哈尔滨理工大学 自动化学院

《单片微型计算机原理及嵌入式系统-II》

项目设计与实训报告

题目：实验一流水灯实验

姓名 [redacted]

学号 [redacted]

成绩 _____

完成时间：2023 年 11 月 15 日

一、项目概述（500 字以内，10 分）

本项目旨在为我们提供实践机会，通过使用 STM32F103C8T6 最小系统板搭建外围电路，实现一个经典的 8 位流水灯实验。通过实验，我们能够掌握 STM32 单片机 GPIO 的初始化配置方法，并理解如何驱动指示灯的方法。主要目标是通过对 GPIO 的控制，实现 LED 灯按顺序依次点亮然后依次熄灭，并持续这个循环 2 秒。

在项目中，我选择了 STM32F103C8T6 作为核心处理器，并搭配洞洞板、杜邦线、直插式 LED 灯来完成实验。软件设计方面，核心是 GPIO 的初始化配置及控制。首先，将 STM32F103C8T6 的 GPIO 引脚配置为输出模式，以便控制 LED 灯。然后通过编程逻辑，依次点亮或熄灭每个引脚，实现流水灯的效果。利用定时器或延时函数来控制 LED 灯亮起和熄灭的时间，确保持续 2 秒的流水灯效果。

项目的步骤包括首先配置 STM32F103C8T6，然后编写控制逻辑，确保 LED 灯按顺序点亮和熄灭，并在预设的时间内循环。我们学习了如何使用定时器或延时功能，以及如何调试和优化代码，以确保流水灯效果的正确性和稳定性。

最后通过实践操作，我对 STM32 单片机的 GPIO 初始化配置方法有了更深的理解，也熟练地掌握了如何使用 STM32 单片机的 GPIO 驱动 LED 灯，从多种角度理解了流水灯的实现逻辑，包括 LED 的顺序控制和时间控制。在实验过程中，通过调试和优化代码，我感觉自己的软件开发技能有所提高。

二、项目的具体目标和指标要求（10 分）

设计子目标：

- 设计与搭建外围电路：使用 STM32F103C8T6 最小系统板、面包板和 8 个 LED 灯及杜邦线构建一个外围电路。
- 实现 8 位流水灯效果：控制 LED 灯按顺序依次亮起，然后依次熄灭，并确保整个循环持续 2 秒。
- 完成 GPIO 初始化配置：针对 8 个 LED 灯，将对应的 GPIO 引脚配置为输出模式。
- 控制 LED 灯的顺序：设计程序逻辑，确保 LED 灯按照指定的顺序依次点亮和熄灭。
- 控制 LED 灯的流水时间间隔：使用定时器或延时函数确保 LED 灯的亮起和熄灭时间，使流水灯效果持续 2 秒。

欲达到的指标要求：

- 外围电路搭建：STM32 与面包板连接稳固，以连接 8 个 LED 灯。LED 灯正确地连接至相应的 GPIO 引脚，确保 LED 能被控制。
- GPIO 初始化配置：GPIO 引脚成功配置为输出模式，以控制 LED 灯的开关。没有短路或其他硬件连接问题，确保引脚配置正确。
- LED 灯控制实现：LED 灯按照指定的顺序依次亮起、熄灭，构成流水灯效果。LED 的亮起和熄灭时间符合预期，使流水灯效果在 2 秒内完成一个完整循环。
- 稳定性与效率：确保程序运行稳定，无明显闪烁或错误。代码设计高效，逻辑清晰，易于理解和维护。

三、项目的具体内容（62 分）

（一）详细分析设计项目的各项内容，具体应包括以下几方面：

1、原理分析

本次流水灯实验采用库函数法来对相应的寄存器进行配置。通过调用 stm32

官方公司编写好的库函数，能够快速的对实验所需要的寄存器进行相应的配置，从而完成本次实验的目标。对手中 stm32 最小系统板进行硬件分析，对 LED 用库函数对其进行初始化配置和控制，来完成实验。

- GPIO 引脚配置原理：GPIO 引脚的配置涉及到控制相应的寄存器位，以设置引脚工作模式、输出速度、输出引脚等。这是通过对 GPIO 的寄存器进行操作实现的。
- LED 驱动原理：控制 LED 灯的亮灭是通过控制对应的 GPIO 输出引脚状态实现的。在 STM32 中，通过设置输出寄存器的特定位可以使引脚输出高电平或低电平，进而控制 LED 的亮度。而流水灯效果的原理正是通过循环控制，逐一点亮和熄灭每个 LED，形成流水灯效果。

