# 第 2-3 讲：独立按键检测

## 1. 学习目的

1. 学习独立按键硬件电路原理。
2. 掌握如何读取 GPIO 状态。
3. 掌握编写独立按键检测程序。

## 2. 硬件电路设计

### 2.1. 轻触按键

轻触按键又称轻触开关，是电路中常用的一种开关元器件，也是一种常用的人机接口。广泛用于家电、数码产品、便携仪产品、电脑产品等电子设备中。

轻触按键，顾名思义我们只需要施加很小的力量即可改变开关连接的状态。轻触按键在所需外力作用下（按下按键）触点导通，无外力作用时（释放按键）触点断开，如下图所示：

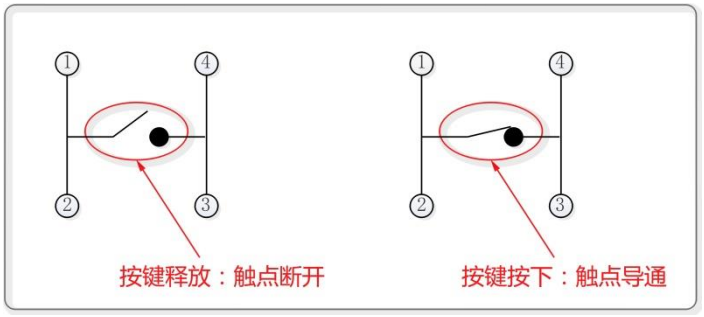


图 1：轻触开关原理

由此，我们可以通过将连接到轻触按键的 GPIO 配置为输入模式，之后读取该 GPIO 的状态来判断轻触按键是否按下。

PK107D 开发板上设计了 16 个轻触按键，通过 J6 端子选择这 16 个轻触按键作用：当短接 J6 端子的 1 和 2 针脚时，选择按键 S4、S5、S6 和 S7 为独立按键，此时按键 S8~S19 空闲。当短接 J6 端子的 2 和 3 针脚时，没有按键是独立按键，此时按键 S4~S19 均为矩阵按键使用。当选择 4 个轻触按键 S4、S5、S6 和 S7 为独立按键时，他们分别连接到单片机的 GPIO P3.3、P3.2、P3.1 和 P3.0。4 个独立轻触按键电路如下图所示，当独立按键 S4、S5、S6 和 S7 按下时，按键导通，按键连接的 GPIO 短接到 GND，这时，该 GPIO 的输入为低电平。

