的那个.h 文件所在的那一层文件夹，切忌出错，否则必然导致所引用的.h 文件找不到出处，造成许多错误。

5.7　实验参考程序

5.7.1　led 文件夹

1. led.h

```
#include "stm32f4xx.h"

/*引脚定义*/
```

```
/* 绿色 LED 灯 */
#define LED1_PIN GPIO_Pin_12
#define LED1_GPIO GPIOB
#define LED1_CLK RCC_AHB1Periph_GPIOB

/* 初始化函数的说明 */
void LED_Config(void);
```

2. led.c

```
#include "./led/led.h"

void LED_Config()
{
    /* 定义一个 GPIO_InitTypeDef 类型的变量 */
    GPIO_InitTypeDef  GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(LED1_CLK, ENABLE);
    /* 设置控制的引脚号 */
    GPIO_InitStructure.GPIO_Pin = LED1_PIN ;
    /* 设置该引脚为输出类型 */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    /* 设置其类型为推挽模式 */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    /* 设置其引脚速度 */
    GPIO_InitStructure.GPIO_Speed = GPIO_High_Speed;
    /* 定义一个 GPIO_InitTypeDef 类型的变量 */
    GPIO_Init (LED1_GPIO, &GPIO_InitStructure) ;

}
```

5.7.2　main.c

```
#include "stm32f4xx.h"
#include "./led/led.h"
#include "main.h"

void TimingDelay_Decrement(void)
{
    int i,j;
    for(i=0;i<5;i++)
    {
        j=5000000;
        while (j>=0) j--;
    }
}

int main()
```

```
{
    LED_Config();
    while(1)
    {
        GPIO_SetBits(LED1_GPIO, LED1_PIN);
        TimingDelay_Decrement();
        GPIO_ResetBits(LED1_GPIO, LED1_PIN);
        TimingDelay_Decrement();
    }
}
```

5.8 实验总结

本实验利用固件库完成 LED 灯的点亮，该工程项目也是后面项目的工程模板，后面的实验将利用此工程模板完成实验。

5.9 思考题

(1) 关于 STM32 的寄存器开发与库开发各有什么优缺点？我们应该如何合理地选用开发方式，从而达到想要的效果？

(2) 完成连接到 GPIOB13、GPIOB14 上的 LED 灯的亮灯灭灯实验，并体会和实验 4 的不同和相同之处。