2、方案论证

首先对 LED 进行相关的 GPIO 初始化配置，然后编写以实现流水灯效果的代码，在每一步编写代码时都进行编译，以防出现错误。最后，在确保编译无误的情况下进行仿真，仿真效果达到预期效果后，将程序导入最小系统板中，观察实验现象。

- GPIO 引脚选择

在进行本次实验前，应当先查看 STM32F103C8T6 引脚定义表，避免选择一上电就是默认复用功能的引脚，并确保这些引脚不会干扰其他必要的功能。同时为了方便控制，最好是选择同一组连续的引脚。因此本次实验中我选择的是 PA0-PA7 来控制 8 个 LED 灯。

- 硬件连接

在上电观察实验现象前，为确保安全以及不烧坏板子的原则下，应当检查 STLINK 与 STM32 的连线以及面包板上 LED 的连接，特别注意电源线和 LED 正负极不要插错。

LED 的连接有两种方式，可以采用共阳极和共阴极接法，共阳极即 LED 的负极接到单片机的引脚上，正极一起接到单片机的 3V3 上，共阴极即 LED 的正极接到单片机的引脚上，负极与单片机共地。本次实验我采用的是共阴极接法。

- 流水灯效果可行性

通过控制每个 LED 的 GPIO 引脚来实现流水灯效果，并保证持续时间为 2 秒。

3、功能模块设计和算法流程

本次实验我采用了两种编程方法，两种方法主要的区别在于延时方法上面，一种采用的是定时器延时，一种采用的是延时函数延时，来达到持续时间 2 秒的效果，整体思路还是一致的。整个实验的原理图如图 1 所示，程序的流程图如图 2 所示：

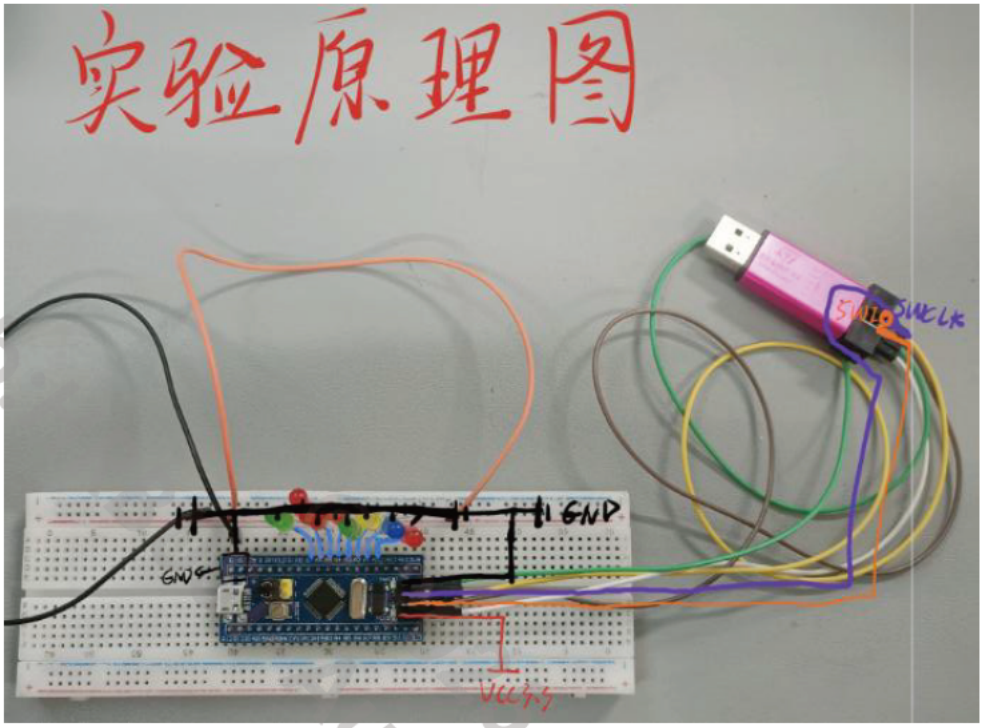

图 1　流水灯实验原理图

图 2　流水灯实验整体流程图

①LED 初始化模块

初始化部分主要是对 PA0-PA7 这八个引脚配置，通过 GPIO 的库函数，将 PA 组引脚配置为推挽输出，以连接 8 个 LED 灯，为 LED 点亮和熄灭做好准备。