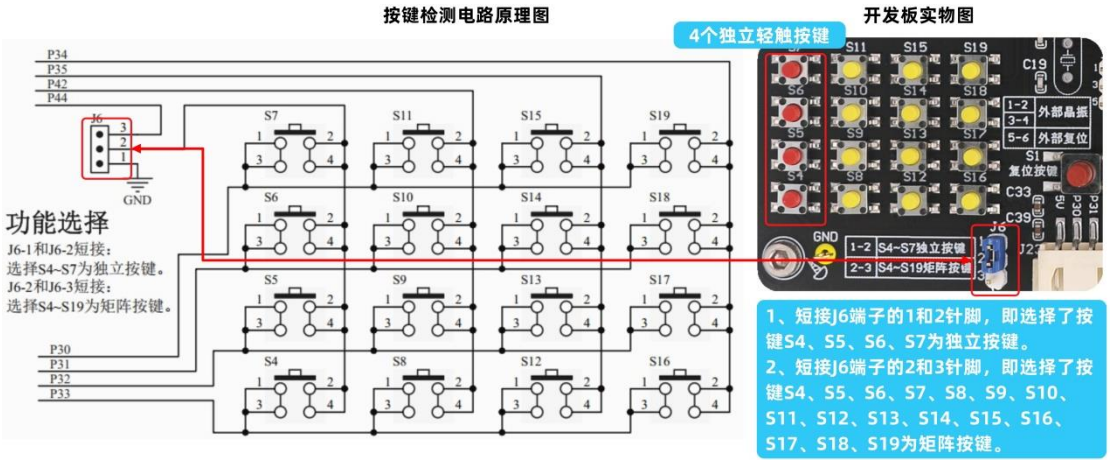


图 2：轻触按键电路

4 个独立轻触按键占用的单片机的引脚如下表：

表 1：独立轻触按键检测引脚分配

| 名称 | 引脚 | 说明 |
|---|---|---|
| S4 | P3.3 | 非独立 GPIO。 |
| S5 | P3.2 | 非独立 GPIO。 |
| S6 | P3.1 | 非独立 GPIO。 |
| S7 | P3.0 | 非独立 GPIO。 |

✧ 注：独立 GPIO 表示开发板没有其他的电路使用这个 GPIO，非独立 GPIO 说明开发板有其他电路用到了该 GPIO。读者在使用非独立 GPIO 使用时需要注意电路的连接，以避免多个电路使用了同一个 GPIO。

# 3. 软件设计

## 3.1. 独立按键检测实验

✧ 注：本节的实验是在“实验 2-2-1：有源蜂鸣器鸣响控制”的基础上修改，本节对应的实验源码是：“实验 2-3-1：独立按键检测”。

### 3.1.1. 实验内容

1. 配置独立按键 S4、S5、S6 和 S7 的 GPIO P3.3、P3.2、P3.1 和 P3.0 为准双向口。
2. 主循环中检测按键状态并使用软件消抖，当检测到按键按下后，翻转对应的 LED 指示灯的状态，即检测到按键 S4~ S7 按下后，分别翻转指示灯 D1~D4 的状态。

### 3.1.2. 代码编写

1. 新建一个名称为“button.c”的文件及其头文件“button.h”并保存到工程的“Source”文件夹，并将“button.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“button.c”文件中的函数，所以需要引用下面的头文件“button.h”。

**代码清单：**引用头文件

```
1.  //引用头文件
2.  #include    " button.h"
```

3. 引脚定义和配置

4 个独立按键 S4、S5、S6 和 S7 的 GPIO P3.3、P3.2、P3.1 和 P3.0，这里，我们使用“#define”定义如下宏。

**代码清单：**定义连接按键的引脚

```
1. #define BUTTON1_P33   P33         //用户按键 S4 用 GPIO P3.3
2. #define BUTTON2_P32   P32         //用户按键 S5 用 GPIO P3.2
3. #define BUTTON3_P31   P31         //用户按键 S6 用 GPIO P3.1
4. #define BUTTON4_P30   P30         //用户按键 S7 用 GPIO P3.0
```

4. 按键扫描函数

在编写按键检测函数之前，为了方便判断按键的状态，我们定义了一些常量用来表示按键的状态和有效按键（按键按下）的标号，代码如下。

**代码清单：**按键检测相关常量定义

```
1.  //定义按键按下和释放状态
2.  #define BUTTON_PRESSED      0
3.  #define BUTTON_RELEASED     1
4.
5.  //定义按键有效状态编号（按键按下）
6.  #define BUTTONS_RELEASED      0           //没有按键按下
7.  #define BUTTON1_PRESSED       1           //按键 S4 按下
8.  #define BUTTON2_PRESSED       2           //按键 S5 按下
9.  #define BUTTON3_PRESSED       3           //按键 S6 按下
10. #define BUTTON4_PRESSED       4           //按键 S7 按下
```

按键扫描函数中，对连接 4 个按键的 GPIO 的状态进行读取，如果为低电平，则延时 10ms 后再次读取（软件消抖），如仍为低电平，则认为按键有效，并返回对应的按键编号。

