嵌入式系统实验平台
STM32-DMCK

实验指导书

2023.2.24

目录

实验一 单个 LED 闪烁实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，和其他相关的操作。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 熟悉 STM32 单片机的 GPIO 口的输出控制。

4 熟悉 JLINK 仿真下载步骤。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

实现实验平台核心板上的指示灯 LD2（PC13）延时闪烁的功能

4.实验分析

PC13 为核心板上 LD2 的接口。将 PC13 配置为 GPIO 输出模式（Out_pp）,对 PC13 延时配置高或者低。即可实现闪烁的功能。通过软件延时语句实现最基本的闪烁功能，延时的长短决定闪烁的快慢。

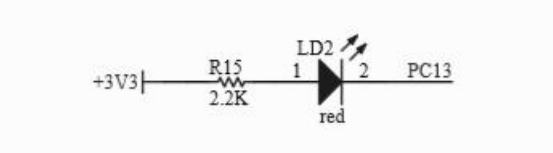

图 5.1.1 LD2 小灯电路图

时钟是单片机运行的基础，时钟信号推动单片机内各个部分执行相应的指令。时钟系统就是 CPU 的脉搏，决定 CPU 速率，像人的心跳一样只有有了心跳，人才能做其他的事情，而单片机有了时钟，才能够运行执行指令，才能够做其他的处理（点灯，串口，ADC），时钟的重要性不言而喻。STM32F4 的时钟配置较为复杂，但是，为了方便时钟的配置，STM32CubeMX 界面有专门的时钟树可以对时钟进行配置。在实验平台上，STM32 最小系统外焊 25M 晶振，经过内部分频后，系统时钟可配置到 168MHZ。时钟树工作原理，请参考 STM32F407 编程手册的时钟数章节。

5.实验步骤

（1）通过 STM32CubeMX 完成对外设和时钟的配置。并生成工程文件

（2）打开生成的工程文件，在适当处添加自己的程序。

（3）打开实验平台，对工程文件进行仿真下载，查看实验平台上的 LD2 是否闪烁。

（4）多次实验，熟悉 STM32CubeMX 和 MDK 的使用。

6.实验例程参考

1） 通过 STM32CubeMX 建立工程文件。打开 STM32CubeMX 软件，点击 File->New Preject，如下图所示。

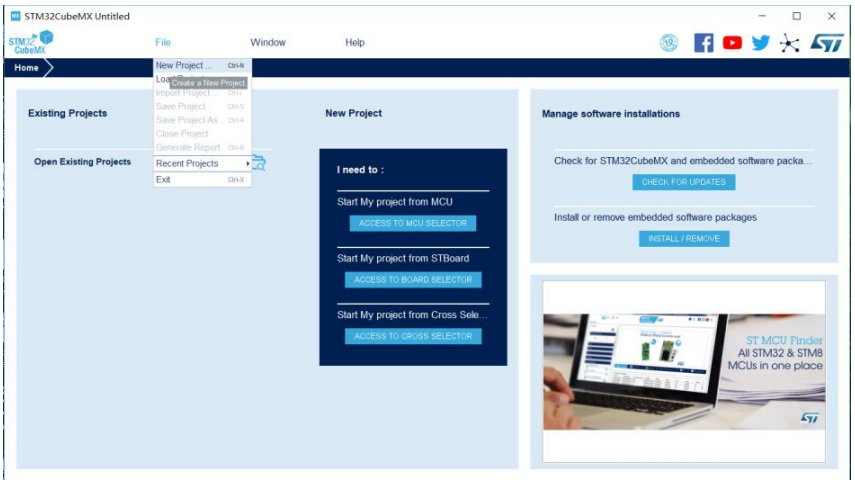

图 5.1.2 新建工程

2） 选择 MCU：STM322F407ZGT6,创建工程，如下图所示。

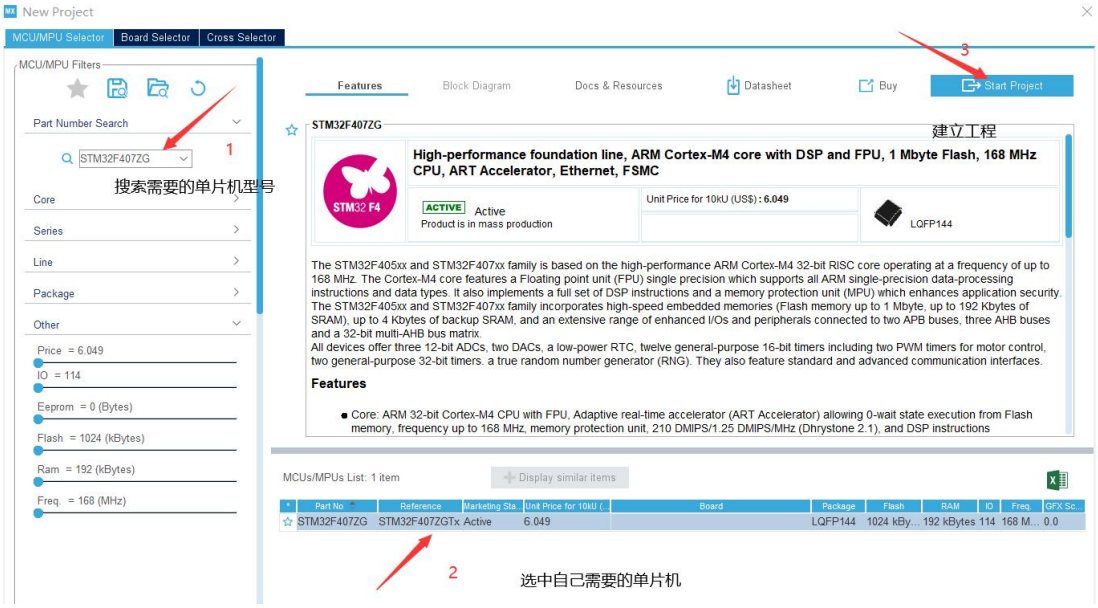

图 5.1.3 新建工程

3） 进入工程后，选中我们需要控制的引脚 PC13，并将该引脚设置为输出模式，如下图所示。

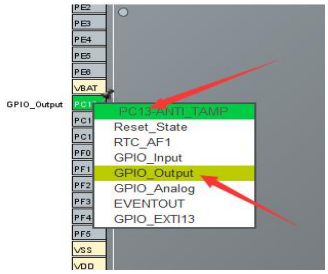

图 5.1.4 引脚配置

4） 点击System Core中的GPIO选项，PC13引脚配置如下图所示。

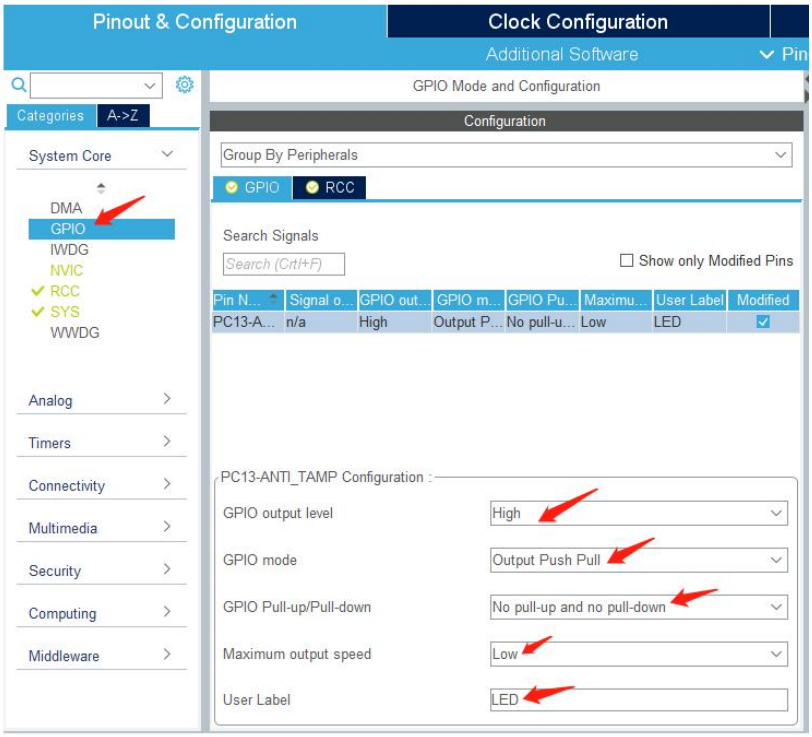

图 5.1.5 引脚配置

5） 时钟的 RCC 配置。如果没有配置那么默认时钟是使用内部 RC 振荡器（HSI）。因为内部时钟精度不高，这里选用外部时钟。如下图所示。

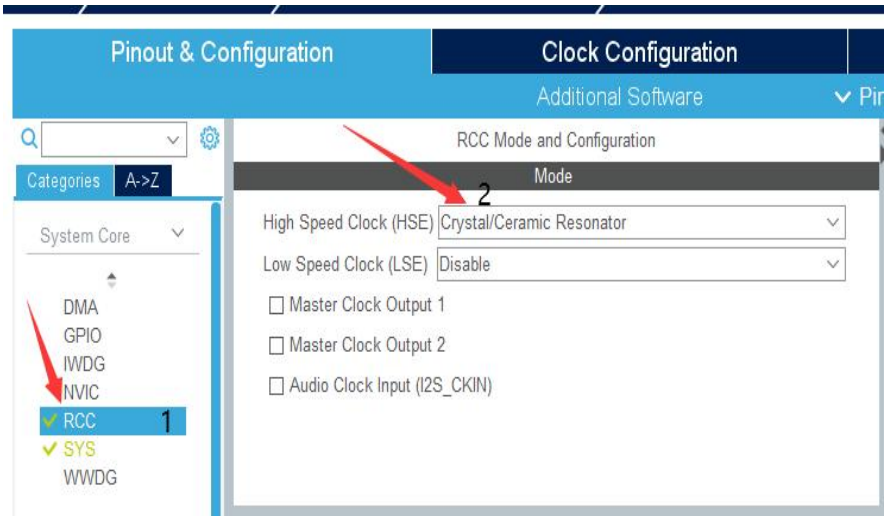

图 5.1.6 时钟配置

6） 配置时钟树。在 HCLK（MHz）对应的方框中，可以在里面输入你想要的时钟大小，点击回车即可自动的配置好。不用自己在慢慢的去调，如果在时钟配置中出现了深红色即说明配置出现错误，该路不可能出现这种情况下的时钟。这里我们将 HCLK(MHz)设置为最大值 168MHz，核心板外部晶振为 25MHz，如下图所示。

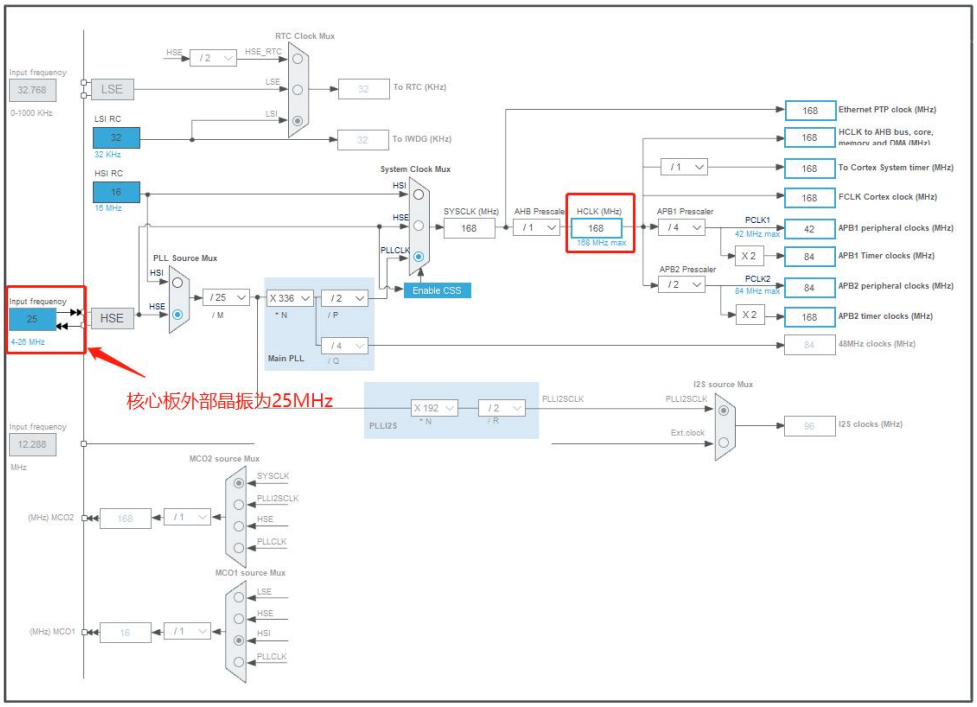

图 5.1.7 时钟配置

7） 外设与时钟配置完成后，可以开始建立工程。选择 Project Manager 选项，如图 5.1.8 中，红色标号为 1 处表示为建立工程的名字（此处不能出现任何的中文字符）。红色标号为 2 处表示工程文件保存的地方（此处不能出现有中文字符），红色标号为 3 处表示使用的编译器，这里选择 MDK-ARM V5.其他地方保持默认即可。