该模块代码如下：

```
1.    void led_init(void)
```

```
{
    GPIO_InitTypeDef GPIO_LED_structure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);          //开启 GPIOA 时钟
    GPIO_LED_structure.GPIO_Mode=GPIO_Mode_Out_PP;     //推挽输出
    GPIO_LED_structure.GPIO_Pin=GPIO_Pin_All;               //选择控制引脚
    GPIO_LED_structure.GPIO_Speed=GPIO_Speed_50MHz;      //选择输出速度
    GPIO_Init (GPIOA,&GPIO_LED_structure);
    GPIO_SetBits(GPIOA,GPIO_Pin_All);          //初始化都给 0，亮
}
```

该模块配置流程图流程如下图 3 所示：

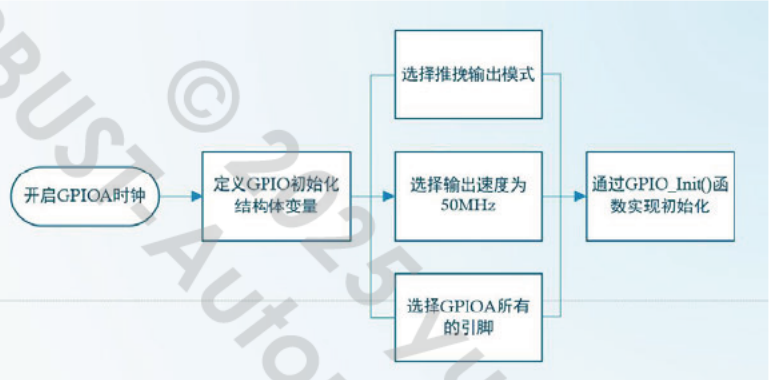

图 3　LED 灯初始化模块流程图

②流水灯控制模块

流水灯控制模块负责控制 8 个 LED 灯的状态变化，使其依次点亮然后依次熄灭，构成流水灯效果。通过循环移位控制 GPIO 引脚的状态，依次将其设置为高电平或低电平，以便控制 LED 灯的亮灭状态。设置完成后并延时一段时间，由于整个过程需要持续 2 秒，经过我的计算，每次移位需要延时 143 毫秒。这个模块需要精确的顺序控制以实现流水灯的效果。

该模块代码如下：

```
void LED_Control(void)
{
    for(i=0;i<8;i++)
    {
    GPIO_Write(GPIOA,0x0001<<i); //0000 0001 -> 0000 0010
    Delay_ms(143);
    }
    for(i=0;i<8;i++)
    {
    GPIO_Write(GPIOA,0x0001<<(8-i)); //1000 0000 -> 0100 0000
    Delay_ms(143);
    }
}
```

③时间控制模块（定时器和延时函数）

一：延时函数延时

采用延时函数来延时比较简单，只需编写一个函数并且调用即可，代码如下：

```
void Delay_ms(int32_t ms)
{
 int32_t i;
 while(ms--)
 {
 i=7500; //开发板晶振 8MHz 时的经验值
 while(i--);
 }
}
```

