STM32 MCU Development

STM32 单片机开发

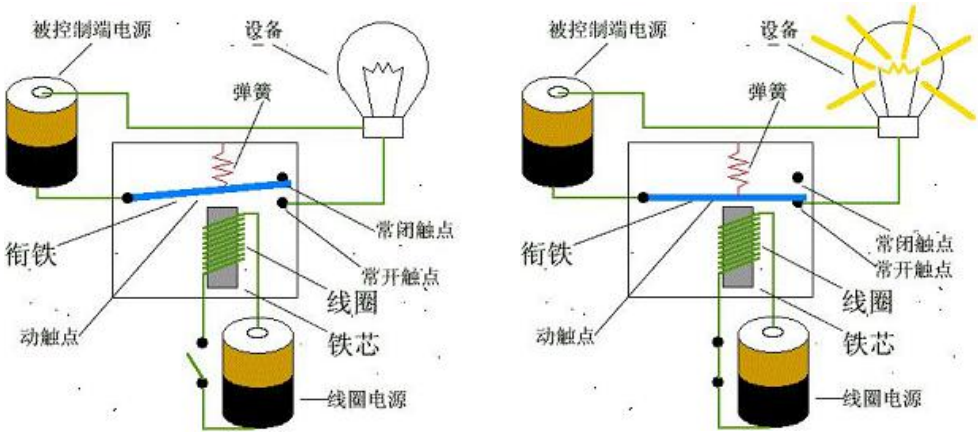
-- 阶段项目--楼道声、光控灯实现

目录

1. 继电器使用

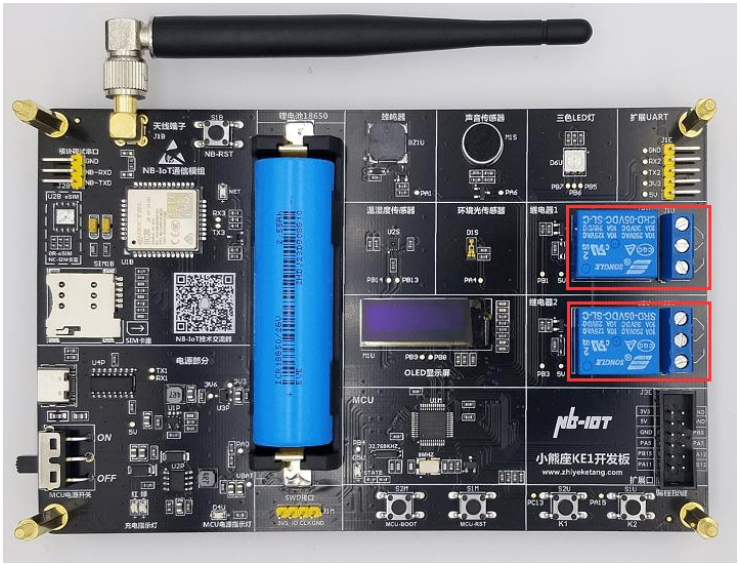
1.1. 继电器工作原理

继电器，一般指的是电磁继电器，也就是机械动作那种。它的基本原理，是利用了电磁效应来控制机械触点达到通断目的，给带有铁芯线圈通电-线圈电流产生磁场-磁场吸附衔铁动作通断触点，整个过程是“小电流-磁-机械-大电流”这样一个过程。



如上图所示，继电器有常开触点和常闭触点，动触点就是一个公共端，这是直流继电器，也就是继电器的线圈通过直流电时候（图上利用电池供电），带铁芯的线圈会输出对应的磁场，把衔铁吸附住，动触点一边会从常闭触点这边，跑到常开触点那边去，相当于常开触点吸合了。从图上来，启动/停止按钮，电池，继电器线圈形成了一个控制回路，只要这个回路吸合，线圈就会有电流通过，同时产生磁场。

如下图所示，小熊座 NB-IoT 开发板上提供了两个继电器，其输出控制能力为输出控制能力： 250V AC-10A、125V AC-10A、30V DC-10A、28V DC-10A。