提示：文件名，文件路径，用户可自行修改。注意，避开中文。

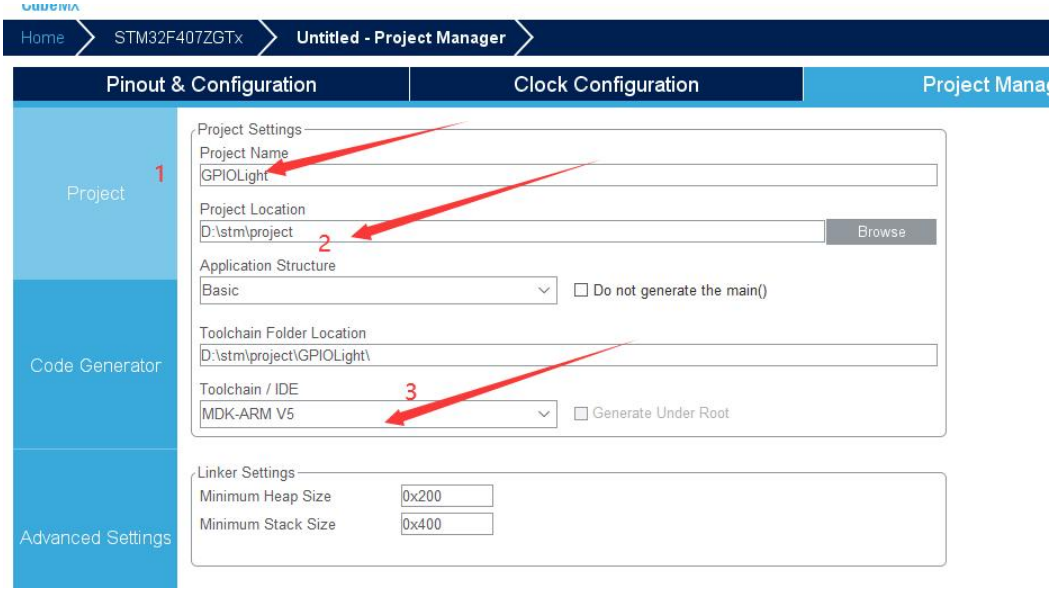

图 5.1.8 工程文件配置

8） 点击“GENERATE CODE”，生成工程文件。（此处可以不设置，使用者可以根据自己的要求进行设置）

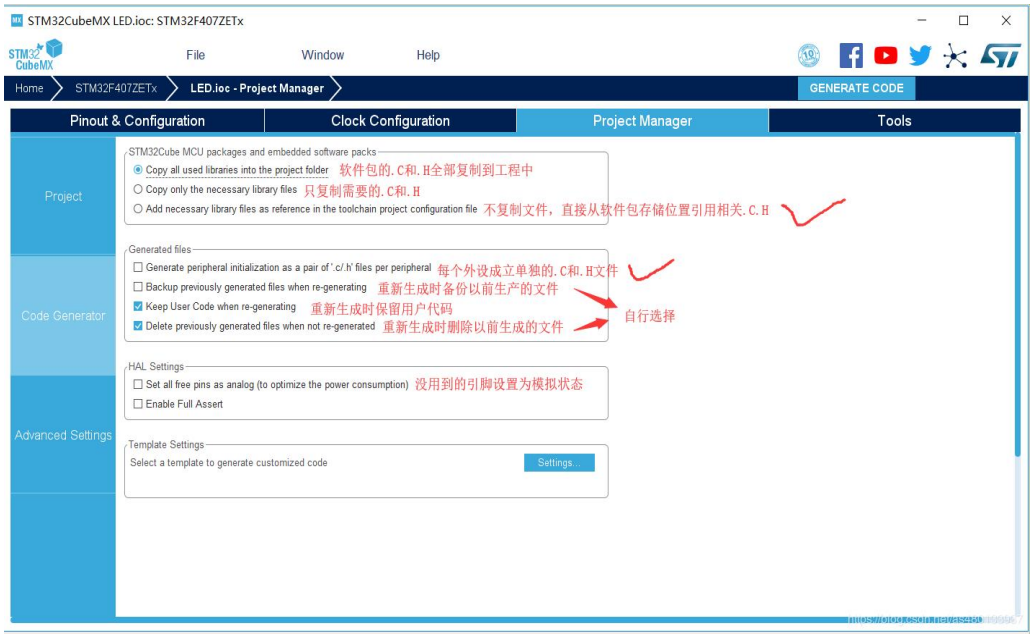

图 5.1.9 工程文件配置

9） 生成成功后，有如图 5.1.10 所示。新创建工程时，可以选择打开工程文件。若已打开生成过的工程文件，可以选择 CLOSE。

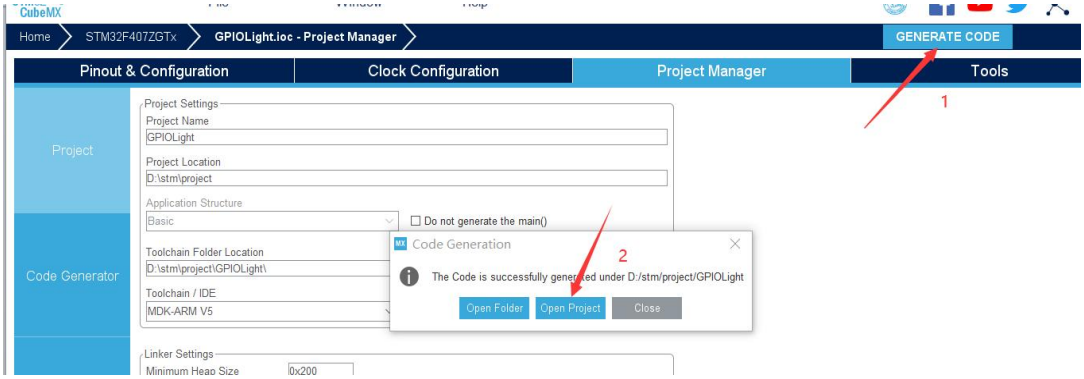

图 5.1.10 生成工程

10）进入工程后进行仿真调试工程的相关设置，如图 5.1.11 与 5.1.12 所示。

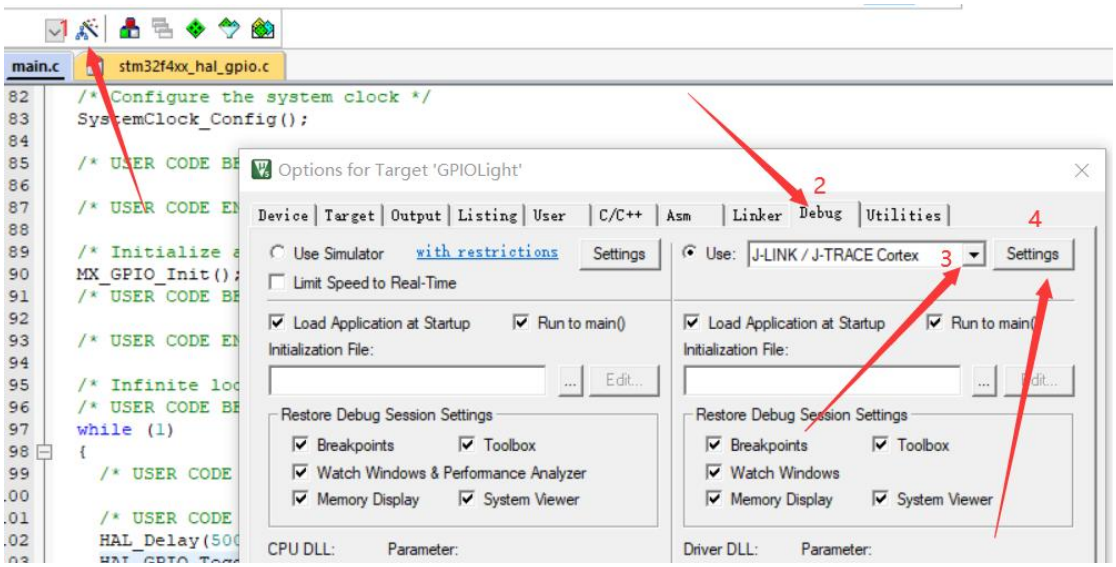

图 5.1.11 下载编译设置

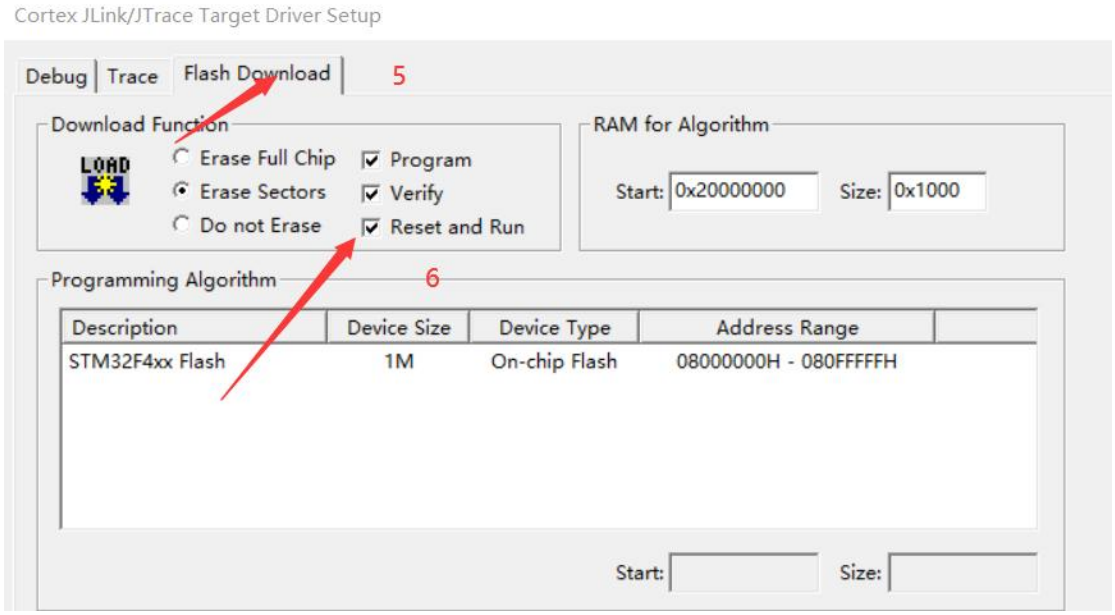

图 5.1.12 下载编译设置

11）STM32CubeMX 在配置完成后，将配置好的引脚在程序里面已经设置好了。我们只需要添加应用代码程序进去即可。在 man.c 主程序的 while（1）中，添加两行代码。