二：定时器延时

采用定时器可以对 LED 灯亮灭的时间做到更为精确的控制，确保流水灯效果在 2 秒内完成一个完整循环。该模块配置流程图流程如下图 4 所示：

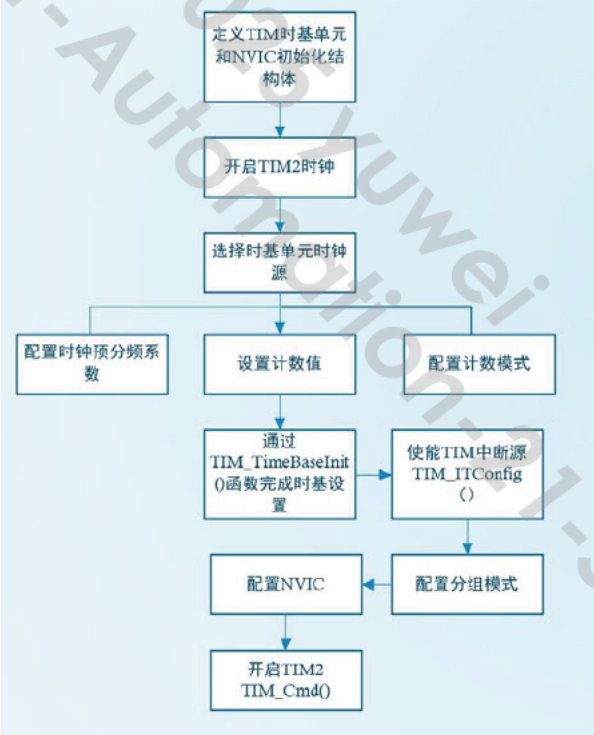

图 4　定时器延时模块配置流程图

该模块相关代码如下：

```
void Timer2_init(void)
{
 TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStru;
 NVIC_InitTypeDef NVIC_InitStru;
 RCC_APB2PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
 //定时 143ms
 TIM_InternalClockConfig(TIM2);
 TIM_TimeBaseInitStru.TIM_ClockDivision=TIM_CKD_DIV1;
 TIM_TimeBaseInitStru.TIM_CounterMode =TIM_CounterMode_Up;
```

```
    TIM_TimeBaseInitStru.TIM_Period=1430-1;    //自动重装值，ARR
    TIM_TimeBaseInitStru.TIM_Prescaler=7200-1; //预分频，10KHZ
    TIM_TimeBaseInitStru.TIM_RepetitionCounter=0;

    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStru);
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    NVIC_InitStru.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStru.NVIC_IRQChannelCmd =ENABLE ;
    NVIC_InitStru.NVIC_IRQChannelPreemptionPriority =1 ;
    NVIC_ nitS ru.NVIC_IRQChannelSubPriority =1 ;
    NVIC_Init(&NVIC_InitStru);

    TIM_Cmd(TIM2,ENABLE);
}
```

4、实验实训步骤

①准备工作

a. 准备好所需的材料和设备：STM32F103C8T6 单片机最小系统板、面包板、8 个直插 LED 灯、STLINK 下载器、杜邦线等

b. 将 STM32F103C8T6 单片机固定在洞洞板上，连接面包板，准备进行电路连接。

②电路搭建

a. 将 8 个 LED 灯连接到面包板上，确保 LED 长短腿分辨清楚。

b. 使用杜邦线将 LED 灯分别连接到 STM32F103C8T6 的 GPIO 引脚，确保正确对应引脚。

③编写程序和编译下载

a. 编写实验的初始化配置代码，将相应的 GPIO 引脚配置为输出模式。确保 GPIO 初始化代码正确，能够控制 LED 的开关状态。

b. 编写流水灯控制程序，通过循环控制每个 GPIO 引脚，使 LED 按顺序依次亮起和熄灭，形成流水灯效果。

c. 编写两种编程方法，利用定时器或者延时函数，确保每个 LED 灯保持亮起和熄灭状态的时间。调整延时函数或定时器，使得整个流水灯循环在 2 秒内完成。

d. 编译代码，确保没有语法错误和警告。将编译后的程序通过 STLINK 下载至 STM32F103C8T6 单片机。

④测试与调试

a. 通上电，观察 LED 灯的状态变化，检查 LED 按照预期顺序亮起和熄灭，以及持续时间是否符合要求。若出现问题，进行调试，检查电路连接和代码逻辑，逐步修正错误。

⑤实验报告撰写

a. 记录电路连接图和代码。

b. 分析实验过程中遇到的问题和解决方法。

c. 总结实验成果，进行技术和经验上的反思。

5、调试方法和过程记录

我采用的调试方法是每编写完一个模块的代码，就对该模块进行测试。这样可以缩小查错范围，有效提高效率。

①GPIO 初始化配置调试

首先编写 GPIO 初始化代码并确保正确配置 GPIO 引脚的模式。先从点亮一组灯开始，以确保 LED 初始化模块编写无误。观察并记录每个引脚的配置状态和输出情况，注意是否有异常状态出现。通过观察现象发现问题所在，再返回到程序上修改，最后经过测试 LED 初始化模块功能正常，能够按照预期工作。如下图 5 所示：

