1 跑马灯实验（寄存器版本）

1.1 实验目的

通过代码控制 ALIENTEK 阿波罗 STM32 开发板上的两个 LED 灯 DS0 和 DS1 交替闪烁，实现类似跑马灯的效果。

修改代码，初步掌握 STM32F429 基本 IO 口的使用。

1.2 代码描述

main.c

```c
// 寄存器版本
#include "delay.h"
#include "led.h"
#include "sys.h"
int main(void) {
  Stm32_Clock_Init(360, 25, 2, 8);
  delay_init(180);
  LED_Init();
  int d = 5;
  int sist = 7;
  while (1) {
    for (int i = 0; i < d; ++i) {
      for (int _ = 0; _ < sist; ++_) {
        LED1 = 0;
        delay_ms(i);
        LED1 = 1;
        delay_ms(d * 4 - i);
      }
    }
    for (int i = d - 1; i > 0; --i) {
      for (int _ = 0; _ < sist; ++_) {
        LED1 = 0;
        delay_ms(i);
        LED1 = 1;
        delay_ms(d * 4 - i);
      }
    }
  }
}
```

用到了两个 for 循环，外层循环控制占空比，内层循环起到延时作用，使人眼能够观测到占空比变化。

变量 d 用来控制闪烁周期，sist 控制每个亮度持续的周期数。

1.3 实验结果

观察到 DS1 由暗到亮，再由亮到暗的循环闪烁。

1.4 心得体会

想要达到比较好的观察效果需要细致地调整参数。LED 的占空比在 0.3 以下时可以明显看出亮度变化，而在 0.4 以上时，LED 较亮，导致亮度变化不明显。

2 跑马灯实验（HAL 库版本）

2.1 实验目的

通过代码控制 ALIENTEK 阿波罗 STM32 开发板上的两个 LED 灯 DS0 和 DS1 交替闪烁，实现类似跑马灯的效果。

使用 HAL 库版本重写上述跑马灯程序，同样使 DS1 发生由暗到亮，再由亮到暗的循环闪烁。

2.2 代码描述

main.c

```c
// 与 1.c 相同效果，不过是库函数版本
#include "delay.h"
#include "led.h"
#include "sys.h"
#include "usart.h"
int main(void) {
  HAL_Init();                     // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8); // 设置时钟,180Mhz
  delay_init(180);                // 初始化延时函数
  LED_Init();
  int d = 5;
  int sist = 7;
  while (1) {
    for (int i = 0; i < d; ++i) {
      for (int _ = 0; _ < sist; ++_) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
        delay_ms(i);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        delay_ms(d * 4 - i);
      }
    }
    for (int i = d - 1; i > 0; --i) {
      for (int _ = 0; _ < sist; ++_) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
        delay_ms(i);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        delay_ms(d * 4 - i);
      }
    }
  }
}
```

2.3 实验结果

观察到了与实验 1 相同的跑马灯闪烁。

2.4 心得体会

了解了 HAL 库的使用方法，以及在 STM32 开发板上通过 HAL 库控制 LED 灯的方法。学习了如何使用 GPIO 端口和时钟，并逐步构建复杂的应用程序。

3 按键输入实验

3.1 实验目的

利用板载的 4 个按键，来控制板载的两个 LED 的亮灭，熟悉 STM32F4 的 IO 口作为输入口的使用方法。

3.2 代码描述

main.c

```c
#include "delay.h"
#include "key.h"
#include "led.h"
#include "sys.h"
#include "usart.h"
// 调整占空比，即工作时间与总时间的比值。函数需要放在循环中才可用。
void space_ratio(u8 delayms, float rate) {
  LED1 = 0;
  delay_ms((u8)(delayms * rate));
  LED1 = 1;
  delay_ms((u8)(delayms * (1 - rate)));
}
// 根据不同按键，让 LED 灯显示不同亮度。
int main(void) {
  u8 key;
  HAL_Init();                     // 初始化 HAL 库
  Stm32_Clock_Init(360, 25, 2, 8); // 设置时钟,180Mhz
  delay_init(180);                // 初始化延时函数
  uart_init(115200);              // 初始化 USART
  LED_Init();                     // 初始化 LED
  KEY_Init();                     // 初始化按键
  float f = 0;
  while (1) {
    key = KEY_Scan(0); // 按键扫描
    switch (key) {
    case WKUP_PRES:
      f = 0;
      break;
    case KEY2_PRES:
      f = f == 0.05f ? 0 : 0.05f;
      break;
    case KEY1_PRES:
      f = f == 0.2f ? 0 : 0.2f;
      break;
    case KEY0_PRES:
      f = f == 0.8f ? 0 : 0.8f;
      break;
    }
    space_ratio(20, f);
  }
}
```