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  HAL_Delay(1000);      //延时
  HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);     //引脚电平翻转
}
/* USER CODE END 3 */
```

图 5.1.13 添加功能代码

编程提示：用户代码必须添加在“/* USER CODE BEGIN...*/”至“/* USER CODE END... */” 之间。这样 CUBEMX 工程文件可以重复配置，并生成相关代码，而

不影响用户原来写好的代码。

12) 上述在主程序中添加的函数对 GPIO 口引脚操作的库函数不只这一个，使用者可以打开箭头所指的函数，里面有许多对引脚操作的函数，使用者也可以用其他函数来操作引脚，并不一定只使用这个函数。

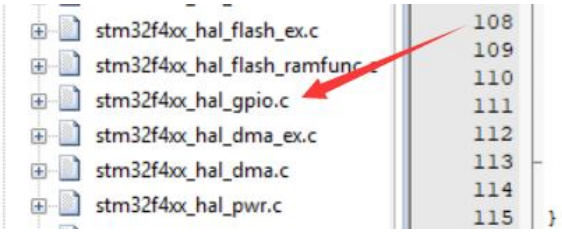

图 5.2.9 GPIO 外设库文件

实验二 LED 流水灯实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，和其他相关的操作。

2 在熟练使用 MDK 的过程中、熟悉实验平台的仿真和下载方法。

3 掌握 STM32 单片机的 GPIO 口的输出控制。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

实现实验平台底板上的指示灯 LD1-LD4（TFT LCD 正下方）四个指示灯流水点亮的功能。

4.实验分析

主控板上配有 4LED 指示灯。 PF0-PF3 为底板上 LD1-LD4 的接口。将 4 个接口配置为推挽输出模式（Out_pp）,对 PF0-PF3 延时配置高或者低，即可实现流水点亮的功能。通过软件延时语句实现最基本的闪烁功能，延时的长短决定闪烁的快慢。

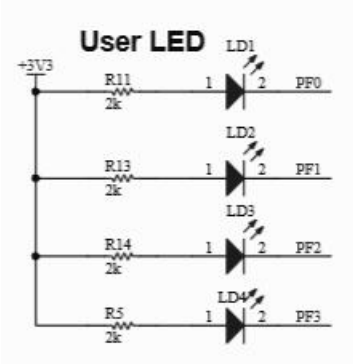

图 5.2.1 4 个小灯电路图

5.实验步骤

（1）根据上一节学习的实验内容，建立一个新的工程。并且配置好相关参数。

（2）生成工程文件并打，在相应的位置添加程序。

（3）打开实验平台，对工程文件进行仿真下载，查看指示灯的流水效果。

6.实验例程

13）按照实验一的方法，打开 STM32CubeMX，并创建一个新的工程，选择好使用的主芯片。开始配置实验中需要的引脚，将 PF0-PF3 全部设置为 GPIO 输出模式，如图 5.2.2。

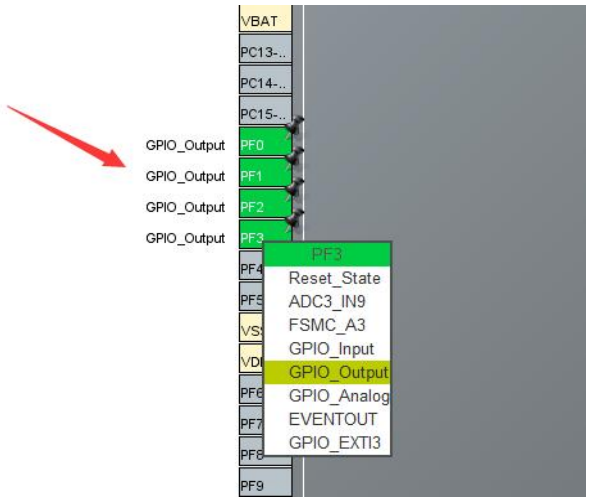

图 5.2.2 引脚配置

14）指示灯引脚配置完成后，还需要配置端口的工作模式。如图 5.2.3 所示，选中 GPIO 弹出上一步选中的 4 个引脚，选中每一个引脚，然后再下方配置其初始化的状态，如图 5.2.3.

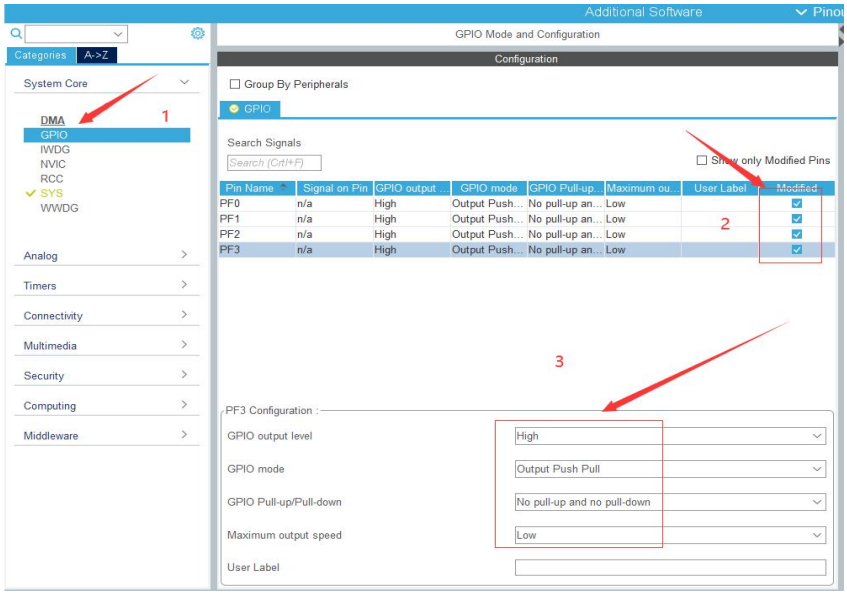

图 5.2.3 引脚配置

15）将四个LED的Label分别改成LED1、LED2、LED3、LED4，如下图所示。

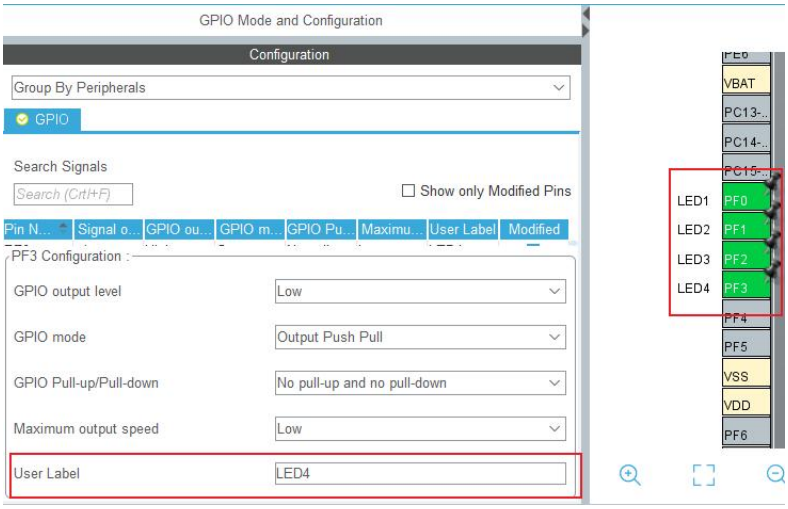

图 5.2.4 更改 Label

16）引脚配置完成后就应该对时钟进行相应的操作配置，时钟配置参考实验一。

17）配置工程文件，并生成相关代码。参考图如图 5.2.5，注意文件名及路径自行修改，且不能有中文字符，参考实验一。

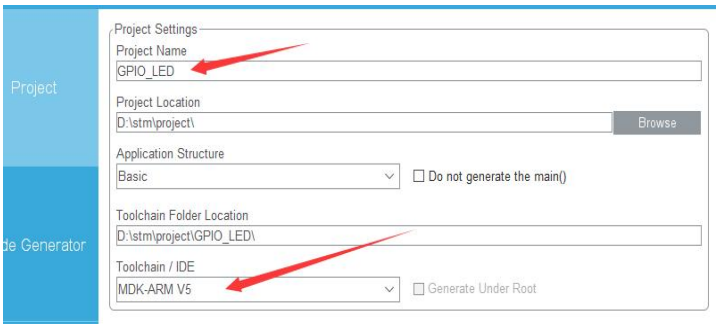

图 5.2.5 工程文件设置

18）打开生成的工程文件，配置仿真调试器，如下图 5.2.6 和图 5.2.7 所示，亦可参考实验一。

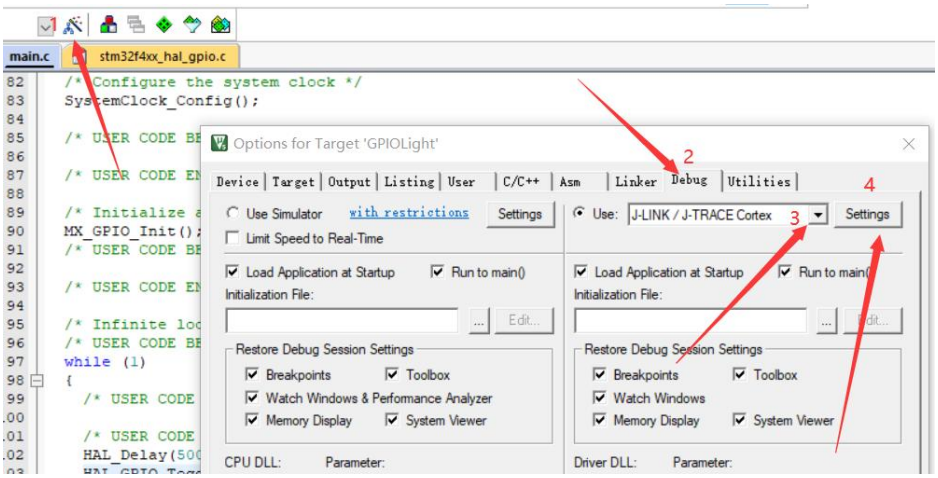

图 5.2.6 下载编译设置

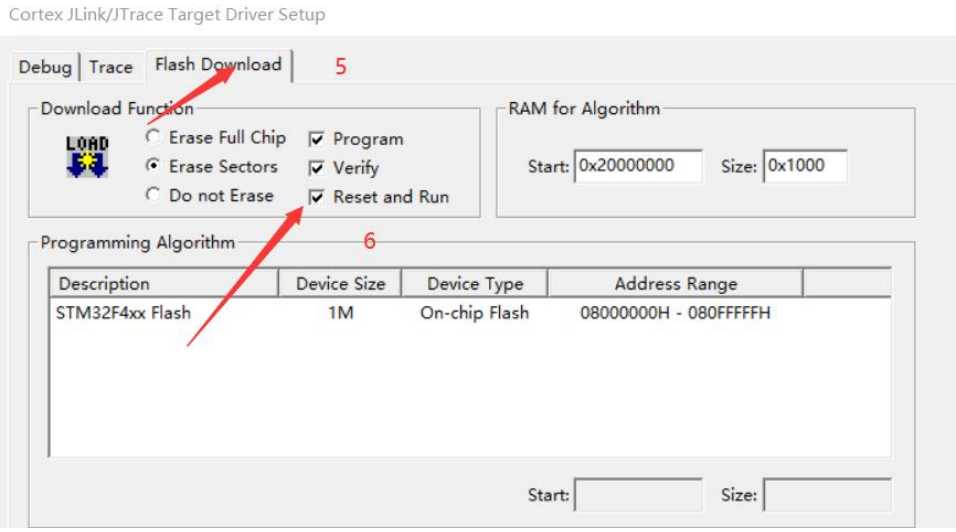

图 5.2.7 下载编译设置

19）在主程序中添加自己代码（请同学们自行完成），编译下载，观察 4 个小灯的工作状态。

注意：用户代码必须添加在“/* USER CODE BEGIN...*/”至“/* USER CODE END... */” 之间。这样 CUBEMX 工程文件可以重复配置，并生成相关代码，而不影响用户原来写好的代码。

实验三 按键控制 LED 实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置和其他相关的操作。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 掌握 STM32 单片机的 GPIO 口的输入输出功能。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

利用实验平台的 LD1-LD4 和 4 和按键（皆在 LCD 屏幕下方），试编程实现 LETF、RIGHT、UP、DOWN、按键分别对应的 LD1、LD2、LD3、LD4 指示灯亮，其他指示灯灭。

4.实验分析

实验分析 PF0-PF3 为核心板上 LD1-LD4 的接口。将 4 个接口配置为推挽输出模式(Out_pp)

LETF、RIGHT、UP、DOWN、按键分别占 PG14、PG13、PG15、PG12 按键为低电平有效（即按键按下时引脚电平状态为低电平）按键电路图和指示灯电路图如下。

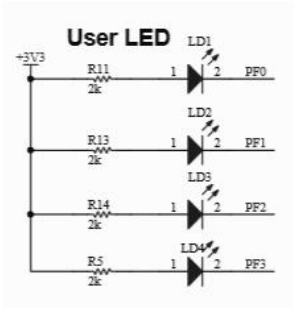

图 5.3.1 指示灯电路图

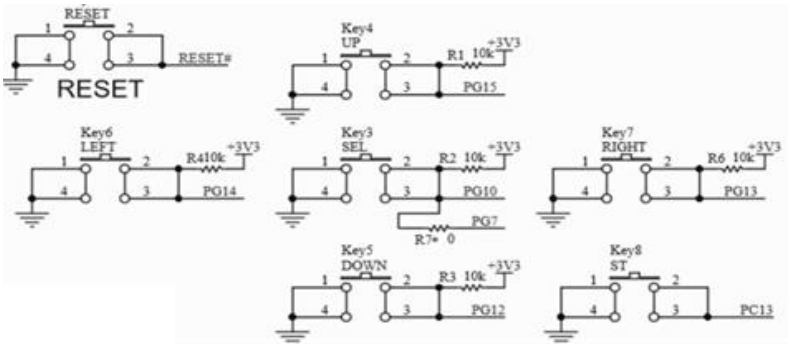

图 5.3.2 按键电路图

5.实验步骤

（1）根据上 2 节学习的实验内容，建立一个新的工程。并且配置好相关参数。

（2）生成初始 c 代码，打开工程文件，在相应的位置添加属于自己的程序。

（3）打开实验平台，对工程文件进行仿真下载，查看运行结果。

6.实验例程

1） 打开 STM32CubeMX 建立一个新的工程，选择要使用的芯片型号（参考实验一）。 开始配置实验中需要的引脚，将 PF0-PF3 全部设置为 GPIO 输出模式， 并更改 Label，如下图所示。

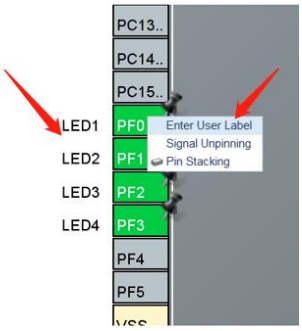

图 5.3.3 配置 GPIO 输出

2） PG12-PG15 配置为 GPIO 输入模式，并更改 Label，如下图所示。

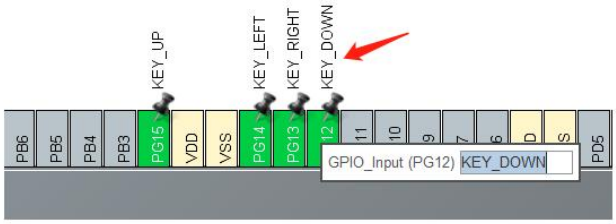

图 5.3.4 配置 GPIO 输入

3） 配置引脚。

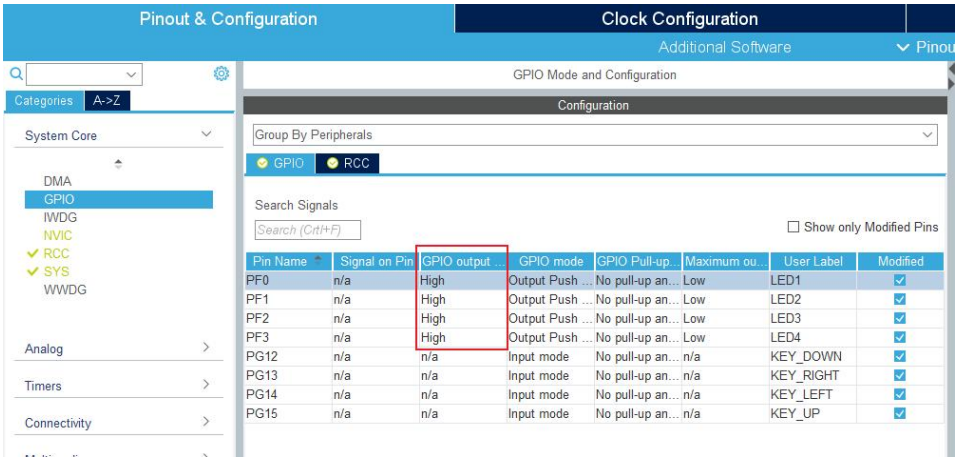

图 5.3.5 引脚配置

4） 引脚配置完成后就应该对时钟进行相应的操作配置，时钟配置参考实验一。

5） 配置工程文件，并生成相关代码，参考实验一。

6） 打开生成的工程文件，配置仿真下载器，参考实验一。

7） 添加用户代码，在主程序 while（1）中添加如下程序段，编译下载，操作四个按键，观察 4 个小灯的工作状态。

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if(HAL_GPIO_ReadPin(KEY_LEFT_GPIO_Port,KEY_LEFT_Pin)==GPIO_PIN_RESET)
    {
      HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,GPIO_PIN_RESET);
      HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,GPIO_PIN_SET);
    }
    else if(HAL_GPIO_ReadPin(KEY_RIGHT_GPIO_Port,KEY_RIGHT_Pin)==GPIO_PIN_RESET)
    {
      HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,GPIO_PIN_RESET);
      HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,GPIO_PIN_SET);
    }
    else if(HAL_GPIO_ReadPin(KEY_UP_GPIO_Port,KEY_UP_Pin)==GPIO_PIN_RESET)
    {
      HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,GPIO_PIN_RESET);
      HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,GPIO_PIN_SET);
    }
    else if(HAL_GPIO_ReadPin(KEY_DOWN_GPIO_Port,KEY_DOWN_Pin)==GPIO_PIN_RESET)
    {
      HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,GPIO_PIN_RESET);
    }
    else
    {
      HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,GPIO_PIN_SET);
    }
  }

  /* USER CODE END 3 */
}
```

图 5.3.6 添加代码

编程提示： HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)函数主要用来读取单片机引脚的状态；HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinStatePinState)函数主要用来写引脚的状态。

7.拓展实验

在上述例程中，我们使用了 HAL_GPIO_ReadPin 函数读取按键引脚电平，这样读取到的是按键瞬时的引脚电平，但实际上，普通按键按下和弹开都有抖动过程，抖动的时间一般为 5ms~10ms，如下图所示。

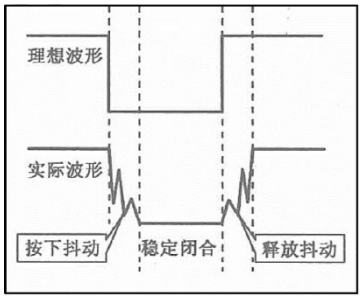

图 5.3.7 按键抖动

有了抖动，就需要消除抖动，不然很容易造成误操作。消抖的方法可以分为：硬件消抖和软件消抖。软件消抖就是通过程序控制实现消抖，一般有两种方法，一种是简单的延时读取（通常为 10ms 延时），另一种是比较复杂的按键状态机。硬件消抖就是通过硬件电路来消除抖动，如通过电容进行滤波，如下图中的 C46.

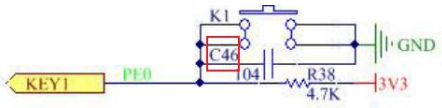

图 5.3.8 硬件消抖

请同学们采用延时的方法对按键进行消抖处理，完善上述例程。

编程提示： 可以将 HAL_GPIO_ReadPin 读取函数替换为自定义的按键扫描函数 KEY_StateRead，在该函数中进行软件消抖处理，最终确定按键状态。

实验四 外部中断实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，外部中断设置。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 熟悉 STM32 单片机的中断。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

利用实验平台 LD1 和 UP 按键（皆在 LCD 下方在），试用外部中断功能编程实现按键 UP 按键按下时 LD1 亮灭状态切换。

4.实验分析

LD1 由 PF0 控制输出，在 GPIO 初始化时，需要将这个引脚配置为 GPIO 输出模式。

UP 按键占用的是 PG15 引脚，按键为低电平有效（即按键按下时引脚电平为低电平）。按键电路图如图 5.3.2 所示。

PG15 外部中断占 EXTI15 口,具体外部中断原理请参考相关教材或手册。

5.实验步骤

（1）按照前几节的实验相关的内容，建立一个新的工程。并配置好相应的参数。

（2）添加相应的程序，实现本节实验功能

（3）打开是按平台，对工程文件进行编译仿真下载，查看运行结果。

（4）下载程序后，多次按 UP 按键，查看 LD1 的变化。

6.实验例程

1) 打开 STM32CubeMX 建立一个新的工程，选择使用的芯片型号,配置相应的引脚。

2) 将 PF0 引脚配置成输出模式。

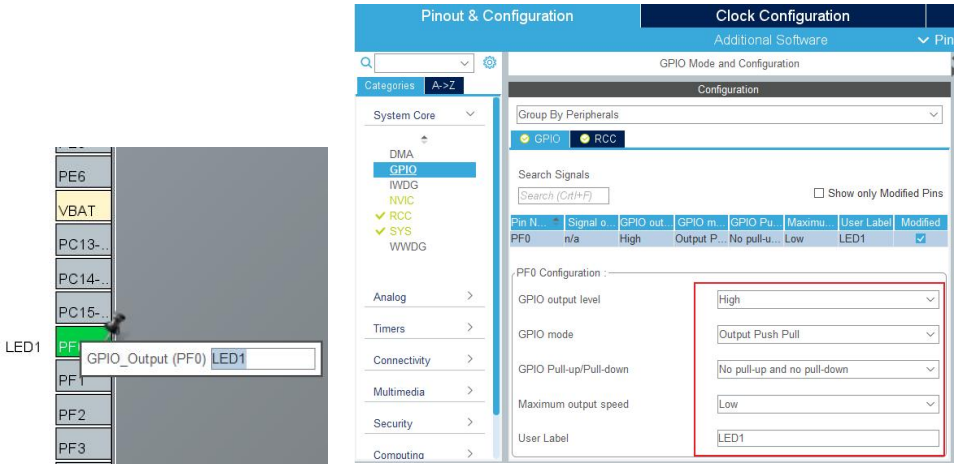

图 5.4.1 引脚配置

3） 配置中断，选中 PG15 号引脚将其设置为外部中断模式。

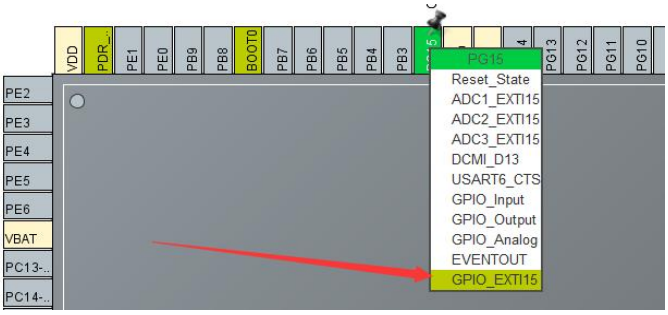

图 5.4.2 引中断配置

4） 中断触发模式选择，选择的是下降沿触发方式。

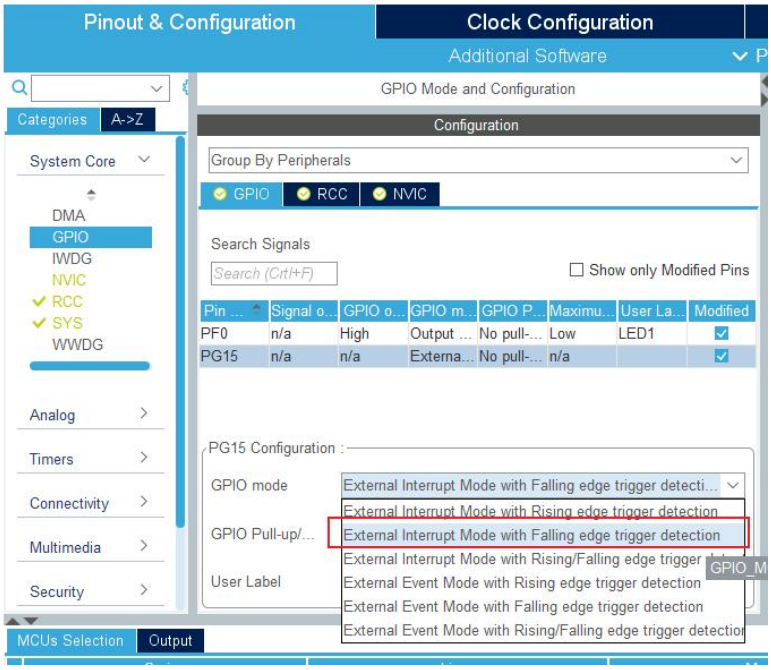

图 5.4.3 GPIO 中断配置

5) 配置完 GPIO 引脚，还应该使能中断，选择 System Core 下面的 NVIC 选项，如下图所示，将 EXTI line[15:0] 使能勾选，将抢占优先级设置为 1，优先级的编号越小，表明它的优先级别越高。