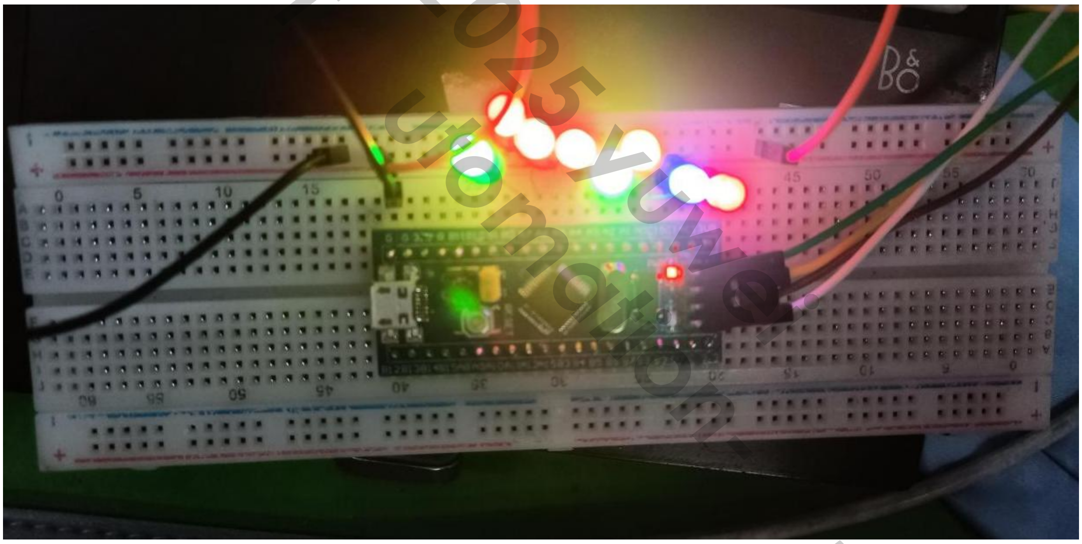

图 5 测试 LED 初始化模块

②LED 控制调试

其次通过编写控制 LED 灯依次亮起和灭掉的代码，来逐个检查 LED 灯的亮起和熄灭，确保按预期顺序进行。记录每个 LED 的点亮和熄灭情况，注意是否有灯未按预期工作。刚开始灯不流水，一上电就全亮，后来才发现是犯了低级错误没有调用 LED 控制相关的函数。在代码编写上，我一开始是写一个个移位的语句，后来发现可以放在循环里移位，这样就大大减少了代码量，也使得整个程序看上去简洁明了。以下是改进前后的代码对比图：

改进代码后：

```
1.  for(i=0;i<8;i++)
2.  {
3.  GPIO_Write(GPIOA,0x0001<<i); //0000 0001
4.  Delay_ms(143);
5.  }
6.  for(i=0;i<8;i++)
7.  {
8.  GPIO_Write(GPIOA,0x0001<<(8-i)); //0000 0001
9.  Delay_ms(143);
10. }
```

之前的代码：

```
1.  GPIO_Write(GPIOA,0x0001); //0000 0001
2.  Delay_ms(100);
3.  GPIO_Write(GPIOA,0x0002); //0000 0010
4.  Delay_ms(100);
5.  GPIO_Write(GPIOA,0x0004); //0000 0100
6.  Delay_ms(100);
7.  GPIO_Write(GPIOA,0x0008);  //0000 1000
8.  Delay_ms(100);
9.  GPIO_Write(GPIOA,0x0010);  //0001 0000
10. Delay_ms(100);
11. GPIO_Write(GPIOA,0x0020);  //0010 0000
12. Delay_ms(100);
13. GPIO_Write(GPIOA,0x0040);  //0100 0000
14. Delay_ms(100);
15. GPIO_Write(GPIOA,0x0080);  //1000 0000
16. Delay_ms(100);
17. GPIO_Write(GPIOA,0x0040);  //0100 0000
18. Delay_ms(100);
19. GPIO_Write(GPIOA,0x0020);  //0010 0000
20. Delay_ms(100);
21.   GPIO_Write(GPIOA,0x0010);  //0001 0000
22. Delay_ms(100);
23.   GPIO_Write(GPIOA,0x0008);  //0000 1000
24. Delay_ms(100);
25.   GPIO_Write(GPIOA,0x0004);  //0000 0100
26. Delay_ms(100);
```

图 6 测试 LED 控制模块

2、综合调试和运行仿真

在测试完各个功能模块后，接下来是将他们综合在一起，实现实验最终的流水灯效果。利用搭建好的电路，使用 ST-Link 下载器将程序烧录进 STM32 里面，测试整体流水灯效果，观察 LED 的依次点亮和熄灭，运行完整代码，确保持续时间为 2 秒。并记录流水灯效果的整体运行情况，注意是否有任何 LED 未按照顺序亮起和熄灭。下图 7 为实现的流水灯效果。

