厦门大学

XIAMEN UNIVERSITY

嵌入式系统开发

实验一 控制 LED 灯

姓 名	
学 号	
学 院	信息学院
专 业	计算机科学与技术专业
年 级	2022
日 期	2024/11/15

目录

1 实验目的

（1）掌握 GPIO 的原理与应用，学会使用 GPIO 控制 LED 灯的亮灭。

（2）掌握 I/O 的配置和按键驱动的方法，掌握轮询和中断两种工作模式的原理。

2 实验方案设计

2.1 基础实验

（1）实验要求

①通过配置 GPIO 点亮实验箱上的 LED 灯。

②使用拨码开关点亮实验箱上的 LED 灯。

③使用轮询方式，用按键开关点亮 LED 灯。

④使用中断方式，用按键开关点亮 LED 灯。

（2）实验步骤

步骤一：仿照实验一中新建工程模板的方式新建 02_GPIO_Runing_LED 的工程文件，添加 GD32F4xx_XII-IOT.h 头文件。在该头文件中已经定义好 8 个 LED、4 个按键开关以及 8 个波码开关的 GPIO 口的配置，调用 GD32F4xx 的库函数设置 GPIO 口的输出方式，进行对应的初始化函数封装。其中 LED 的 GPIO 口设置成上拉推挽输出模式，按键的 GPIO 口设置成输入方式，拨码开关的 GPIO 口设置成输入方式。

步骤二：在 GD32F4xx_it.c 文件中配置按键中断处理函数，设置好按键 K2 和 K3 的中断处理函数,K2 控制 LED 的灭，K3 控制 LED 的亮。

步骤三：硬件上需要把拨码开关 GPIO 复用的跳线帽连上，如图 2.1 所示。编译相对应的代码程序将其下载到实验箱中测试。

图 2.1 波动开关复用位置

2.2 提高实验

（1）实验要求

①使用拨码开关控制 LED 灯的亮灭：

当拨码开关 S1 拨为 1 时，若将拨码开关 S2 也拨为 1 则 LED 灯顺序亮起，将拨码开关 S2 拨为 0、拨码开关 S3 拨为 1 则灯顺序熄灭，若拨码开关 S2、S3 同时拨为 1 灯亮起；若拨码开关 S1 拨为 0，则无论拨码开关 S2、S3 为何值 LED 灯均不亮起。对应的逻辑表如下表 2.1 所示：三个数字分别代表拨码开关的开关状态，1 代表开 0 代表关 x 代表任意状态如 110 表示 S1 开 S2 开 S3 关 ABCD 分别表示四种 LED 灯状态：A：1-8 灭/B：1-8 顺序亮/C：1-8 顺序灭/D：1-8 亮。

表 2.1 拨码与灯状态逻辑关系表

拨码 1	拨码 2	拨码 3	灯状态
0	X	X	都不亮（A）
1	1	0	1-8 顺序亮起(B)
1	0	1	1-8 顺序熄灭(C)
1	1	1	1-8 亮(D)
1	0	0	1-8 灭(A)

②使用按键开关控制 LED 灯的亮灭：将任一 LED 灯作为指示灯。

按下按键 1，所有 LED 灯亮起；

而后按下按键 4，除指示灯外所有灯间隔闪烁；

再次长按按键 4，灯亮时长即为按键时长；

最后按下按键 1，包括指示灯在内的所有灯熄灭，此时按下按键 2，LED 灯不亮起。

（2）实验步骤

步骤：在 test.c 中编写拨码开关的轮询函数和按键开关的中断处理函数，在 void test2（void）按实验要求编写相应的程序，在 main 函数中注释掉 test1，填入 test2 下载程序到电路板中测试。

2.3 拓展实验

（1）实验要求

①设置灯的工作模式：

使用实验箱上的 8 个 LED 灯其中一个灯作为实验开始指示灯，当实验开始后即保持常亮。一个灯作为运行过程指示灯（报错灯），在工作过程中，灯的工作方式改变则运行指示灯会闪烁。其余灯的工作方式可为表 1.2.4 所示。