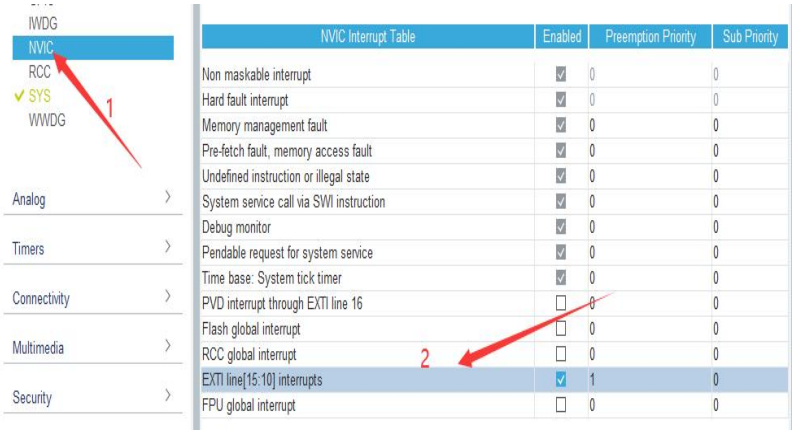

图 5.4.4 引中断配置

6) 引脚配置完成后就应该对时钟进行相应的操作配置，时钟配置参考实验一。

7) 配置工程文件，并生成相关代码，参考实验一。

8) 打开生成的工程文件，配置仿真下载器，参考实验一。

9) 在 Application/User 文件夹下可以看到 stm32f4xx_it.c 文件，如下图红色标记 1 处所示，其作用是用来编写中断服务函数。打开该文件，可以看到在程序里面已经将我们需要使用的中断函数写好了，如下图红色标记 2 处所示。

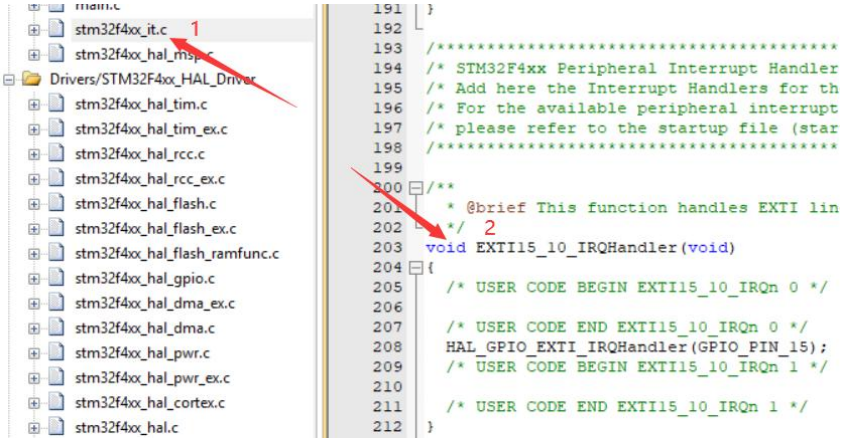

图 5.4.5 中断函数

10) 利用程序的跟踪功能，可以看到中断程序的定义。中断程序的内容和标准库函数配置差不多，就是清除中断标志位,之后进入中断回调函数。在 HAL 库中，中断运行结束后不会立刻退出，而是会先进入相对应的中断回调函数，处理该函数中的代码之后，才会退出中断。所以在 HAL 库中我们一般将中断需要处理代码放在中断回调函数中。

```
494   */
495  void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
496  {
497    /* EXTI line interrupt detected */
498    if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
499    {
500      __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
501      HAL_GPIO_EXTI_Callback(GPIO_Pin);
502    }
```

图 5.4.6 中断函数

11）中断回调函数为：HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)，此函数就是外部中断处理函数。该函数在 HALL 库定义中为弱回调函数，如下图所示。

```
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

图 5.4.7 弱定义的中断回调函数

12）可以在 main.c 函数中重新写该弱回调函数，在其中写入我们需要处理的代码，用按键控制小灯的亮灭，参考如下代码：

```
/* USER CODE BEGIN 4 */
  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
  {
    if(GPIO_Pin==KEY_UP_Pin) //KEY_UP PRESSED
    {
      HAL_Delay(10); //DELAY
      if(HAL_GPIO_ReadPin(KEY_UP_GPIO_Port,KEY_UP_Pin)==GPIO_PIN_RESET)
        {
          HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
        }
    }
  }
/* USER CODE END 4 */
```

图 5.4.8 参考代码

13）编译下载，按下 UP 观察小灯的状态。

7.拓展实验

上面例程中我们只用了一个按键和一个 LED，为了强化同学们对中断的理解，我们扩展为四个按键，通过中断控制四个 LED 小灯。功能与实验三功能相同，但是实现方法完全不一样。下面请同学们采用中断功能，编程实现 LETF、RIGHT、UP、DOWN、按键分别控制的 LD1、LD2、LD3、LD4 指示灯亮亮灭（按键与 LED 的引脚参考实验三）。

编程提示：可以给按键中断设置不同的抢占优先级与响应优先级，分析优先级的设置对实验结果的影响。

实验五 定时器定时应用实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，定时器的设置。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 熟悉 STM32 单片机的定时器及其编程。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

利用实验平台核心板 PC13 指示灯，实现定时器控制的 1s 闪烁功能。

4.实验分析

PC13 为核心板上 LD2 灯的接口，将 PC13 配置为 GPIO 输出模式，设置定时器功能，实现指示灯的闪烁，PC13 指示灯电路如下图所示。

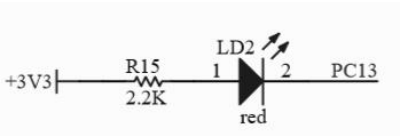

图 5.5.1 指示灯电路图

STM32F4 芯片具有 14 个定时器，含有 2 个 32 位高级定时器 TIM1 和 TIM8，10 个 16 位通用定时器，2 个基本定时器 TIM6 和 TIM7。

可以采用通过定时器 TIM2 来完成实验的定时功能，通用定时器包含了一个 16 位或 32 位的自动装载计数器，该计数器由可编程预分频驱动。他们可以通过多种用途，包括测量输入信号的脉冲宽度或者生成输出波形。使用定时器预分频和 RCC 时钟控制控制分频器，可将脉冲宽度和波形周期从几微妙调制到几毫秒。

5.实验步骤

（1）按照前几节的实验相关的内容，建立一个新的工程。并配置好相应的参数。

（2）添加相应的程序，实现本节实验功能

（3）打开实验平台，对工程文件进行编译仿真下载，查看运行结果。

6.实验例程

1） 新建CubeMX工程，参考前述实验。

2） 配置PC13端口输出（配置方法，参考前述实验）。

3） 配置时钟，参照前述实验。

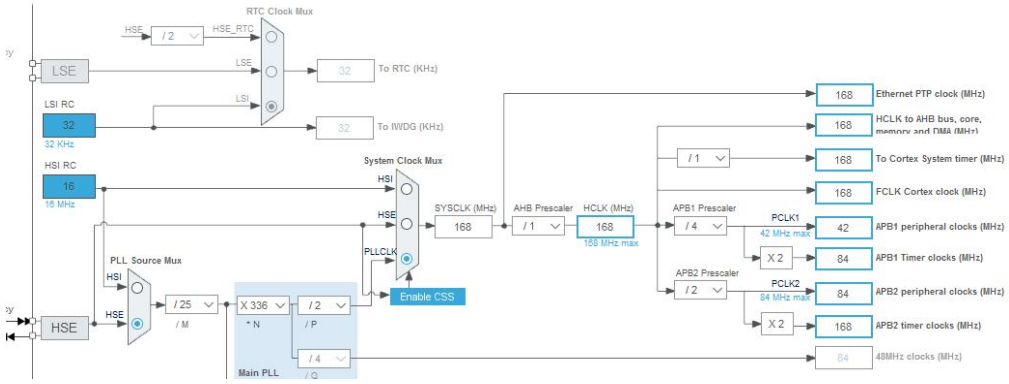

图 5.5.2 时钟配置

4） 根据STM32F407内部结构框图，TIM2定时器是挂在APB1桥上，APB1时钟频率为84M，如下图所示。

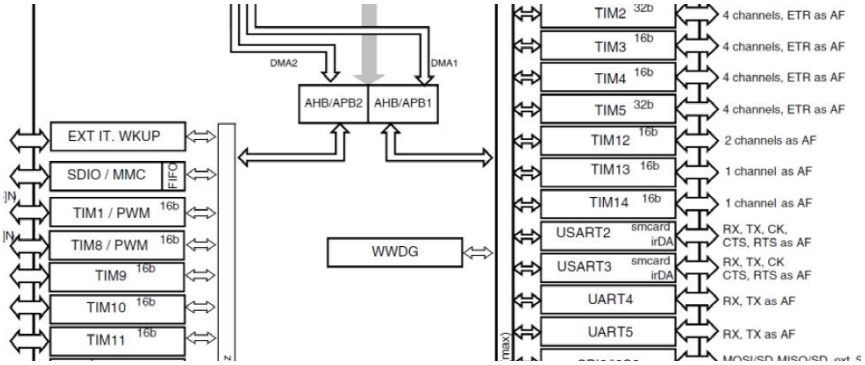

图 5.5.3 STM32F407 内部结构框图

5） 定时器配置，选择内部时钟源：84M，分频值为 8399，定时器周期 4999，得到的更新中断的时间为 500ms。如图 5.5.3 所示。

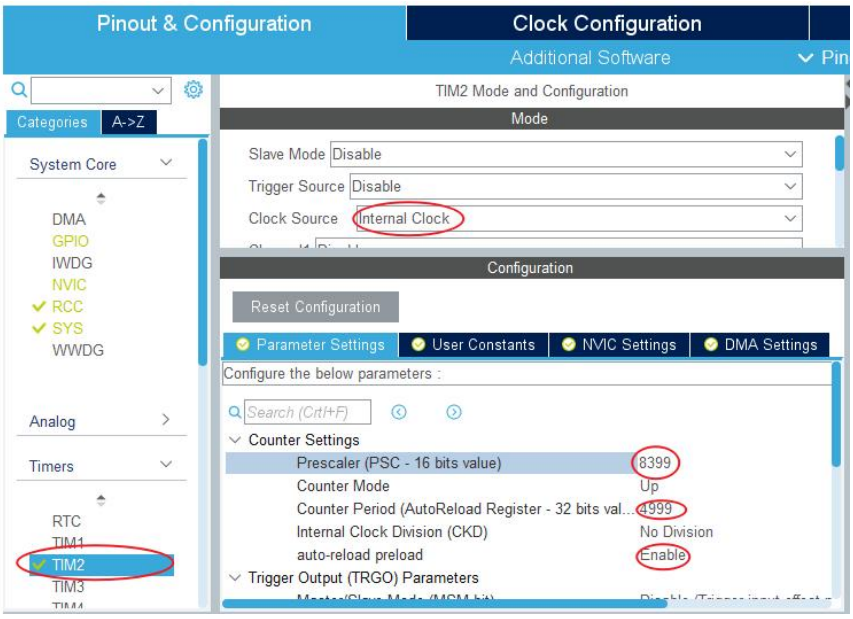

图 5.5.4 定时器参数配置

6） 开启定时器中断。

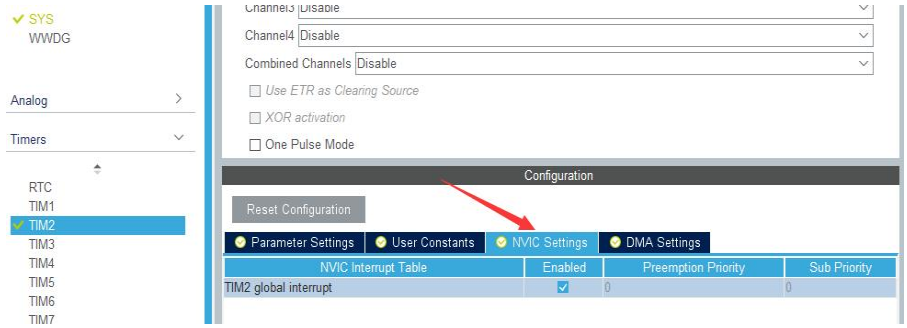

图 5.5.5 定时器设置

7） 配置工程文件，并生成相关代码，参考前述实验。

8） 打开生成的工程文件，配置仿真下载器，参考前述实验。

9） 打开生成的工程文件，在程序中添加代码，实现定时指示灯闪烁功能。

10）在主程序中添加如下代码，启动定时器开始计数。

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

图 5.5.6 启动定时器计数

11）写定时器中断回调函数，在此回调函数中，实现指示灯端口状态的控制功能，如：

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance==TIM2)
  {
    HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin);
  }
}

/* USER CODE END 4 */
```

图 5.5.7 写定时器回调函数

12）下载并执行程序，观察核心板指示灯的变化。

13）改变定时器的配置参数，使得 LED 的闪烁周期为 2s。

实验六 DAC 基本实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，DAC 的工作原理及设置方法。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 学会使用 STM32 的 DAC 功能。

2.实验设备

1 PC 机一台

2 实验平台一台

3 示波器一台

3. 实验内容

试编程使用 DAC 功能实现输出为三角波。

4.实验分析

STM32F407 处理的 DA 外设可通过 PA4（DAC0 通道）输出，也可以使用 PA5（DAC1 通道）输出。关于 STM32F407 外设原理，请详见芯片编程手册。

平台主控板中，CN1 接口引出可做此实验的 PA4 或 PA5 引脚，如图 5.6.1 所示。本次实验采用 PA5 输出三角波。

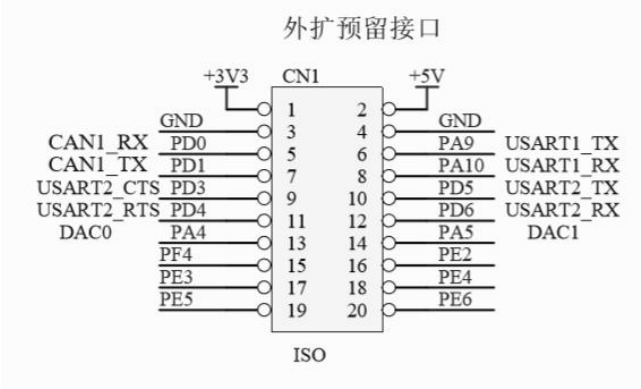

图 5.6.1 CN1 接口定义

5.实验步骤

（1）按照前几章所学的内容，建立一个新的工程。并配置好相关参数。

（2）添加相应的程序，实现本节实验功能

（3）打开实验平台，对工程文件进行编译仿真下载，查看运行结果。

6.实验例程

1) 新建CubeMX工程，参考前述实验。

2) 进入工程后按照本次实验的要求，第一步将需要输出的 PA5 引脚设置为模拟量输出（DAC_OUT）模式。

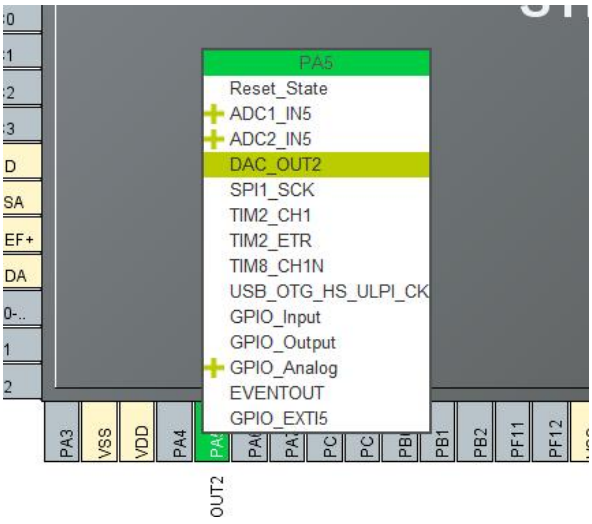

图 5.6.2 设置 DAC 输出引脚

3) PA5 管脚配置为 DAC 输出通道 2，下方的最大三角波(Maximum Triangle Amplitude)为 4095，DAC 配置触发(Trigger)为定时器 2 触发，波形生成模式(Wave generation mode)为三角波发生器(Triangle wave generation).最大三角波幅(Maximum Triangle Amplitude)为 4095，即为 3.3V。