将调节亮度的占空比调节过程抽象成 `space_ratio()` 函数，通过不同按键，控制灯的亮度。代码中 `f = f == x ? 0 : x;` 的设计，目的是附加一个功能，在第二次按下按钮时熄灭 LED。

3.3 实验结果

通过不同按键，实现 LED 的亮度调节。按动 WKUP 键后关闭 LED，按动 KEY2 后调整亮度为低，而按动 KEY1 后亮度调整为较高，按动 KEY0 后亮度调整为高。

3.4 心得体会

由于占空比在 0.3 以上时亮度变化不明显，因此设置占空比为 0, 0.05, 0.2, 0.8，可以很好地观察到 LED 的亮度变化。

经过测试，使用 KEY_Scan(0) 只能监测到按键按下时的一次信号发送。如果需要监测按键持续时间或 release 时机，可能需要修改库函数代码。

4 外部中断实验

4.1 实验目的

学习如何使用 STM32F429 的外部输入中断，了解中断优先级的设置方法与表现，学习将中断线映射为 GPIO 的方法，在中断服务函数中处理逻辑。

4.2 代码描述

exti.c 片段

```
void space_ratio(u8 delayms, float rate) {
  LED0 = 0;
  delay_ms((u8)(delayms * rate));
  LED0 = 1;
  delay_ms((u8)(delayms * (1 - rate)));
}
void space_ratio(u8 delayms, float rate) {
  LED1 = 0;
  delay_ms((u8)(delayms * rate));
  LED1 = 1;
  delay_ms((u8)(delayms * (1 - rate)));
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  delay_ms(100);
  switch (GPIO_Pin) {
  case GPIO_PIN_0:
    if (WK_UP == 1) {
      for (int j = 0; j < 100; j++) {
        space_ratio(20, (float)j / 100.0 * 1.8);
      }
      for (int j = 99; j > 0; --j) {
        space_ratio(20, (float)j / 100.0 * 1.8);
      }
    }
    break;
  case GPIO_PIN_2:
    if (KEY1 == 0) {
      LED1 = !LED1;
    }
    break;
  case GPIO_PIN_3:
    if (KEY0 == 0) {
      for (int j = 0; j < 100; j++) {
        space_ratio2(20, (float)j / 100.0 * 1.8);
      }
      for (int j = 99; j > 0; --j) {
        space_ratio2(20, (float)j / 100.0 * 1.8);
      }
    }
    break;
  case GPIO_PIN_13:
    if (KEY2 == 0) {
      LED0 = !LED0;
    }
    break;
  }
}
```

通过改进的 `space_ratio()` 函数实现了双 LED 灯的亮度调节，通过按键按照优先级进入中断程序，执行对应的服务。

4.3 实验结果

在本次的实验中，使用外部中断来实现控制灯的暗灭。利用占空比的原理实现了让灯从暗至亮再从亮至暗的过程。

4.4 心得体会

由于不熟悉串口调试工具的使用，在串口调试时遇到了麻烦。需要接对串口线，选中正确的波特率，才可以看到正常输出。选择了错误的波特率可能导致接收的信息乱码。

一开始发现灯频闪的很快，而不是缓慢变亮，因为频率慢，没有让灯的图像在人眼中滞留，后增加了频率改善了这一问题。但由于未知原因没有实现中断优先级，即不能用其他优先级高的按钮中断当前灯的状态。