表 1.2.4

1	依次亮（亮后不灭）
2	依次缓慢亮（亮后不灭）
3	流水亮（亮起一定时间后灭）
4	一起闪烁
5	间隔闪烁

6	依次灭
7	依次缓慢灭

选择上述工作方式中的至少三种作为 LED 灯的工作方式并将其排序。

②设置拨码开关：

若拨码开关拨成 1 的个数为奇数个，则将所有灯的工作方式置为工作方式 1；若拨码开关拨成 1 的个数为偶数个，则将工作方式+1。

③设置按键开关：

若按下按键 1，则所有灯保持当前状态不变（工作方式、亮灭均不变）；

再按下按键 1 则恢复；

若按下按键 2，所有灯停止工作，除开始指示灯外其他灯灭；

再次按下按键 2 则所有灯按工作方式 1 开始工作；

若连续两次按下按键 2 则包括开始指示灯在内的所有灯灭，且再次按键、更改拨码开关无反应。

3 实验过程与结果

3.1 基础实验

3.1.1 通过配置 GPIO 点亮实验箱上的 LED 灯

首先在 main 函数中中调用 systeminit()函数，执行初始化

```
void system_init(void) {
    gd_XII_systeminit();//初始化 8 个 LED 灯
    dipinit();//初始化 8 个拨码开关
    gd_eval_key_init(USER1_KEY, KEY_MODE_GPIO);//K1 初始化成 IO 模式
    gd_eval_key_init(USER2_KEY, KEY_MODE_EXTI);//K2 初始化成中断模式
    gd_eval_key_init(USER3_KEY, KEY_MODE_EXTI);//K3 初始化成中断模式
    gd_eval_key_init(USER4_KEY, KEY_MODE_GPIO);//K4 初始化成 IO 模式
}
```

在 test1 函数中，通过调用 gd_eval_led_on(LED1);点亮 LED1。

```
#if TEST1_1 > 0
//基础实验 1：通过配置 GPIO 点亮实验箱上的 LED 灯 1；
gd_eval_led_on(LED1);
#endif
```

3.1.2 使用拨码开关点亮实验箱上的 LED 灯

在 test1 函数中，通过调用 `gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)`判断拨

码开关的状态，为 SET 打开则点亮 LED1，为 RESET 关闭则熄灭 LED1。

```
#if TEST1_2 > 0
    //基础实验 2：使用拨码开关点亮实验箱上的 LED 灯；
if(gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)==SET){
    gd_eval_led_on(LED1);
}
if(gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)==RESET){
    gd_eval_led_off(LED1);
}
#endif
```

3.1.3 使用轮询方式，用按键开关点亮 LED 灯

用 gd_eval_key_state_get(USER1_KEY)函数获取按键状态，如果为 RESET 的话，则调用 gd_eval_led_toggle(LED1);

```
#if TEST1_3 > 0
//基础实验 3：使用轮询方式，用按键开关点亮 LED 灯；
if(RESET==gd_eval_key_state_get(USER1_KEY)){
  if(RESET==gd_eval_key_state_get(USER1_KEY)){
      gd_eval_led_toggle(LED1);
  }
}
#endif
```

这里延时函数有误，会导致阻塞现象的发生。所以取消掉了按键消抖。

3.2 提高实验

3.2.1 使用拨码开关控制 LED 灯的亮灭

gpio_input_bit_get(DIPx_GPIO_PORT, DIPx_PIN): 读取指定拨码开关的状态。DIPx_GPIO_PORT 是拨码开关所在的 GPIO 端口，DIPx_PIN 是拨码开关对应的引脚。函数返回 SET 表示开关处于按下状态，返回 RESET 表示开关未被按下。

my_delay 函数：因为给的 delay_ms 函数不知道为什么会阻塞，所以只能自己写一个 my_delay 函数。

```
void my_delay(void) {
    volatile uint32_t i;
    for(i = 0; i < 1000000; i++) {
        __NOP();
    }
}
```