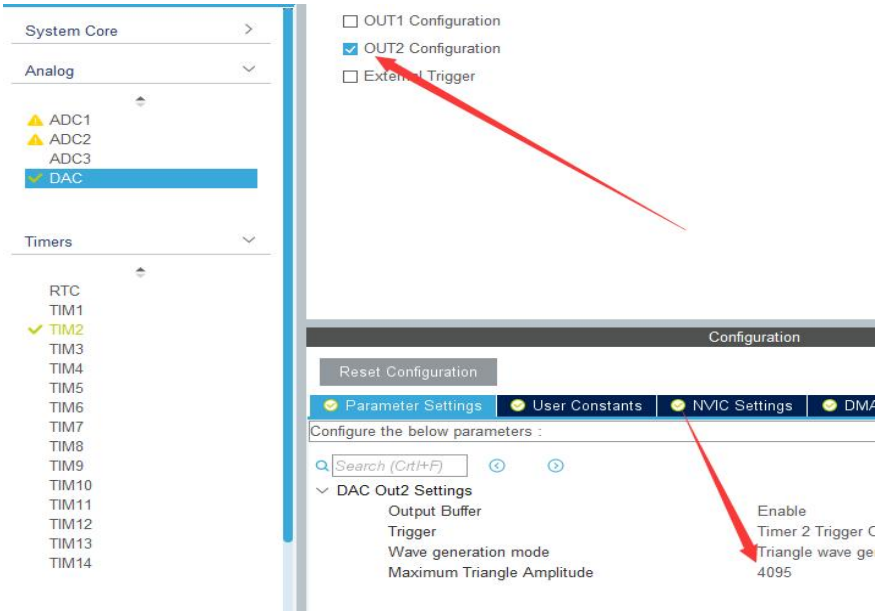

图 5.6.3 配置 DAC

4) 开启定时器2，并设置如下，触发事件TRGO(Trigger Event Selection TRGO)为更新事件。TRGO为触发信号，当定时器发送更新事件时，即发生溢出等事件时，定时器会发送触发信号TRGO到DAC,触发DAC转换输出模拟量。

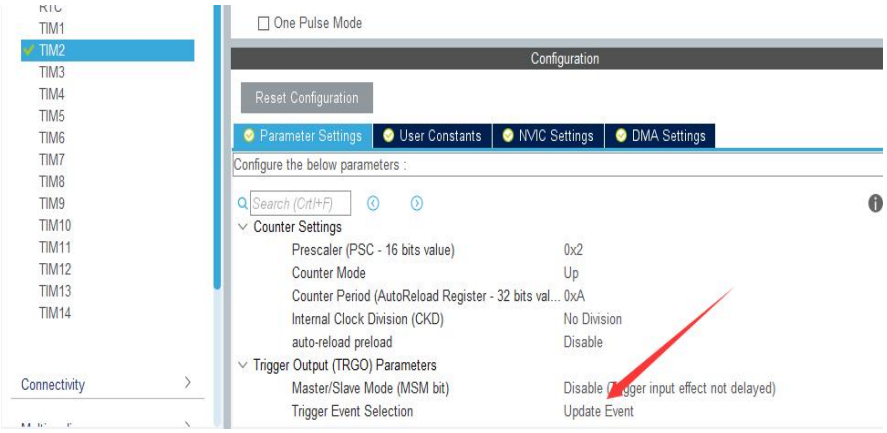

图 5.6.4 定时器配置

5) 配置时钟，参照前述实验。

6) 配置工程文件，并生成相关代码，参考前述实验。

7) 打开生成的工程文件，配置仿真下载器，参考前述实验。

8) 在主程序中添加如下代码：

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim2);
  HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

图 5.6.5 添加代码

9） 将示波器表笔与 DAC 输出引脚连接，连接方法如下图所示。正极探头勾在 PA5 引脚上；从 GND 引脚接一根杜邦线出来，将示波器负极夹在杜邦线的另一端（**切勿直接将鳄鱼夹夹在负极，以防短路！！！**）。

图 5.6.6 示波器探头连接

10）下载完成后查看示波器显示如下图所示：

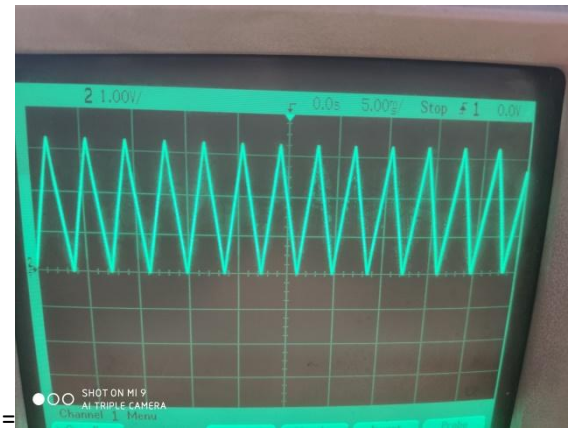

图 5.6.7 示波器显示

11）由三角波波形发生原理可知，三角波频率决定于定时器的预分频值和自动重载值的大小，调整TIM2的配置参数，使三角波的周期为200Hz。

7.拓展实验

以上例程使用了定时器触发生成三角波，请同学们自行尝试使用 DAC 无触发直接输出模式，实现按键控制的可调节电压的输出。具体实现的功能为：如果按下 KEY_UP 或者 KEY_DOWN 按键，可以实现对输出电压大小的调节（可使用万用表或者示波器观测输出）。电压的可调节范围是 0~3.3V。DAC 直接输出模式配置如下图所示。

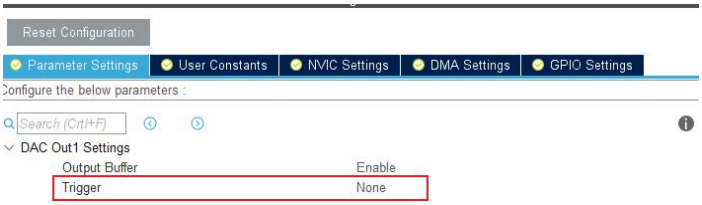

图 5.6.8 DAC 无触发配置

编程提示：使用 HAL_DAC_SetValue()设置 DAC 的输出，注意 8 位/12 位模式选择。

实验七 TFT 屏基本实验

1.实验目的

1 学会使用 2.8 寸 TFT 彩屏显示器编程。

2.实验设备

1　PC 机一台

2　实验平台一台

3.实验内容

试编程在 TFT 彩屏上显示系列字符串。根据屏的底层驱动程序，在屏上设计显示数据和字符。

4.实验分析

TFT 彩屏控制器支持 SPI、16 位并行接口。实验平台中默认采用 8 位并口方式。彩屏如图 5.7.1 所示。在该平台实验中未用到触摸功能，因此接口中 29-34 引脚并位使用。彩屏采用 ILI928（或 ILI935）驱动芯片驱动，具体驱动原理请参考相关资料手册。TFT 彩屏模块原理也请参考相资料。

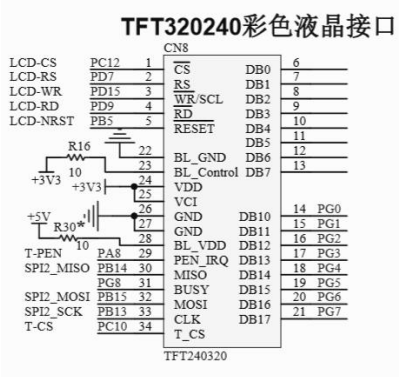

图 5.7.1 TFT 彩屏显示接口

5. 实验步骤

（1）按照前几次实验相关内容，建立一个新工程。

（2）在程序中添加现有的 LCD 底层驱动文件，并实现 TFT 初始化和显示的功能。

（3）打开实验平台，对工程文件进行仿真下载，查看运行结果。

6.实验例程

1） 新建CubeMX工程，参考前述实验。

2） 配置时钟，参考前述实验。

3） 配置工程文件，并生成相关代码，参考前述实验。

4） 找到工程生成的文件夹地址。下方红色标记1出是的本次实验的储存位置，在下方新建议一个文件夹，用于保存TFT LCD的底层驱动函数；红色标记2处为自己建立的文件夹，需要用来储存TFT LCD的底层驱动函数，文件夹文件名字可以随意取。

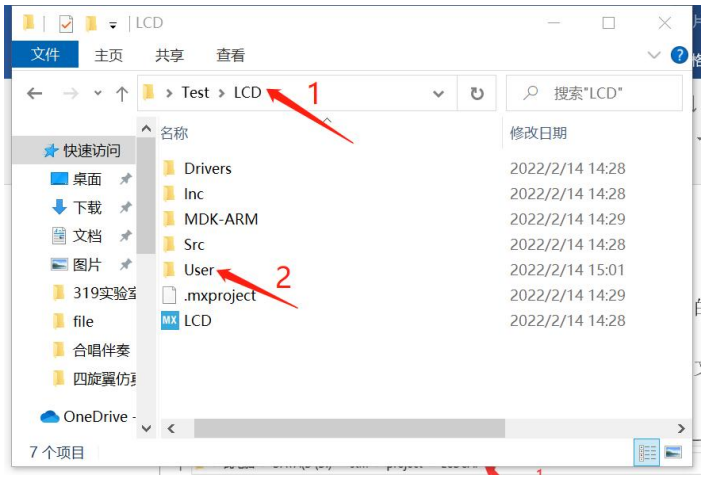

图 5.7.2 工程文件地址

5） 打开里面的User文件夹，将TFT LCD的底层驱动文件复制到该文件夹中。

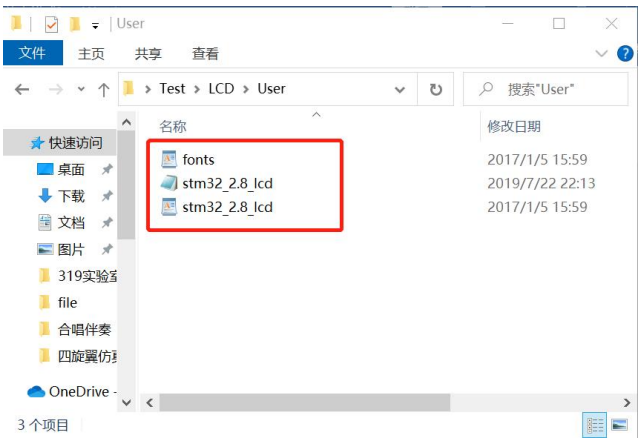

图 5.7.3 复制工程文件

6） 复制完成后，打开上面生成的Keil工程文件。右击工程文件名，再点击“Add Group”添加一个文件夹（用于添加底层驱动函数），可以将该文件夹命名为“User”。