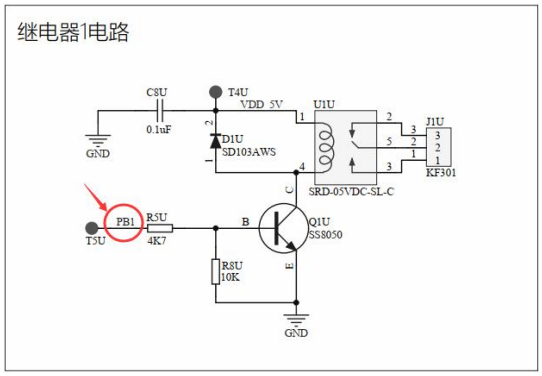
1.2. 灯泡与继电器连接

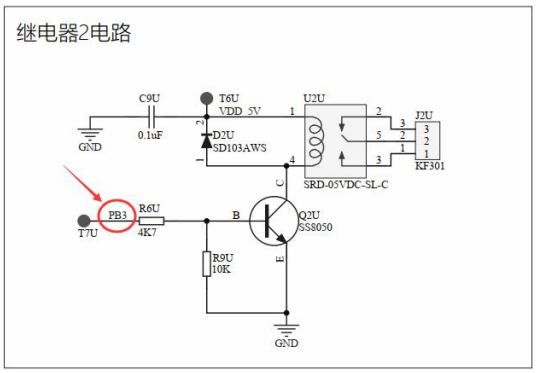
我们知道我们的两相电 220V 由零线(N)和火线(L)两根线来提供供电，此时我们将零线接到 Led 灯泡的灯座上，插头和灯座的火线则分别接到继电器的两个常开触点上。这样我们将插头插到插座上灯泡并不会亮，因为继电器的两个常开触点连接的火线处于断开状态，此时 220V 并没有形成回路则灯泡依然处于熄灭状态。

如果我们想要让灯亮，则可以通过编程控制继电器，让它的两个常开触点闭合，这样就能够形成 220V 的交流电回路，此时灯泡就亮了。

1.3. 原理图分析

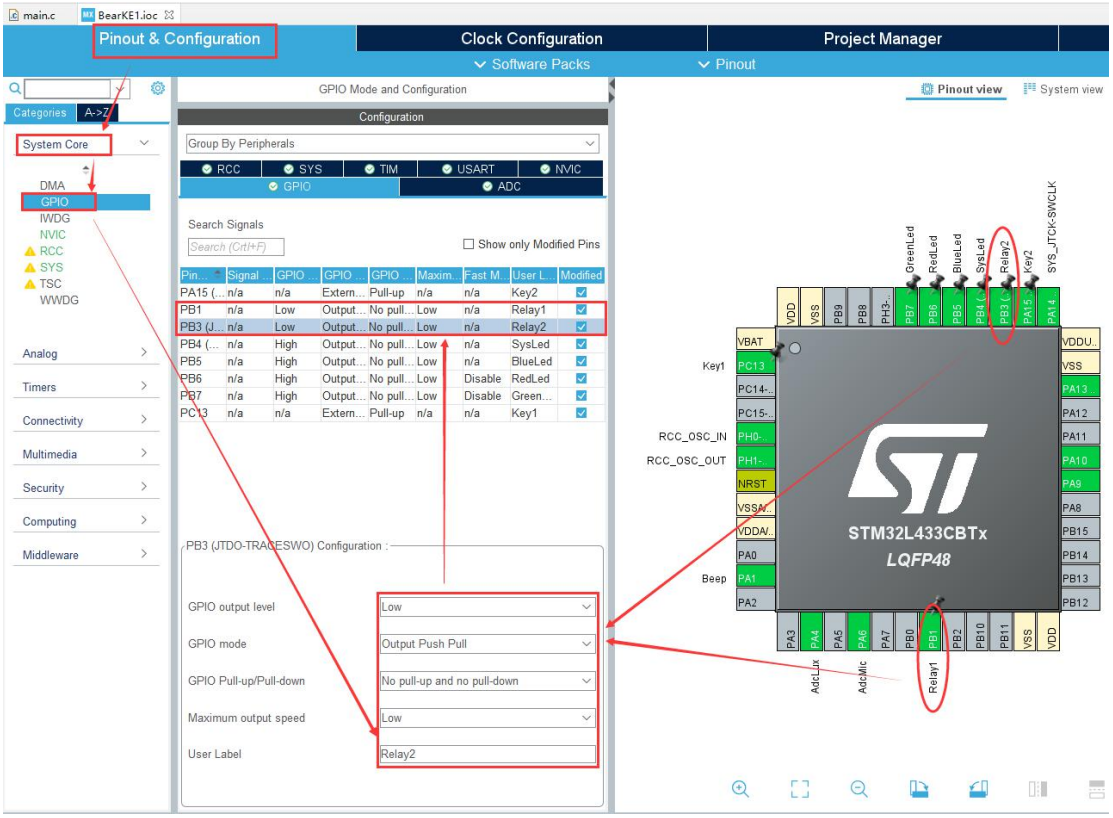
下面是小熊座 NB-IoT 开发板的继电器原理图，从原理图可知这两个继电器分别受 PB1 和 PB3 两个管脚控制，其中高电平有效。





1.4. STM32CubeIDE 配置

继电器跟 Led 灯一样，通过 GPIO 口输出高低电平来控制接触点的闭合与断开。接下来我们配置这两个管脚为 GPIO 输出模式，并配置默认电平为低电平。



配置好后，任意位置按 Ctrl+S 将会自动重新生成代码。

1.5. 源码修改

头文件 gpio.h 里的合适位置添加继电器的 enum 定义及控制函数声明，如下所示：

```
... ...
/* USER CODE BEGIN Prototypes */
enum
{
    Relay1,
    Relay2,
    RelayMax,
};

extern void turn_relay(int which, int status);
/* USER CODE END Prototypes */
```

c 文件 gpio.c 里添加继电器操作函数 turn_relay()函数定义，如下所示：

```
... ...
/* USER CODE BEGIN 2 */

typedef struct gpio_s
{
    GPIO_TypeDef       *group;
    uint16_t            pin;
} gpio_t;

gpio_t    relays[RelayMax] =
{
    { Relay1_GPIO_Port,   Relay1_Pin},
    { Relay2_GPIO_Port,   Relay2_Pin},
};

void turn_relay(int which, int status)
{
    GPIO_PinState       level;

    if( which >= LedMax )
        return ;

    level = status==OFF ? GPIO_PIN_RESET : GPIO_PIN_SET;

    HAL_GPIO_WritePin(relays[which].group, relays[which].pin, level);
}
```