lightLED():顺序点亮所有 LED 灯，从 LED1 到 LED8。

extinguishLED():顺序熄灭所有 LED 灯，从 LED1 到 LED8。

lightLEDImmediately():立即点亮所有 LED 灯。

extinguishLEDImmediately():立即熄灭所有 LED 灯。

```
void lightLED(void){
    gd_eval_led_on(LED1);
    my_delay();
    gd_eval_led_on(LED2);
    my_delay();
    gd_eval_led_on(LED3);
    my_delay();
    gd_eval_led_on(LED4);
    my_delay();
    gd_eval_led_on(LED5);
    my_delay();
    gd_eval_led_on(LED6);
    my_delay();
    gd_eval_led_on(LED7);
    my_delay();
    gd_eval_led_on(LED8);
    my_delay();
}
void lightLEDImmediately(void){
    gd_eval_led_on(LED1);
    gd_eval_led_on(LED2);
    gd_eval_led_on(LED3);
    gd_eval_led_on(LED4);
    gd_eval_led_on(LED5);
    gd_eval_led_on(LED6);
    gd_eval_led_on(LED7);
    gd_eval_led_on(LED8);
}
```

```
void extinguishLED(){
    gd_eval_led_off(LED1);
    my_delay();
    gd_eval_led_off(LED2);
    my_delay();
    gd_eval_led_off(LED3);
    my_delay();
    gd_eval_led_off(LED4);
    my_delay();
    gd_eval_led_off(LED5);
    my_delay();
    gd_eval_led_off(LED6);
    my_delay();
    gd_eval_led_off(LED7);
    my_delay();
    gd_eval_led_off(LED8);
    my_delay();
}
void extinguishLEDImmediately(){
    gd_eval_led_off(LED1);
    gd_eval_led_off(LED2);
    gd_eval_led_off(LED3);
    gd_eval_led_off(LED4);
    gd_eval_led_off(LED5);
    gd_eval_led_off(LED6);
    gd_eval_led_off(LED7);
    gd_eval_led_off(LED8);
}
```

DIP1 拨为 0： 所有 LED 灯立即熄灭。

DIP1 拨为 1，DIP2 拨为 1 且 DIP3 拨为 0： LED 灯从 LED1 到 LED8 顺序亮起。

DIP1 拨为 1，DIP2 拨为 0 且 DIP3 拨为 1： LED 灯从 LED1 到 LED8 顺序熄灭。

DIP1 拨为 1，DIP2 和 DIP3 同时拨为 1： 所有 LED 灯立即亮起。

DIP1 拨为 1，DIP2 和 DIP3 为任意状态： 所有 LED 灯立即熄灭。

```
//进阶实验 1：使用拨码开关控制 LED 灯的亮灭
//当拨码开关 1 拨为 1 时，将拨码开关 2 也拨为 1 则 LED 灯顺序亮起；
//将拨码开关 2 拨为 0、拨码开关 3 拨为 1 则灯顺序熄灭；
//若拨码开关 2、3 同时拨为 1 灯亮起；
//若拨码开关 1 拨为 0，则无论拨码开关 2、3 为何值 LED 灯均不亮起。
if(gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)==RESET){
        extinguishLEDImmediately();
}
else{
    if(gpio_input_bit_get(DIP2_GPIO_PORT,DIP2_PIN)==SET
        &&gpio_input_bit_get(DIP3_GPIO_PORT,DIP3_PIN)==RESET)
            lightLED();
    else if(gpio_input_bit_get(DIP2_GPIO_PORT,DIP2_PIN)==RESET
        &&gpio_input_bit_get(DIP3_GPIO_PORT,DIP3_PIN)==SET)
            extinguishLED();
    else if(gpio_input_bit_get(DIP2_GPIO_PORT,DIP2_PIN)==SET
        &&gpio_input_bit_get(DIP3_GPIO_PORT,DIP3_PIN)==SET)
        lightLEDImmediately();
    else
        extinguishLEDImmediately();
}
```

3.2.2 使用按键开关控制 LED 灯的亮灭

lightKey1 和 lightKey4: 这两个整型变量用于跟踪按键 1 和按键 4 被按下的次数。

key1 和 key4: 这两个整型变量用于存储按键 1 和按键 4 的当前状态，防止抖动。

key4_press_time: 这个无符号整型变量用于记录按键 4 被长按时的时间戳。

```
int lightKey1=0;
int lightKey4=0;
int key1=SET;
int key4=SET;
uint32_t key4_press_time = 0;
```

```
#if TEST2_2 > 0

//  进阶实验 2：使用按键开关控制 LED 灯的亮灭：
//  使用按键开关控制 LED 灯的亮灭：将任一 LED 灯作为指示灯。
//  按下按键 1，所有 LED 灯亮起；
//  而后按下按键 4，除指示灯外所有灯间隔闪烁；
//  再次长按按键 4，灯亮时长即为按键时长；
//  最后按下按键 1，包括指示灯在内的所有灯熄灭，此时按下按键 4，LED 灯也不亮起。
//  此例程将 LED1 作为指示灯，

    //gd_eval_led_on(LED8);

    if(RESET==gd_eval_key_state_get(USER1_KEY)){
        if(key1==SET){
            lightKey1++;
            key1=RESET;
        }
    }
    else{
        key1=SET;
    }
    if(RESET==gd_eval_key_state_get(USER4_KEY)){
        if(key4==SET){
            lightKey4++;
            key4=RESET;
        }
    }
    else{
        key4=SET;
    }

        if(lightKey1==1){
            if(lightKey4==0){
                lightLEDImmediately();
            }
            else if(lightKey4==1){
                lightLEDImmediately();
                my_delay();
                extinguishLEDImmediately();
                my_delay();
            }
            else{
                extinguishLEDImmediately();
                gd_eval_led_on(LED1);
```

```
            }
        }
        else if(lightKey1>1){
            if(lightKey4>1){
                extinguishLEDImmediately();
            }
        }
#endif
```