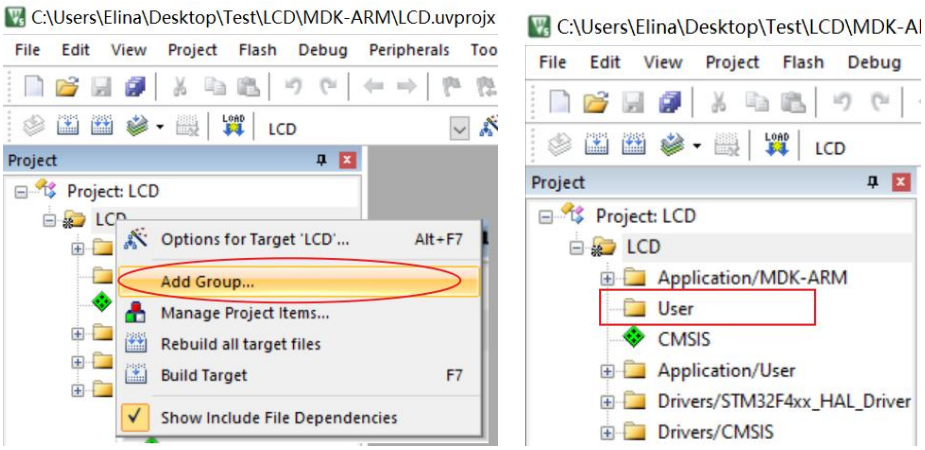

图 5.7.4 工程配置

7） 再将驱动文件添加到User文件夹中。右击“User”，点击下方红色标记的地方。

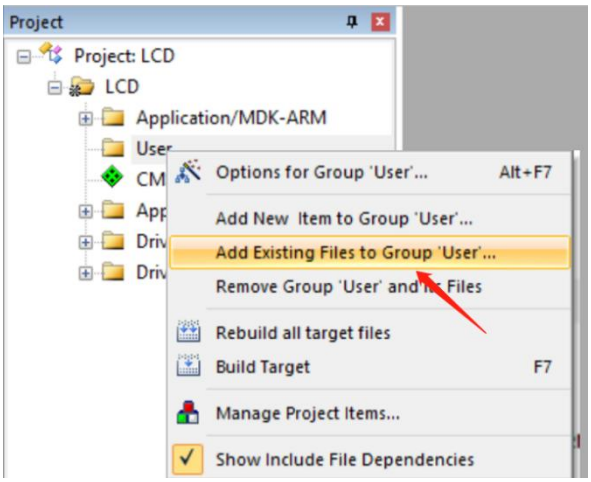

图 5.7.5 添加文件

8） 打开在工程目录下新建的“User”文件夹，选中里面的3个文件，点击“Add”即可。

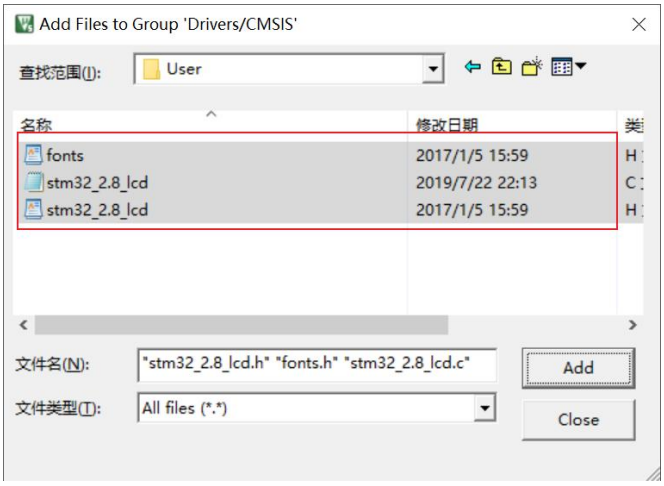

图 5.7.6 添加底层 TFT 驱动文件

9） 完成后，打开工程目标选项配置窗口，点击箭头所指地方，按顺序操作，添加头文

件路径。

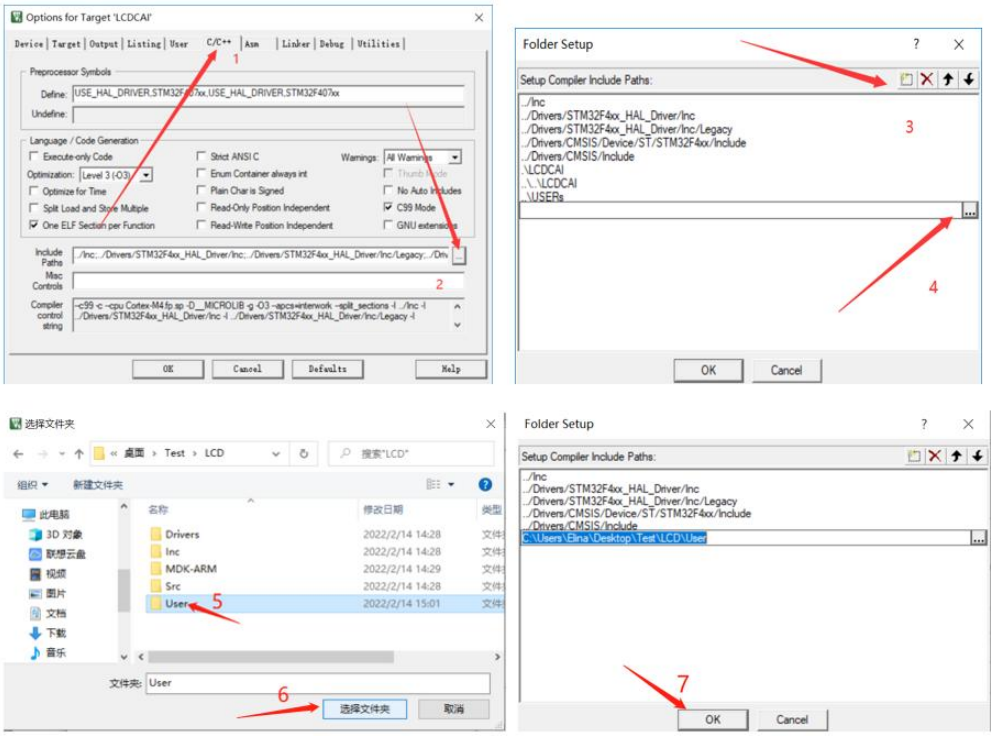

图 5.7.7 增加头文件路径

在主函数 mian 开始位置，添加 LCD 驱动头文件，参考代码如下。

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32_2.8_lcd.h"
/* USER CODE END Includes */
```

图 5.7.8 include 头文件

编程提示：用户代码必须添加在“/* USER CODE BEGIN Includes...*/”至“/* USER CODE END Includes... */” 之间。这样 CUBEMX 工程文件可以重复配置，并生成相关代码，而不影响用户原来写好的代码。

10）在主程序中添加如下代码，实现LCD对基本字符串的显示功能。

```
/* USER CODE BEGIN 2 */
STM32_LCD_Init(); //显示屏初始化
LCD_Clear(BackColor); //使用背景色清屏
LCD_SetTextColor(Blue);  //文本字体颜色，设为蓝色
LCD_DisplayStringLine(2,"www.fretech.com");  //第2行显示
LCD_DisplayStringLine(4,"STM32F407ZGT6... ");//第4行显示
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

图 5.7.9 添加 LCD 显示字符串代码

11）编译下载即可，最终在LCD上显示字符串如下图所示。

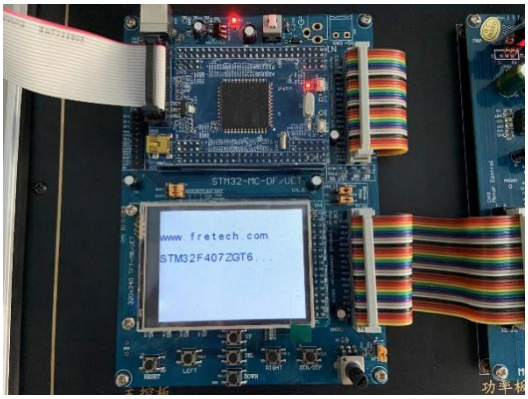

图 5.7.10 LCD 显示字符串

7.拓展实验

在基础实验中，我们掌握了用 LCD 显示字符串的方法。下面请同学自行实现在 LCD 上显示一个随着时间递增的整数。

编程提示：可以直接使用 LCD 头文件中的驱动函数实现，也可以自行编写函数实现。

实验八 串行通讯基本实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，串口的设置。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 学会使用 STM32 的串口通讯配置及编程。

2.实验设备

1 PC 机一台

2 实验平台一台

3 串口线一根

3. 实验内容

试编程实现串口与电脑进行通讯。

4.实验分析

平台主控板中，CN1 接口引出有 PA9 和 PA10 引脚，如图 5.8.1 所示。

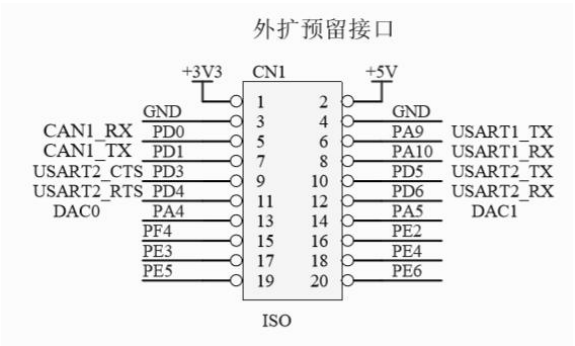

图 5.8.1CN1 串口定义

使用 TTL 串口线可以完成串口通讯的基本实验功能。TTL 线一端接到电脑 USB 接口上，一端的四根线中的三根线接 MCU 串行引脚（TXD、RXD、GND）。在 PC 电脑中，需要安装驱动程序方可识别串口。

5.实验步骤

（1）安装 TTL 串口驱动程序。

（2）串口线的白色线接到 CN1 中的 PA9 引脚（TX1），绿色接 CN1 中的 PA10（RX1），黑色接 GND。

（3）创建工程，配置相关参数，生成工程文件，并在工程文件中添加相应的代码。

（4）打开实验平台，对工程文件进行仿真下载。在电脑上打开串口调试助手，调试程序，查看运行结果。

6.实验例程

1） 新建CubeMX工程，参考前述实验。

2） 连接TTL串口线，USB一端接到PC，飞线一端白色线接到CN1中的PA9引脚（TX1），绿色接CN1中的PA10（RX1），黑色接GND，连接如下图所示（**注意红线不接！**）。

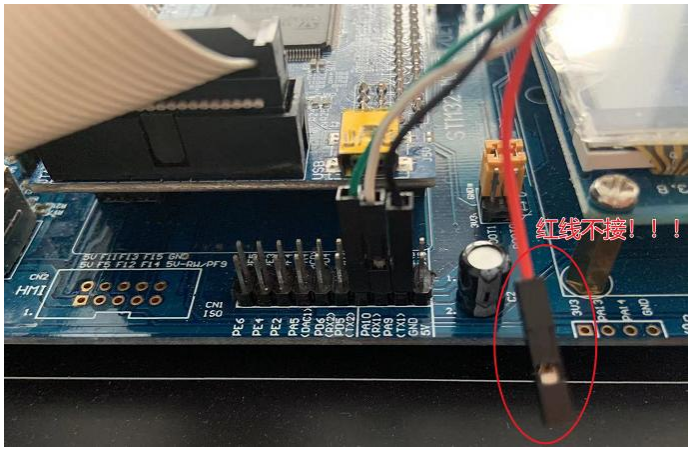

图 5.8.2 TTL 串口线连接

注意事项：连接时要非常小心，不要把针脚弄弯了！

3） 配置时钟，参考前述实验。

4） 对串口进行配置。在CubeMx最左边的配置选项目录窗口，选择USART1，在右侧界面对其进行配置，如下图所示。设置串口通信模式为“Asynchronous”异步通信模式，设置串口波特率为19200。串口配置完成后，可以在“Pinout view”界面看到PA10、PA9引脚已经被占用并分配给了USART1的RX端与TX端。

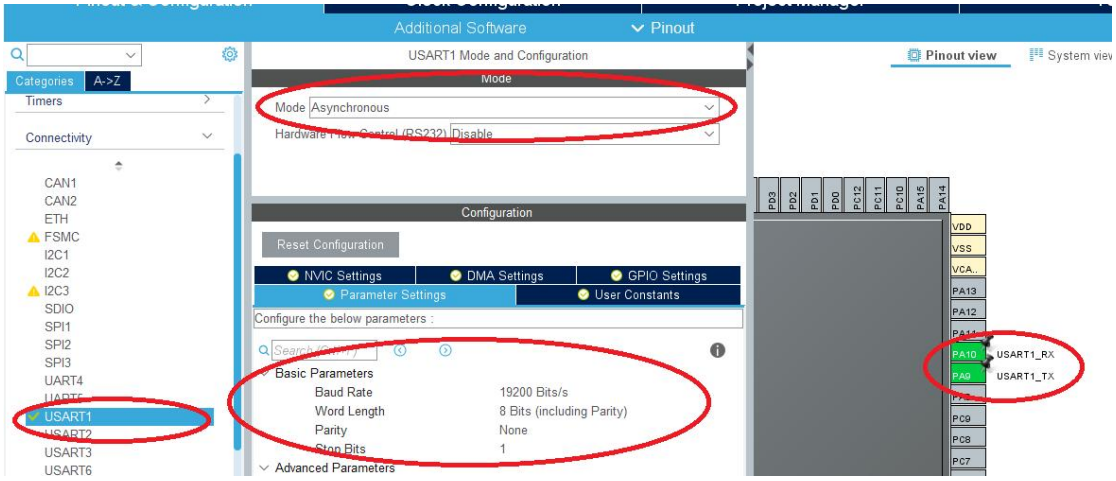

图 5.8.3 串口配置

5) 本示例采用中断的方式进行串口通信，所以还要对串口进行中断配置，中断配置的方法很简单，如下图所示，将“USART1 global interupt”使能即可。

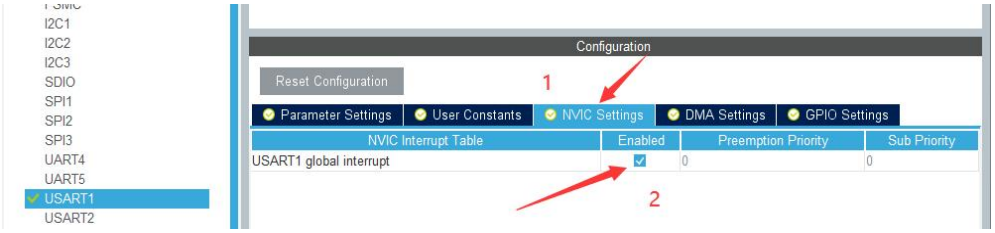

图 5.8.4 串口中断配置

6) 配置工程文件，并生成相关代码，参考前述实验。

7) 重定向printf、scanf函数。在C标准库中所用的标准输出函数，默认的输出设备是显示器，要实现打印数据通过串口或LCD的输出，必须重新定义标准库函数里与输出函数相关的函数。例如:printf输出到串口，需要将fputc里面的输出指向串口，改变打印输出方式的这一过程被称作“重定向”。方法如下：重写 int fputc(int ch, FILE *f)函数，使字符通过串口发送。重写int fgetc（FILE *f）函数，使字符通过串口接收。

8) 参考代码如下（以下代码添加在"stm32f4xx_hal_msp.c"中）。**注意：一定要将代码添加到相应的代码保护区，否则当你再一次使用STM32CubeMX生成c代码的时候就会全部被清除掉！**

```
/* USER CODE BEGIN Includes */
#include <stdio.h>
extern UART_HandleTypeDef huart1;
/* USER CODE END Includes */
```

```c
/* USER CODE BEGIN 0 */
int fputc(int ch, FILE *f)
{
  HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xffff);
  return ch;
}
int fgetc(FILE *f)
{
  uint8_t ch = 0;
  HAL_UART_Receive(&huart1, &ch , 1, 0xffff);
  return ch;
}

/* USER CODE END 0 */
```

图 5.8.5 重定向参考代码

9) 到此已经完成了Printf函数和scanf函数的重定向。下一步开始添加串口功能程序，实现STM32与PC机之间的简单通信。通信功能设定为：串口中断接收回显实验。由PC端串口调试助手发送数据给STM32串口端，STM32串口将接收到的数据返回给PC端，形成一个Loopback测试。以上功能的参考代码如下：

```c
/* Private includes ----------
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
#include "stm32_2.8_lcd.h"
/* USER CODE END Includes */
```

```c
/* USER CODE BEGIN PV */
uint8_t aRxBuffer = 0;
uint8_t Rx_buf[128] = {0};
uint8_t Rx_Count = 0;

/* USER CODE END PV */
```

```c
/* USER CODE BEGIN 1 */
uint8_t txbuf[50]= {0};
/* USER CODE END 1 */
```

```c
/* USER CODE BEGIN 2 */
memcpy(txbuf,"这是一个串口中断接收回显实验\n",50);
//TX_MODE();
HAL_UART_Transmit(&huart1,txbuf,strlen((char *)txbuf),1000);

memcpy(txbuf,"输入数据并以回车键结束\n",50);
HAL_UART_Transmit(&huart1,txbuf,strlen((char *)txbuf),1000);
//RX_MODE();
/* 使能接收，进入中断回调函数 */
HAL_UART_Receive_IT(&huart1,&aRxBuffer,1);
//RX_MODE();
/* USER CODE END 2 */
```

```c
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(huart);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_UART_TxCpltCallback could be implemented in the user file
   */


  HAL_UART_Transmit(&huart1,&aRxBuffer,1,0);
  HAL_UART_Receive_IT(&huart1,&aRxBuffer,1);
  HAL_UART_Receive_IT(&huart1,&aRxBuffer,1);
}
/* USER CODE END 4 */
```

图 5.8.6 串口通信参考代码

10）编译下载，在PC端用串口调试助手调试，确认串口收发是否正常。

11）使用串口调试助手时，注意配置串口参数与STM32串口一致，如下图所示。

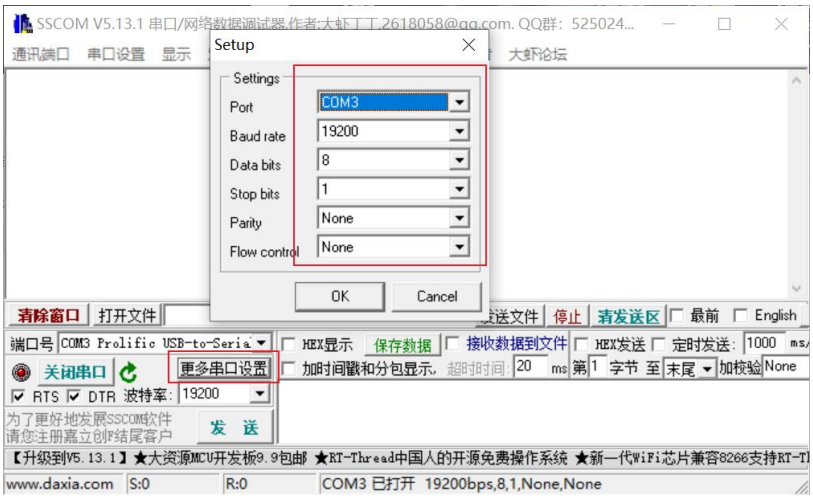

图 5.8.7 串口调试助手界面

12）在串口调试助手的发送窗口，可以输入要发送的字符或字符创，编辑好之后点击发送则将数据发送至STM32串口，勾选“加换行和回车”可以自动在所编辑的发送数据之后添加换行和回车。可以看到发送出去的数据，最终显示在接收窗口，如下图所示。

图 5.8.8 串口调试助手收发数据

7.拓展实验

拓展一：请尝试用串口重定向后的函数 printf 和 scanf 实现基础实验中的功能。

拓展二：有时候为了方便调试，我们会将串口数据通过 LCD 显示出来。请同学们自行尝试完成如下功能：由 STM32 串口的 TX 端向 PC 端定时发送一个递增的整数，STM32 的 RX 端通过中断方式接收 PC 发送的数据；由 STM32 发送和接收的数据均通过 LCD 屏显示出来;PC 端用串口调试助手来监控。在 LCD 显示屏上，可以看到 STM32 串口发送与接收的数据，如下图所示。