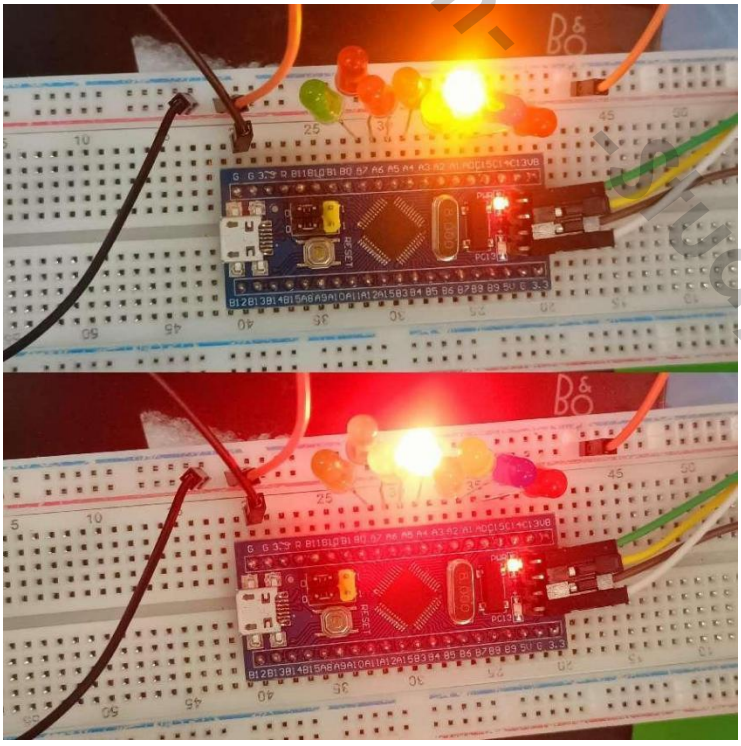

图 7　流水灯效果

在运行时，我还使用了 Keil uVision5 软件自带的 Debug 模式进行调试，通过这种方法可以看到所有变量的值和逻辑判断的走向以及是否跳转中断，方便修改程序。

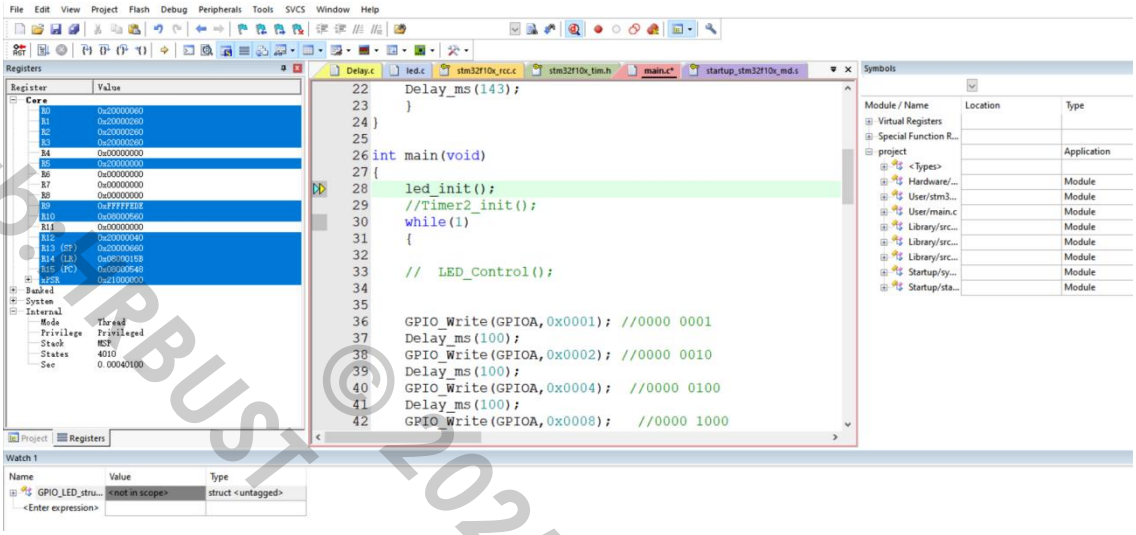

图 8　在 Debug 模式下调试

6、实验结论与结果分析

- 实验结果

通过认真完成实验，我成功地使用 STM32 最小系统板实现了 8 位流水灯的效果。也控制好了 LED 保持亮或灭状态的时间，确保了整体效果持续 2 秒。

- 结果分析

在本次实验中，我掌握了如何使用 GPIO 对其完成相应的初始化工作，通过编程实现 LED 的点亮和熄灭，掌握了 GPIO 驱动指示灯的方法。实验中编写的程序具有稳定性，LED 的控制和流水灯效果稳定可靠。通过这个实验，有助于理解 STM32 单片机的 GPIO 控制。