结果：

按下按键 1 一次： 所有 LED 灯立即亮起，LED1 作为指示灯保持亮起状态。

再按下按键 4 一次： 除指示灯 LED1 外，其余 LED 灯开始间隔闪烁。

长按按键 4： 指示灯 LED1 保持亮起，其余 LED 灯闪烁的持续时间与按键 4 的长按时间相等。

再次按下按键 1： 包括指示灯 LED1 在内的所有 LED 灯熄灭。

在所有灯熄灭的状态下按下按键 4： 没有任何 LED 灯亮起。

3.3 拓展实验

通过不同的模式来控制 LED 灯的状态。程序中定义了三个 LED 控制函数：doubleLed、randomLed 和 breathLed

doubleLed()

通过改变全局变量 led_mode 的值来切换 LED 的状态。led_mode 是一个 uint32_t 类型的变量，初始值为 0。

每次调用 doubleLed 函数时，led_mode 的值会增加 1，并且取模 2，这样 led_mode 的值会在 0 和 1 之间循环。

根据 led_mode 的值，LED1、LED2、LED3、LED4、LED5、LED6、LED7 和 LED8 的状态会交替变化。当 led_mode 为 1 时，LED1、LED3、LED5 和 LED7 关闭，LED2、LED4、LED6 和 LED8 打开；反之亦然。

randomLed()

通过循环随机地打开或关闭 8 个 LED 灯。使用 rand()&1 来生成一个随机的 0 或 1，以此来决定每个 LED 的状态。

breathLed()

实现的是呼吸灯效果。通过改变变量 i 的值来控制 LED 的亮度（实际上是亮灭的时间），i 的值在 1 到 100 之间变化，形成呼吸效果。（占空比）

up 变量用来控制 i 是增加还是减少，从而实现呼吸灯的亮起和熄灭过程。

lightLEDImmediately()和 extinguishLEDImmediately()函数被用来立即改变 LED 的状态（亮灭），而 my_delay_ms(i)和 my_delay_ms(100-i)用来控制 LED 亮和灭的时间。

test3()