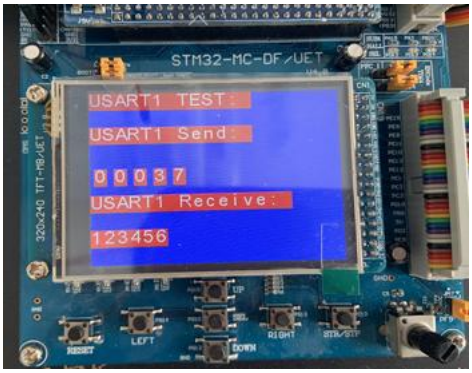

图 5.8.9 LCD 显示

实验九 DMA 直接内存访问实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、DMA 的设置。

2 掌握在 MDK 的使用、DMA 程序的添加以及其他相关的操作。

3 学会使用 STM32 的 DMA 存储器到存储器（MEMTOMEM）传输。

2.实验设备及器件

1 PC 机一台

2 实验平台一台

3. 实验内容

试编程，实现存储器到存储器的 DMA 传输。

4.实验分析

DMA 用来提供外设和存储器之间或者存储器和存储器之间的高速数据传输。传输过程中，CPU 是闲置的，数据的高速传输不需要用到 CPU,节省了 CPU 资源来做其他的操作，比如在例程中的点亮 LED 灯。

因为 DMA 使用的是内部软件设计，所以在此工程中，不需要外接硬件。最终实验可以验证不用处理器的干预即可完成 MEMTOMEM 的数据传输。

5.参考实验例程

1） 新建CubeMX工程，参考前述实验。

2） 配置时钟，参考前述实验。

3） 配置三个LED灯作为DMA的传输提示：LED1~LED3分别作为传输成功、传输出错、未进行传输三种状态的提示，配置方法参照前述实验。

4） 添加DMA，根据下列提示，点击Add按钮，选择“MEMTOMEM”，存储器到存储器的DMA传输。

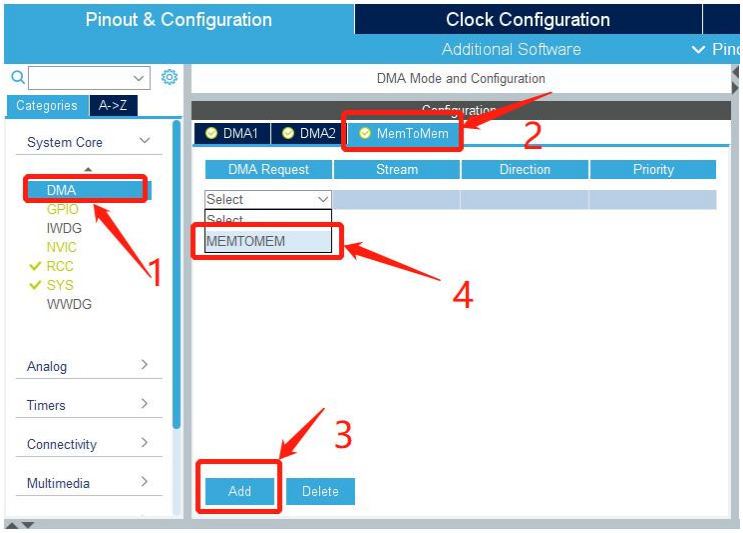

图 5.9.1 添加 DMA

5） 配置DMA，选择通道0，优先权为高，Data Width有三种选择，请同学们自行思考其功能并配置，如下图所示。

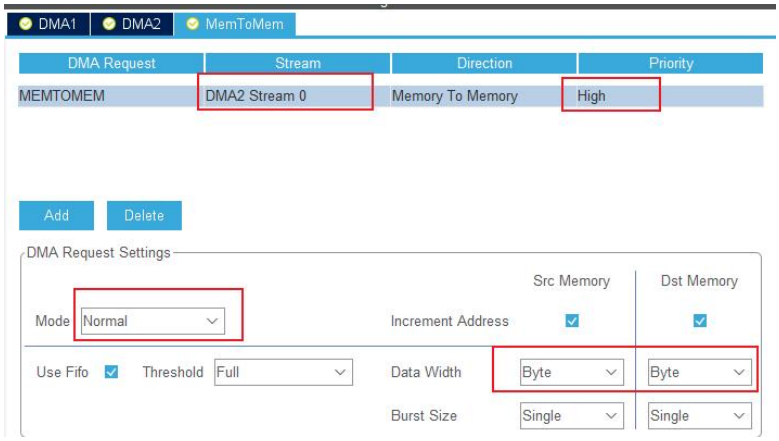

图 5.9.2 配置 DMA

6） 配置工程文件，并生成相关代码，参考前述实验。

7） 我们可以对DMA的MEMTOMEM的程序功能进行分析，程序基本流程如下。

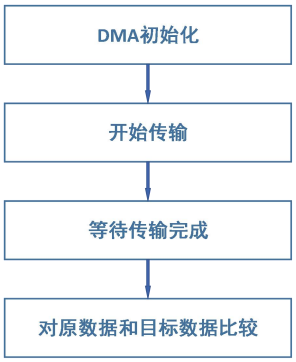

图 5.9.3 DMA 编程流程分析

8） 定义DMA传输相关变量，使用SRC_Const_Buffer[32]来存放原数据，使用const关键字，使得变量存储在Flash空间。使用typedef定义枚举变量TestStatus，用来定义程中关于传输成功与失败的状态，如下所示。

```
typedef enum
{
  FAILED = 0,
  PASSED = !FAILED
} TestStatus;
static const uint32_t SRC_Const_Buffer[32]= {
  0x01020304,0x05060708,0x090A0B0C,0x0D0E0F10,
  0x11121314,0x15161718,0x191A1B1C,0x1D1E1F20,
  0x21222324,0x25262728,0x292A2B2C,0x2D2E2F30,
  0x31323334,0x35363738,0x393A3B3C,0x3D3E3F40,
  0x41424344,0x45464748,0x494A4B4C,0x4D4E4F50,
  0x51525354,0x55565758,0x595A5B5C,0x5D5E5F60,
  0x61626364,0x65666768,0x696A6B6C,0x6D6E6F70,
  0x71727374,0x75767778,0x797A7B7C,0x7D7E7F80};
```

9） 使用Buffercmp函数比较两个数据源是否相等，函数有三个形参，用于判断pBuffer和pBuffer1这两个变量的数据是否相等，根据返回值的不同，在main函数中进行处理。

```
01 TestStatus Buffercmp(const uint32_t* pBuffer, uint32_t* pBuffer1, uint16_t BufferLength)
02 {     /* 数据长度递减 */
03     while (BufferLength--) {
04           /* 判断两个数据是否相等 */
05         if (*pBuffer != *pBuffer1) {
06             return FAILED;
07         }
08         /* 递增两个数据源的地址指针*/
09         pBuffer++;
10         pBuffer1++;
11     }
12     return PASSED;
13 }
```

图 5.9.4 Buffercmp 函数参考代码

10）我们根据三个LED的亮灭来判断传输结果。HAL_DMA_Start函数用来启动DMA传输，可以定义一个结构体变量har_status来判断传输的状态。传输完成后检查数据一致，从而决定三个LED灯的亮灭，主函数参考代码如下。

11）编译下载，观察实验现象。

```
har_status=HAL_DMA_Start(&hdma_memtomem_dma2_stream0,
                         (uint32_t)&SRC_Const_Buffer,(uint32_t)&DST_Buffer,32);
if (har_status==HAL_OK) {
    /* 检查发送和接收的数据是否相等 */
    TransferStatus = Buffercmp(SRC_Const_Buffer, DST_Buffer, 32);

    /* 如果接收和发送的数据都是相同的，则通过 */
    if (TransferStatus == PASSED) {
        LED1_ON;
    }
    /* 如果接收和发送的数据不同，则传输出错 */
    else {
        LED2_ON;
    }
} else {
    LED3_ON;
}
```

图 5.9.5 main 函数参考代码

12）调试代码，最终可以看到LED1点亮，代表数据传输通过。

7.拓展实验

每个 DMA 数据流都有四级 32 位 FIFO，DMA 传输具有 FIFO 模式和直接模式。直接模式下在收到外设请求时启动传输，在间接模式下，DMA 会先将数据放在 FIFO 内，当外设启动 DMA 时再将 FIFO 内的数据传输过去。

FIFO 对于要求源地址和目标地址数据宽度不同时非常有用，此时使用 FIFO 功能先把数据缓存起来，再根据需要输出数据。

在以上例程中，我们使能了 DMA 的 FIFO 功能，但输入和输出的数据宽度是相等的。下面请同学们自行尝试，使用带 FIFO 的 MEMTOMEN 传输，将 4 个 8 位数据拼凑成一个 32 位数据。

实验十 DMA-UART 收发实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，存储器到外设的设置。

2 掌握在 MDK 的使用、DMA 存储器到 UART 外设程序的添加以及其他相关的操作。

3 学会使用 STM32 的 DMA 存储器到外设编程。

2.实验设备及器件

1 PC 机一台

2 实验平台一台

3 串口线一根

3. 实验内容

采用 DMA 存储器到 UART 外设传输模式，将指定的存储器数据转移到 UART 寄存器内，并发送至 PC，由串口调试助手显示。

4.实验分析

本实验采用 DMA 存储器到外设传输模式，经由 DMA 发送 UART 数据到 PC，是 DMA 的一种常见应用。我们可以用简单的例程来实现这一过程。程序的流程大概包括：配置 DMA 传输是内存到外设；配置中断；对要发送的内存数据进行填充；发送数据等。为了直观的展现 DMA 传输并不占用 CPU，我们可以加入 LED 灯的闪烁功能。

5.参考实验例程

1） 检查JP17跳线是否短接。

2） 新建CubeMX工程，参考前述实验。

3） 配置时钟，参考前述实验。

4） 配置LED，用来作为主程序循环正常运行的指示，配置参考前述实验。

5） 配置串口，使能中断，参考前述实验。

6） 配置DMA，根据USART发送端选择DMA通道。根据STM32F407编程手册DMA请求映射表可知，USART1对应DMA2数据流7的通道4，如下图所示。

外设请求	数据流 0	数据流 1	数据流 2	数据流 3	数据流 4	数据流 5	数据流 6	数据流 7
通道 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
通道 1		DCMI	ADC2	ADC2		SPI6_TX(1)	SPI6_RX(1)	DCMI
通道 2	ADC3	ADC3		SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
通道 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
通道 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
通道 5		USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)		USART6_TX	USART6_TX
通道 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
通道 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM

图 5.10.1 DMA2 请求映射表

7） 点击Add按钮，设置发送端启用DMA传输，通道、方向、优先权，传输模式有两种，先选择循环模式（请同学们思考其作用）。存储器地址使用自动递增，外设地址自增不能使用，因为USART有固定的DMA通道，USART数据寄存器的地址也是固定的。如下图所示。

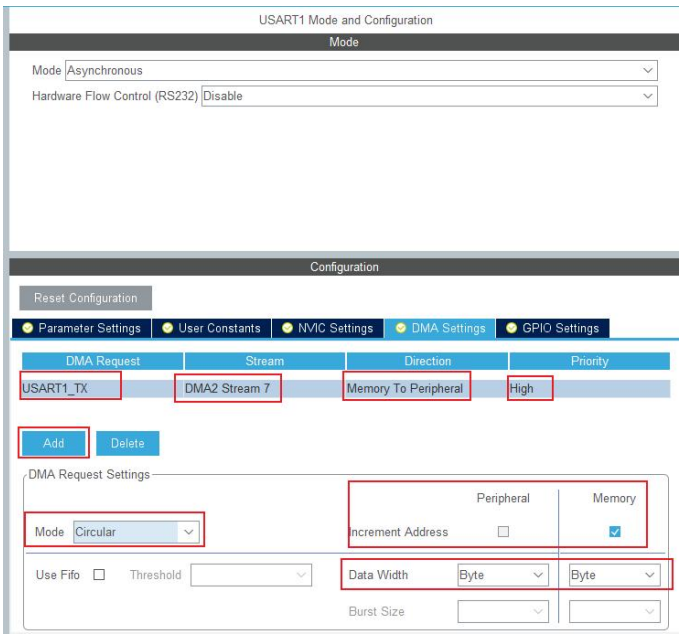

图 5.10.2 DMA2 配置

8） 配置工程文件，并生成相关代码，参考前述实验。

9） 编写程序，在main程序中，首先是一些节本配置和初始化函数的调用，使用aTxBuffer来存放数据，调用DMA发送函数来发送数据到电脑端。调用HAL_UART_Receive_IT()使能接收函数，此时进入中断回调函数，由下方的回调函数实现中断的服务。参考代码如下。

```
int main(void)
{
    uint16_t i;

    /* 复位所有外设，初始化 Flash 接口和系统滴答定时器 */
    HAL_Init();
    /* 配置系统时钟 */
    SystemClock_Config();

    /* 初始化 LED */
    LED_GPIO_Init();

    /* 初始化串口并配置串口中断优先级 */
    MX_USARTx_Init();

    /*填充将要发送的数据*/
    for (i=0; i<SENDBUFF_SIZE; i++) {
        aTxBuffer[i]    = 'Y';
    }

    /* 使能接收，进入中断回调函数 */
    HAL_UART_Receive_IT(&husartx,&aRxBuffer,1);

    /* 使用 DMA 传输数据到电脑端 */
    HAL_UART_Transmit_DMA(&husartx,aTxBuffer, SENDBUFF_SIZE);

    /* 无限循环 */
    while (1) {
        /* 串口使用 DMA 传输数据不占用 CPU，可以正常运行其他函数 */
        LED1_ON;
        HAL_Delay(500);
        LED1_OFF;
        HAL_Delay(500);
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
    HAL_UART_Transmit(&husartx,&aRxBuffer,1,0);
    HAL_UART_Receive_IT(&husartx,&aRxBuffer,1);
}
```

图 5.10.3 参考代码

10）编译下载程序，观察实验现象，可以看到在串口调试助手界面，在数据接收区不停的接收“Y”，在开发板上可以看到LED等在不断闪烁。如下图所示。

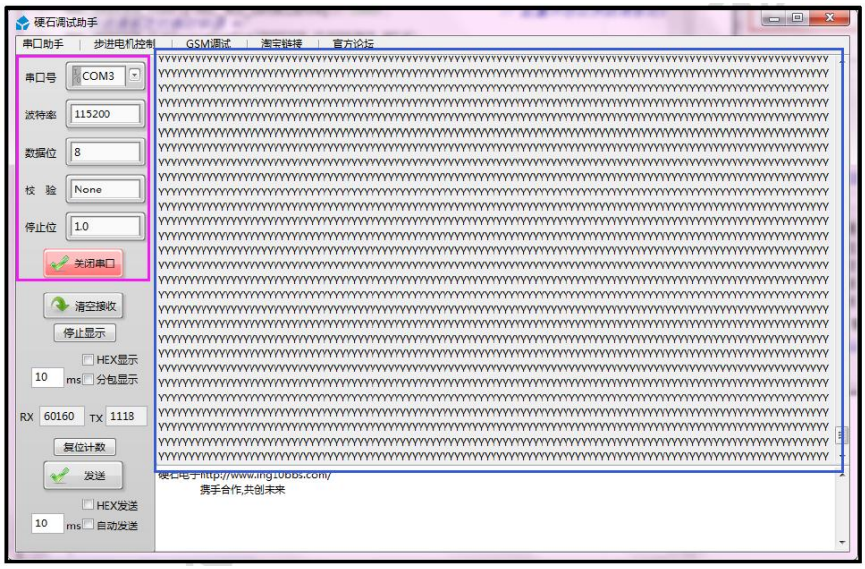

图 5.10.4 串口调试助手界面

11）如果将DMA传输方式配置为“Normal”的话结果会有什么改变？请同学们自行尝试并思考两种传输方式的原理和区别。

实验十一 ADC 采集实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、程序外设的配置，ADC 的设置。