- 实验结论

本次实验我实现了 8 位流水灯效果，从硬件到软件的整个设计过程加深了我对 STM32 单片机 GPIO 控制的理解和掌握。同时，对于嵌入式系统的开发有了更深入的认识。虽然这次项目是 stm32 里一个较小的项目，但是这为我未来的嵌入式系统开发和单片机应用提供了宝贵的经验和技能。

（二）本项目是否涉及工程与社会、环境与可持续发展、职业规范等方面的考虑，若有，请简单说明。

本次实验是通过程序来实现 LED 流水灯的效果实验，在实际工程开发中，这样的项目可能是更复杂系统的一部分，如安全警示系统、指示灯控制等。对于这些系统，正确的硬件和软件设计至关重要，影响到社会的安全和健康。因此，掌握嵌入式系统的基础知识对于更复杂系统的开发是至关重要的。尽管这个项目难以直接与环境和可持续发展联系，但在更广泛的背景下，合理的电子产品设计和

制造如果能够考虑能源效率、资源利用和再利用，那将对环境保护和可持续发展有巨大促进作用。通过完成这样的实验，我们可以培养对工程职业规范的理解和尊重。这包括对工程职业道德和规范的遵守，以及在实际工程项目中负责任地处理问题和解决挑战。

四、项目总结与体会（12 分）

（1）项目的总体完成情况

在该项目中，我成功利用 STM32F103C8T6 单片机最小系统板和 8 个 LED 灯搭建了外围电路，实现了 8 位流水灯依次亮起、熄灭的效果。通过正确的 GPIO 初始化配置和编程实现，顺利达成了项目既定的目标。

（2）项目对（二）中各项具体目标和指标要求的达成情况

基本完成项目中的各项具体目标和指标要求，具体完成情况如下：

GPIO 初始化配置方法掌握：成功掌握了 STM32 单片机的 GPIO 初始化配置方法，能够准确配置引脚为输出模式，用于控制 LED。

GPIO 驱动指示灯方法掌握： 通过编程实现 LED 的控制，达到了 GPIO 驱动指示灯的要求。

8 位流水灯效果：实现了 8 位流水灯的效果，LED 按序亮起并熄灭，持续时间达到了预期的 2 秒。

（3）项目中尚存在的问题和改进方向

精确性与稳定性： 有时 LED 的亮灭状态可能会因硬件接触不良或程序问题出现不稳定情况。应当进一步改善硬件连接和程序逻辑，确保 LED 控制的精确性和稳定性。

代码优化： 需要对代码进行优化，提高程序的效率和可读性，以及减少资源占用。

（4）团队及成员的体会

本次项目的各项功能已经全部完成，各项指标都满足了。在做实验时我犯了不少错误，不过通过我仔细地检查后，修改几遍就能搞定。本课程的实验是一项非常注重细节的测试，能够锻炼我们的细心程度。

通过完成这个项目，我学到了很多在书本上学不到的知识，掌握了一种系统的研究方法,并对 STM32 单片机的 GPIO 初始化配置和 LED 控制有了更深入的理解。同时我也意识到，只有理论知识是远远不够的，要学会如何将理论知识转化为实际操作。我还并体会到调试过程中的挑战和乐趣，同时也意识到在嵌入式系统开发中，每一个细节都至关重要，需要不断调试和改进，才能得到理想的效果。

五、参考文献（6 分）

[1] 张淑清. 嵌入式单片机 STM32 原理及应用[M]. 机械工业出版社, 2019.
[2] 卢有亮. 基于 STM32 的嵌入式系统原理与设计[M] . 机械工业出版社, 2013.
[3] 熊刚, 陈高锋, 刘晨. 单片机控制的多路花式流水灯[J]. 电子世界,2015,0(13):89-89100