通过读取拨码开关的状态来决定调用哪个 LED 控制函数。

如果打开拨码 3，则调用 randomLed 函数，随机亮起一次灯。

如果打开拨码 1，则调用 breathLed 函数，打开呼吸灯模式。

如果打开拨码 2，则调用 doubleLed 函数，打开双闪灯模式。

如果打开拨码 4 被设置，则再次调用 randomLed 函数，随机闪烁灯。

如果没有任何 DIP 开关被设置，则调用 extinguishLEDImmediately()函数来关闭所有 LED。

```
1.
    uint32_t led_mode=0;
2.  void doubleLed(){
3.      led_mode=(led_mode+1)%2;
4.      if(led_mode==1){
5.          gd_eval_led_off(LED1);
```

```
            gd_eval_led_on(LED2);
            gd_eval_led_off(LED3);
            gd_eval_led_on(LED4);
            gd_eval_led_off(LED5);
            gd_eval_led_on(LED6);
            gd_eval_led_off(LED7);
            gd_eval_led_on(LED8);
        }
        else{
            gd_eval_led_on(LED1);
            gd_eval_led_off(LED2);
            gd_eval_led_on(LED3);
            gd_eval_led_off(LED4);
            gd_eval_led_on(LED5);
            gd_eval_led_off(LED6);
            gd_eval_led_on(LED7);
            gd_eval_led_off(LED8);
        }
        my_delay();
    }
    void randomLed(){
        uint32_t i;
        for(i=0;i<8;i++){
            if(rand()&1){
                gd_eval_led_on(i);
            }
            else{
                gd_eval_led_off(i);
            }
        }
    }
    uint32_t last_rand=RESET;
    void breathLed(){
        uint32_t i,j,up;
        i=j=up=1;
        while((gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)==SET)){
            if(up){
                i++;
                if(i>=100){
                    up=0;
                }
            }
            else{
                i--;
```

```
            if(i<=1){
                up=1;
            }
        }
        lightLEDImmediately();
        my_delay_ms(i);
        extinguishLEDImmediately();
        my_delay_ms(100-i);
    }
}
void test3(void) {
        if(gpio_input_bit_get(DIP3_GPIO_PORT,DIP3_PIN)==SET){
                if(last_rand==RESET){
                    randomLed();
                }
                last_rand=SET;
                return;
     }
        last_rand=RESET;
     if(gpio_input_bit_get(DIP1_GPIO_PORT,DIP1_PIN)==SET){
                breathLed();

    }
     else if(gpio_input_bit_get(DIP2_GPIO_PORT,DIP2_PIN)==SET){
        doubleLed();
     }
         else if(gpio_input_bit_get(DIP4_GPIO_PORT,DIP4_PIN)==SET){
        randomLed();
                my_delay();
     }
else{
    extinguishLEDImmediately(); }
}
```

4 实验分析

通过本次实验，我对嵌入式系统开发的 GPIO 操作有了了解。

GPIO 操作

实验中，我学习了如何通过编程配置 GPIO 端口来控制 LED 灯的亮灭。通过设置 GPIO 为输出模式，并使用特定的函数 gd_eval_led_on()和 gd_eval_led_off()来控制 LED 灯的状态，我理解了 GPIO 在嵌入式系统中的基础应用。

输入设备读取

实验涉及到了对拨码开关和按键状态的读取。通过拨码开关 gpio_input_bit_get()和按键开关 gd_eval_key_state_get()函数，我学会了如何检测外部输入设备的状态，并根据这些状态来改变系统的行为

轮询与中断

虽然中断没有要求做中断，但是我查询到两种模式的优缺点，轮询方式简单直观，但可能会导致 CPU 资源的浪费；而中断方式则可以更高效地响应外部事件，但编程复杂度更高。

延时处理

在实现按键长按功能时，我遇到了延时处理的问题。由于系统提供的毫秒级延时函数运行效果不正确，我通过编写 my_delay()函数来模拟延时。

5 实验总结

本次实验主要是关于按键、拨码开关和 LED 灯的。通过这次实验，我掌握了 GPIO 的原理与应用，学会使用 GPIO 控制 LED 灯的亮灭，掌握 I/O 的配置和按键驱动的方法，掌握轮询和中断两种工作模式的原理。

我项目配置遇到了一些困难。虽然软件以前安装过了，但是在实验初期，我花费了大量时间配置工程和理解 GPIO 库函数的使用。而且文件路径和老师放的有出入，所以还要打开 options for target，修改 Include path，并且修改 project items。

在代码调试过程中，由于缺乏调试工具，我只能通过 LED 灯的状态来推断程序的执行流程。

修改代码中宏定义的功能，平时没有用过，积累了新的编程知识。但是我认为宏定义修改起来还是麻烦了，每次修改还要重新 build。在实际的嵌入式开发当中，应该用输入设备来修改选择的 mode。

在处理按键输入时，因为按键的物理震动，应该通过一个延时函数实现消抖，以提高系统的稳定性和响应准确性。然而这里的功能是按下即亮、松开即暗，所以没有什么消抖的必要。如果是按一次亮起，再按一次灭掉，就必须使用按键消抖。在实际操作中，也可以通过模拟电路进行消抖。

延时函数出问题了，所以自己写一个延时函数。我查到__NOP();指令不执行任何实际的操作，但它会占用一个 CPU 的时钟周期。这意味着在执行到 __NOP()时，CPU 会花费一个周期的时间来执行这个“无操作”的指令。所以可以用这个来延时。

在最后想实现随机灯的效果，想用 srand 做随机数种子，结果意外发现如果使用随机数种子，就会使整个程序卡死，无法运行。我查询到，在嵌入式系统中，time()函数可能并不总是可用的，或者其实现可能有问题。特别是在 MDK（Keil）环境中，time()函数可能无法正常工作，因为它依赖于系统的时钟，而 STM32 可能没有配置或不支持标准的时钟服务。

总之，探索的过程中能学到很多。