接下来我们将会调用该函数来实现继电器控制灯泡的亮灭。

1.6. 继电器控制灯泡测试

c 文件 gpio.c 里修改按键中断服务处理程序，通过这两个按键来控制灯泡的亮灭：

```
... ...

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    static uint8_t    relay1_status = OFF;
    static uint8_t    relay2_status = OFF;


    if( Key1_Pin == GPIO_Pin )
    {
        relay1_status ^= 1;
        turn_relay(Relay1, relay1_status);
    }
    else if( Key2_Pin == GPIO_Pin )
    {
        relay2_status ^= 1;
        turn_relay(Relay2, relay2_status);
    }
}
```

1.7. 运行测试

源码编译、烧录及运行，这时可以分别通过 Key1、Key2 按键来控制继电器 Relay1、Relay2，从而控制灯泡的亮灭。

2. 楼道灯实现

前面我们讲解了光强传感器和声音传感器的采样，也讲解了继电器的使用，接下来我们就可以使用它们实现楼道的声、光控灯了。如果大家没有继电器和灯泡的话，也可以使用开发板上自带的三色 Led 来模拟实验。

```
... ...

int main(void)
{
  /* USER CODE BEGIN 1 */
     uint32_t         lux, noisy;
     uint32_t         start = 0;
     uint8_t          light_status = 0;
  /* USER CODE END 1 */

     ... ...

  /* USER CODE BEGIN WHILE */
  sysled_hearbeat();
  beep_start(2, 300);
  printf("Start BearKE1 5G NB-IoT Board Example Program v1.0\r\n");

  while (1)
  {
       if( OFF == light_status )
       {
            adc_sample_lux_noisy(&lux, &noisy);
            printf("Lux[%lu] Noisy[%lu]\r\n", lux, noisy);

            if( lux<30 && noisy>800 )
            {
                 printf("Turn Light on\r\n");
                 turn_relay(Relay2, ON);
                 //turn_led(RedLed, ON);
                 light_status = ON;

                 /* record current time and will turn off in 15 seconds */
                 start = HAL_GetTick();
            }
       }
```

```
      else
      {
           if( time_after(HAL_GetTick(), start+15000) )
           {
                printf("Turn Light off\r\n");
                turn_relay(Relay2, OFF);
                //turn_led(RedLed, OFF);

                /* must give delay for turn relay off got noisy, it will affect next sample */
                HAL_Delay(200);

                /* set light status as OFF */
                light_status = OFF;
           }
      }

      HAL_Delay(10);

   /* USER CODE END WHILE */

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

在上面的代码中，我们通过 ADC 采样当前的光强和噪音值，如果光线较弱并且声音比较大的话就控制继电器开灯。但灯泡不能一直开着，所以做了一个时间的控制，15s 后将自动关灯。

在 15s 的控制逻辑实现里，我们并没有直接用 HAL_Delay(1500)后关灯，这是因为如果直接延时的话，程序就不能再做其它的控制了。在这里的代码实现里，我们在点灯的时候记录当前时间，然后循环获取当前时间看有没有超时，如果超时则控制继电器关灯。

在判断时间是否到没有，我们不能直接 if((HAL_GetTick() - start) - 15000) 这个条件判断，因为变量的最大记值是有限制的(uint32_t 最大计数到 2^32-1)，而时间是无限的，这样会存在时间回绕的问题。

这里我们参考 Linux 内核里的 jiffies 回绕解决方案，使用了 time_after() 这个宏来判断是否超时。这个宏我们需要添加到 main.h 头文件中。

修改 main.h 头文件，添加几个超时判断宏：

```
/* USER CODE BEGIN Private defines */

/*
 *  These inlines deal with timer wrapping correctly. You are strongly encouraged to use them
 *  1. Because people otherwise forget
 *  2. Because if the timer wrap changes in future you won't have to alter your driver code.
 *
 *   time_after(a,b) returns true if the time a is after time b.
 *
 * Do this with "<0" and ">=0" to only test the sign of the result. A good compiler would generate
 * better code (and a really good compiler wouldn't care). Gcc is currently neither.
 */
#define time_after(a,b)      ( (int32_t)(b) - (int32_t)(a) < 0 )
#define time_before(a,b)     time_after(b,a)

#define time_after_eq(a,b)   ( (int32_t)(a) - (int32_t)(b) >= 0 )
#define time_before_eq(a,b)  time_after_eq(b,a)

/* USER CODE END Private defines */
```

修改好代码后，重新编译、烧录运行程序，遮住光强传感器并制造噪音，就会发现继电器控制的灯泡或 Led 灯亮了，15s 之后它们又将自动关闭。