2 掌握在 MDK 的使用、ADC 程序的添加以及其他相关的操作。

3 学会使用 STM32 的 AD 转换编程。

2.实验设备及器件

1 PC 机一台

2 实验平台一台

3. 实验内容

试编程，采集电位器电压并显示到 TFT 彩屏上。

4.实验分析

通过主控板原理图可知，在跳线 JP17 连接的情况下，主控板右下端的电位器 RV1 的输出端与 PC4 通道连接。而 PC4 通道在 STM32 内部连接了 ADC1、ADC2 的 14 通道，如下图所示。

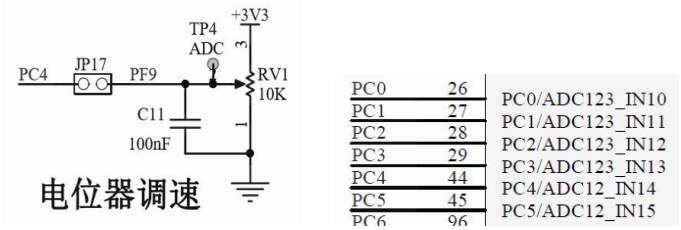

图 5.11.1 电位器原理图

5.参考实验例程

12）检查JP17跳线是否短接。

13）新建CubeMX工程，参考前述实验。

14）配置时钟，参考前述实验。

15）配置PC4引脚为ADC输入模式，在这里选用ADC1的14号通道，如下图所示。

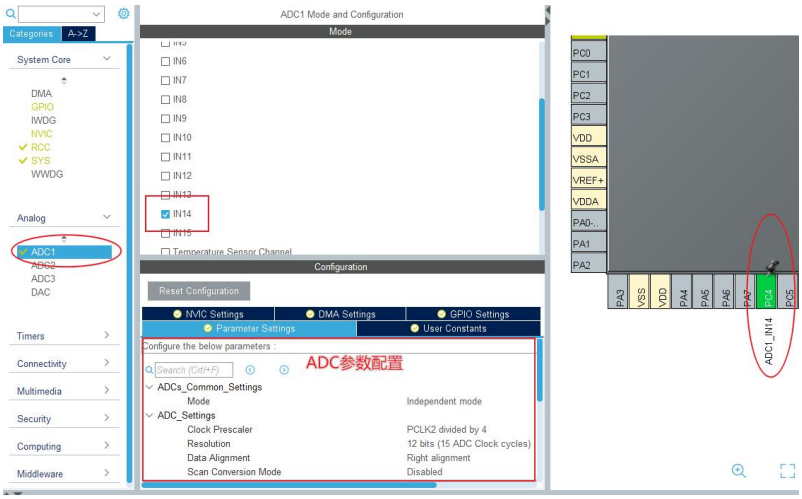

图 5.11.2 配置 ADC 输入引脚

16）在参数配置界面，可以更改ADC转换时钟频率、分辨率 、对齐方式等参数，这里更改“Continous Conversion Mode”为Enable，其他参数保持不变，如下图所示。

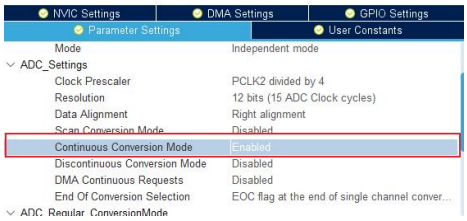

图 5.11.3 配置 ADC 参数

17）配置工程文件，并生成相关代码，参考前述实验。

18）打开工程文件，添加自己的功能程序。

19）定义一个变量用来存放ADC的值：**uint16_t ADC_Value。**

20）在while循环中实现对ADC的值，并用LCD显示出来，参考代码如下（**LCD使用参考前述实验！**）。

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_ADC_Start(&hadc1);     //启动ADC转换
    HAL_ADC_PollForConversion(&hadc1, 50);   //等待ADC转换完成
    if(HAL_IS_BIT_SET(HAL_ADC_GetState(&hadc1), HAL_ADC_STATE_REG_EOC))  //获取ADC转换状态
    {
      ADC_Value = HAL_ADC_GetValue(&hadc1);   //获得ADC的值
      LCD_DisplayStringLine(Line0, " AD test:  ");    //第0行显示
      LCD_Draw_NUM(70,300,ADC_Value);   //显示二进制数值
    }
    HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
```

图 5.11.4 参考代码

21）编译下载，**轻轻**扭动电位器观察显示屏上的数字变化。

图 5.11.5 LCD 显示结果

22）LCD所显示的只是ADC转换的原始值，并不是ADC测量到的真实电压。请同学们写出LCD显示数值与实际电压值的转换关系，将原始值转换成真实值，一并显示在LCD屏上。

6.拓展实验

以上是采用轮询的方式获取 ADC 的值，在实际应用中，经常需要连续采集大量 ADC 数据，为了避免前一个采样时间的数据被下一个时间点覆盖，我们会采用 DMA 的方式进行 ADC 高速采集。

使用 DMA 功能时，时钟和引脚配置完全如上，此外需要添加的是 DMA 的设置，在 ADC1 的“DMA Settings”界面，添加一个 DMA，Select 处选择 ADC1,如下图所示。

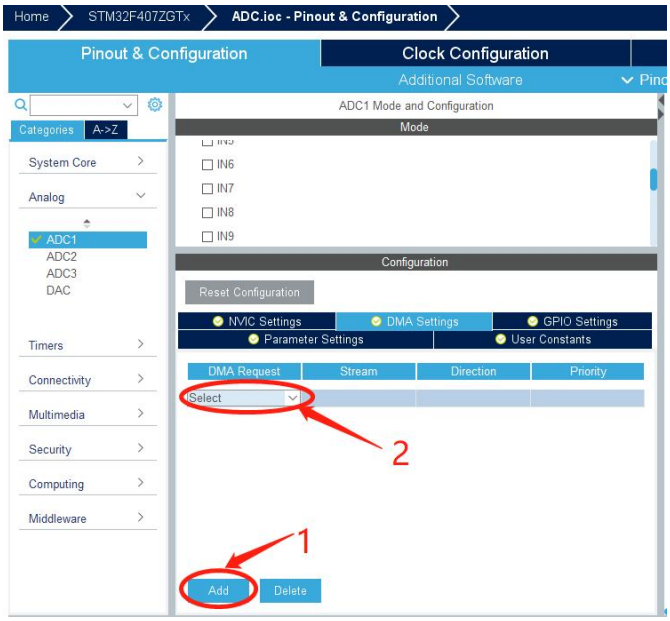

图 5.11.6 增加 DMA 功能模块

DMA 的配置与使用方法与串口 DMA 传输类似，请同学们用使能了 DMA 的 ADC 采集，来实现实验例程的功能。

实验十二 AD 转换及定时器 PWM 输出实验

1.实验目的

1 掌握 STM32CubeMX 使用方法、外设配置方法和定时器 PWM 相关工作原理。

2 掌握在 MDK 的使用、程序的添加以及其他相关的操作。

3 熟悉 STM32 单片机的 PWM、ADC 控制。

2.实验设备

1 PC 机一台

2 实验平台一台

3.实验内容

试编程，通过 PWM 控制的方式实现电位器电压对 LED 亮度的控制。

4.实验分析

以主板右下端电位器 RV1 的输出控制驱动 LED5 的亮度，其原理如下。

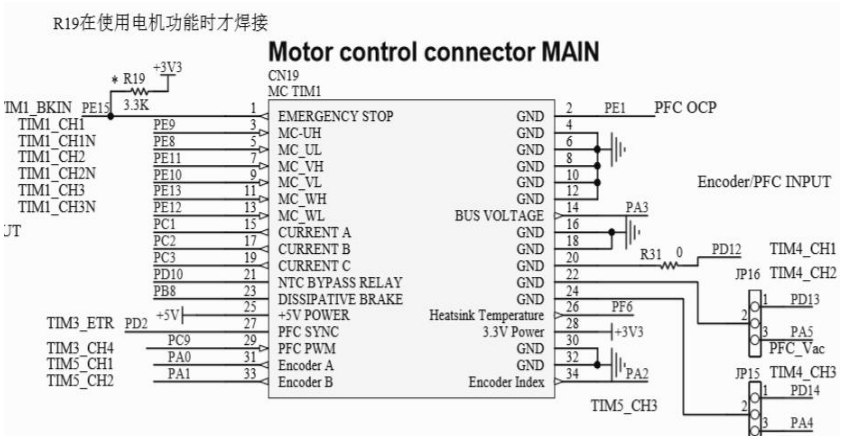

图 5.12.1 主电机接口定义

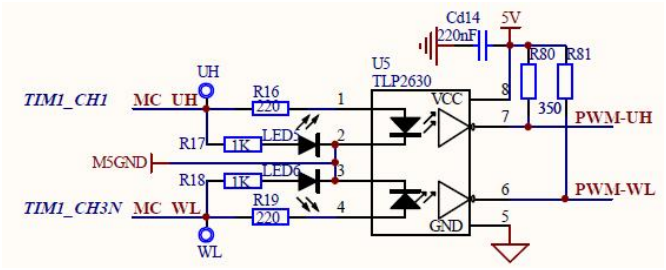

图 5.12.2 驱动板 LED5 部分电路图

从图中分析可得，驱动板 LED5 指示灯的亮灭受 TIM1_CH1（PE9）通道的控制,TIM1_CH1 通道可以生成 PWM 波，调整 PWM 波的占空比可以调整 LED5 的亮度。本实验中，可以用 ADC 采集电位器电压，将 AD 转换的结果值（0~4095）换算成 TIM1_CH1 的 PWM 脉宽值，使不同的 AD 结果值控制 LED5 的亮灭程度，从而实现此实验的功能要求。

5.实验步骤

（1）用跳线将 JP17 短接。按照前几章的内容，新建立一个工程。

（2）生成工程文件，添加相应代码。

（3）打开实验台，对工程文件进行仿真或下载。调节 RV1，观察驱动电路板 LED5 的亮灭。

6.实验例程

1） 检查JP17跳线是否短接。

2） 新建CubeMX工程，参考前述实验。

3） 配置时钟，参考前述实验。

4） 配置ADC，采用DMA采集传输模式，具体配置参考前述实验。

5） 配置Timer1，clock Source选择内部时钟，Channel1模式配置成PWM Generation模式，如下图所示。

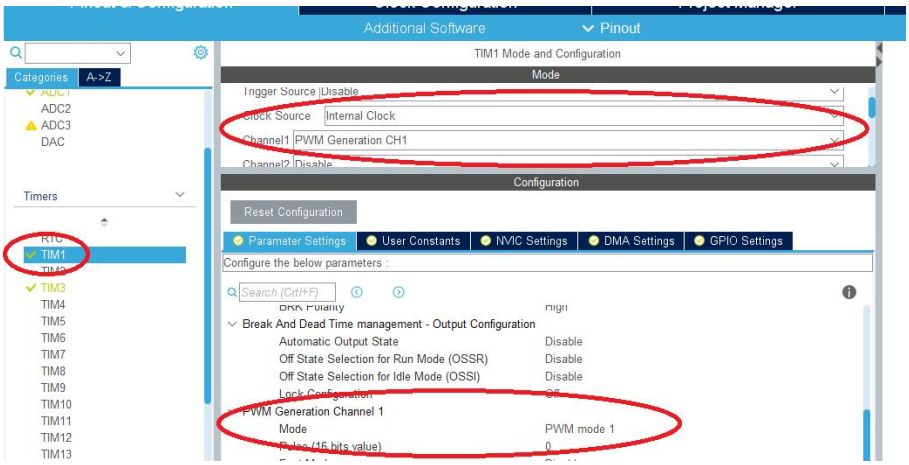

图 5.12.3 Timer1 的 CH1 配置成 PWM 输出模式

6） 配置定时器基本功能，设置定时器的周期，使能自动重载功能，如下图所示。

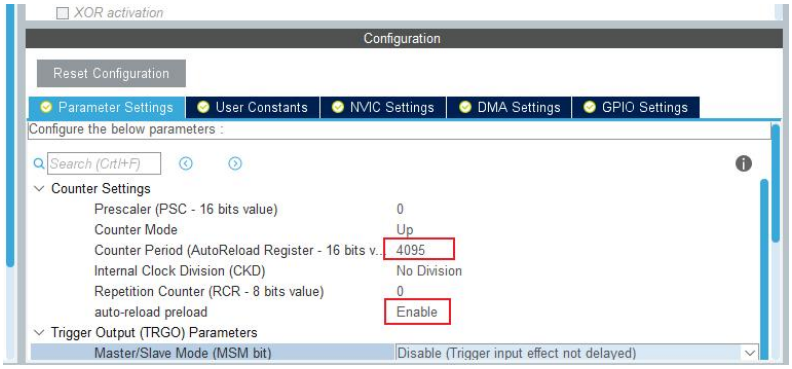

图 5.12.4 配置 PWM 波形参数

7） 配置工程文件，并生成相关代码，参考前述实验。

8） 打开工程文件，添加自己的功能程序。

9） 参考代码如下。

```
/* USER CODE BEGIN 2 */
HAL_NVIC_DisableIRQ(DMA2_Stream0_IRQn);
STM32_LCD_Init();
LCD_Clear(Red);
LCD_SetBackColor(Blue);
LCD_SetTextColor(White);
LCD_DisplayStringLine(Line0,  " AD test:  ");
HAL_ADC_Start(&hadc1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&ADC_ConvertedValue[0], 1);
HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_1); //启动PWM波生成
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  __HAL_TIM_SetCompare(&htim1, TIM_CHANNEL_1, ADC_ConvertedValue[0]);  //设置PWM波占空比为ADC采集值
  LCD_Draw_NUM(35,280,ADC_ConvertedValue[0]);  //显示ADC采集值
  HAL_Delay(20); //延时20ms
}
/* USER CODE END 3 */
```

图 5.12.5 PWM 波形生成参考代码

10）编译下载，**轻轻**扭动电位器，观察小灯的亮度变化。

11）试用定时器实现HAL_Delay(20)的功能，并思考两种方式的区别。

7.拓展实验

以上例程，我们通过旋转电位器，更改 PWM 占空比，从而改变 LED 灯的亮度。LED 亮度随时间由暗到亮逐渐增强，再由亮到暗逐渐衰减，有节奏的起伏，就像呼吸一样，我们称之为呼吸灯。下面请同学们尝试实现一个呼吸灯。

编程提示：LED 亮度随着时间逐渐变强再衰减，可以用指数曲线变化来表示。如果我们让 PWM 输出这种类型的信号，那么就可以实现呼吸灯的功能。我们将制定好的数据存放在 indexWave 中，将表中的数据画成图，如下图所示，其中横坐标为时间，纵坐标为 indexWave[x]，亦为当前时间的 TIMx_CRR 的设定值，两个点之间的时间间隔相等。

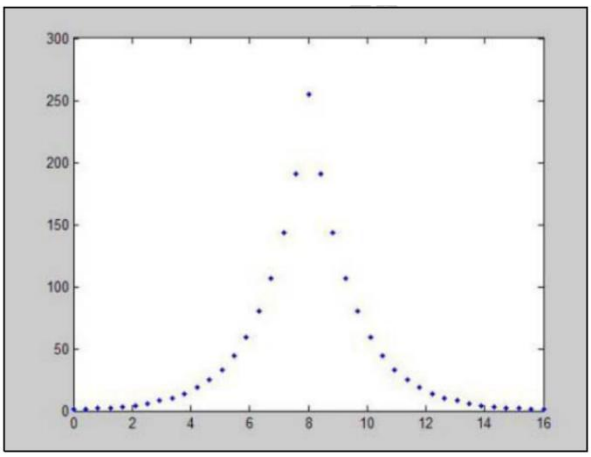

图 5.12.6 数组内数据对应的图形

indexWave 可以参考下面的定义。

```
uint8_t indexWave[] = {1,1,2,2,3,4,6,8,10,14,19,25,33,44,59,80,107,143,191,255,
                       255,191,143,107,80,59,44,33,25,19,14,10,8,6,4,3,2,2,1,1};
```