附录 1-实验 1 代码

```c
#include "Device/Include/stm32f10x.h"   // Device header
#include "Delay.h"
#include "led.h"

uint8_t i=0;
uint8_t reverse=0;

void Timer2_init(void);

void led_init(void)
{
 GPIO_Init ypeDef GPIO_LED_structure;
 RCC_APB2PeriphCl ckCmd(RCC_APB2Periph_GPIOA,ENABLE);
 GPIO_LED_structure.G IO_Mode GPIO_Mode_Out_PP;
 GPIO_LED_structure.GPIO_Pin=GPIO Pi _All;
 GPIO_LED_structure.GPIO_Speed=GPIO_ peed_50MHz;
 GPIO_Init (GPIOA,&GPIO_LED_structure);

 GPIO_SetBits(GPIOA,GPIO_Pin_All);
}

void LED_Control(void)
{

 for(i=0;i<8;i++)
 {
 GPIO_Write(GPIOA,0x0001<<i); //0000 0001
 Delay_ms(143);
 }
 for(i=0;i<8;i++)
 {
 GPIO_Write(GPIOA,0x0001<<(8-i)); //0000 0001
 Delay_ms(143);
 }
}
int main(void)
{
 led_init();
 //Timer2_init();
 while(1)
 {

 // LED_Control();
```

```
  /*
  GPIO_Write(GPIOA,0x0001); //0000 0001
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0002); //0000 0010
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0004);  //0000 0100
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0008);   //0000 1000
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0010);  //0001 0000
  Delay_m (100);
  GPIO_Write(GPIOA,0x0020);  //0010 0000
  Delay_m ( 00);
  GPIO_Write(GPIOA 0x0040);   //0100 0000
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0080);   //1000 0000
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0040);   //0100 0000
  Delay_ms(100);
  GPIO_Write(GPIOA,0x0020);  //0010  000
  Delay_ms(100);
     GPIO_Write(GPIOA,0x0010);  //0001 0000
  Delay_ms(100);
     GPIO_Write(GPIOA,0x0008);   //0000 1000
  Delay_ms(100);
     GPIO_Write(GPIOA,0x0004);  //0000 0100
  Delay_ms(100);
     GPIO_Write(GPIOA,0x0002); //0000 0010
  Delay_ms(100);
     GPIO_Write(GPIOA,0x0001); //0000 0001
  Delay_ms(100);
  }*/
 }
 }
void Timer2_init(void)
{
  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStru;
  NVIC_InitTypeDef NVIC_InitStru;
  RCC_APB2PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
  //定时143ms
  TIM_InternalClockConfig(TIM2);
  TIM_TimeBaseInitStru.TIM_ClockDivision=TIM_CKD_DIV1;
  TIM_TimeBaseInitStru.TIM_CounterMode =TIM_CounterMode_Up;
  TIM_TimeBaseInitStru.TIM_Period=1430-1;    //自动重装值，ARR
```

```
88.    TIM_TimeBaseInitStru.TIM_Prescaler=7200-1; //预分频，10KHZ
89.    TIM_TimeBaseInitStru.TIM_RepetitionCounter=0;
90.
91.    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStru);
92.    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
93.
94.    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
95.    NVIC_InitStru.NVIC_IRQChannel = TIM2_IRQn;
96.    NVIC_InitStru.NVIC_IRQChannelCmd =ENABLE ;
97.    NVIC_InitStru.NVIC_IRQChannelPreemptionPriority =1 ;
98.    NVIC_InitStru.NVIC_IRQChannelSubPriority =1 ;
99.    NVIC_ nit( NVIC_InitStru);
100.
101.   TIM_Cmd(TIM2 ENABLE);
102. }
103.
104. void TIM2_IRQHandler(voi )
105. {
106.   if(TIM_GetITStatus(TIM2,TIM_IT_Update)==SET)
107.   {
108.    GPIO_Write(GPIOA,0x0001<<1);
109.    /*
110.   if(reverse==0)
111.   {
112.    GPIO_Write(GPIOA,0x0001<<i++); //0000 0001
113.    if(i==7)
114.    {
115.     i=0;
116.     reverse=1;
117.    }
118.   }
119.   else
120.   {
121.    GPIO_Write(GPIOA,0x0001<<(8-i++)); //0000 0001
122.    if(i==7)
123.    {
124.     i=0;
125.     reverse=0;
126.    }
127.   }*/
128.   TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
129. }
130.  }
```