**代码清单：**按键扫描函数

```
1. /*********************************************************************
2. 功能描述：读取开发板上的 4 个独立按键(S4、S5、S6 和 S7)的状态
3. 参     数：mode [in]: 是否支持连按,=true:支持，=false：不支持
4. 返 回 值：有按键按下，返回对应的按键编号，否则返回 BUTTONS_RELEASED(没有按键按下)
5.  *********************************************************************/
6. u8 buttons_scan(u8 mode)
7. {
8.      static u8 btn_up=1;     //标志变量
9.
10.     if(mode==1)             //支持连按
```

```
11.     {
12.          btn_up=1;          //变量 Key_up 会被重新赋值为 1
13.     }
14.
15.     //读取按键 S4、S5、S6 和 S7 连接的 IO 口电平是否为低电平
16.     if(btn_up&&((BUTTON1_P33 == BUTTON_PRESSED ) || (BUTTON2_P32 == BUTTON_PRESSED ) || (BUTTON3
   _P31 == BUTTON_PRESSED ) || (BUTTON4_P30 == BUTTON_PRESSED )))
17.     {
18.         delay_ms(10);                          //软件延时 10ms，消抖
19.         btn_up=0;                              //变量 btn_up 清零
20.         if(BUTTON1_P33 == BUTTON_PRESSED)      //S4 按键按下(P3.3 为低电平)，返回按键有效
   (BUTTON1_PRESSED)
21.         {
22.            return BUTTON1_PRESSED;
23.         }
24.         else if(BUTTON2_P32 == BUTTON_PRESSED)//S5 按键按下(P3.2 为低电平)，返回按键有效
   (BUTTON2_PRESSED)
25.         {
26.            return BUTTON2_PRESSED;
27.          }
28.         else if(BUTTON3_P31 == BUTTON_PRESSED)//S6 按键按下(P3.1 为低电平)，返回按键有效
   (BUTTON3_PRESSED)
29.         {
30.            return BUTTON3_PRESSED;
31.         }
32.         else if(BUTTON4_P30 == BUTTON_PRESSED)//S7 按键按下(P3.0 为低电平)，返回按键有效
   (BUTTON4_PRESSED)
33.         {
34.            return BUTTON4_PRESSED;
35.          }
36.    }
37.    else if((BUTTON1_P33 == BUTTON_RELEASED )&&(BUTTON2_P32 == BUTTON_RELEASED ) &&(BUTTON3_P31 =
   = BUTTON_RELEASED )&&(BUTTON4_P30 == BUTTON_RELEASED ))
38.    {
39.        btn_up=1;              //变量 key_up 置位
40.    }
41.    return BUTTONS_RELEASED; //返回无按键按下
42.}
```

主函数中在主循环里面调用按键扫描函数 buttons_scan()查询是否有按键按下，如果有按键按下则翻转对应编号的指示灯的状态。

**代码清单：**主函数

```
/*******************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *******************************************************************************/
int main(void)
{
    u8 temp;

    P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
    P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
    P3M1 &= 0xF0;   P3M0 &= 0xF0;     //设置 P3.0 ~ P0.3 为准双向口

    SEG_off();          //控制 8 位数码管/点阵不显示
    ULN2003_off();      //控制步进电机、蜂鸣器、继电器等不工作
    leds_off();         //熄灭 D1~D8 指示灯
    delay_ms(10);       //延时

    while(1)
    {
        temp = buttons_scan(0);             //获取开发板用户按键检测值，不支持连按
        if(temp == BUTTON1_PRESSED)         //按键 S4 按下
        {
            led_toggle(LED_1);                //用户指示灯 D1 状态翻转
        }
        else if(temp == BUTTON2_PRESSED)    //按键 S5 按下
        {
            led_toggle(LED_2);                //用户指示灯 D2 状态翻转
        }
        else if(temp == BUTTON3_PRESSED)    //按键 S6 按下
        {
            led_toggle(LED_3);                //用户指示灯 D3 状态翻转
        }
        else if(temp == BUTTON4_PRESSED)    //按键 S7 按下
        {
            led_toggle(LED_4);                //用户指示灯 D3 状态翻转
        }

    }
}
```

### 3.1.3. 硬件连接

本实验程序的编写都是基于 IO 模式，所以 J24 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择为 IO 模式。同时 J6 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择独立按键。

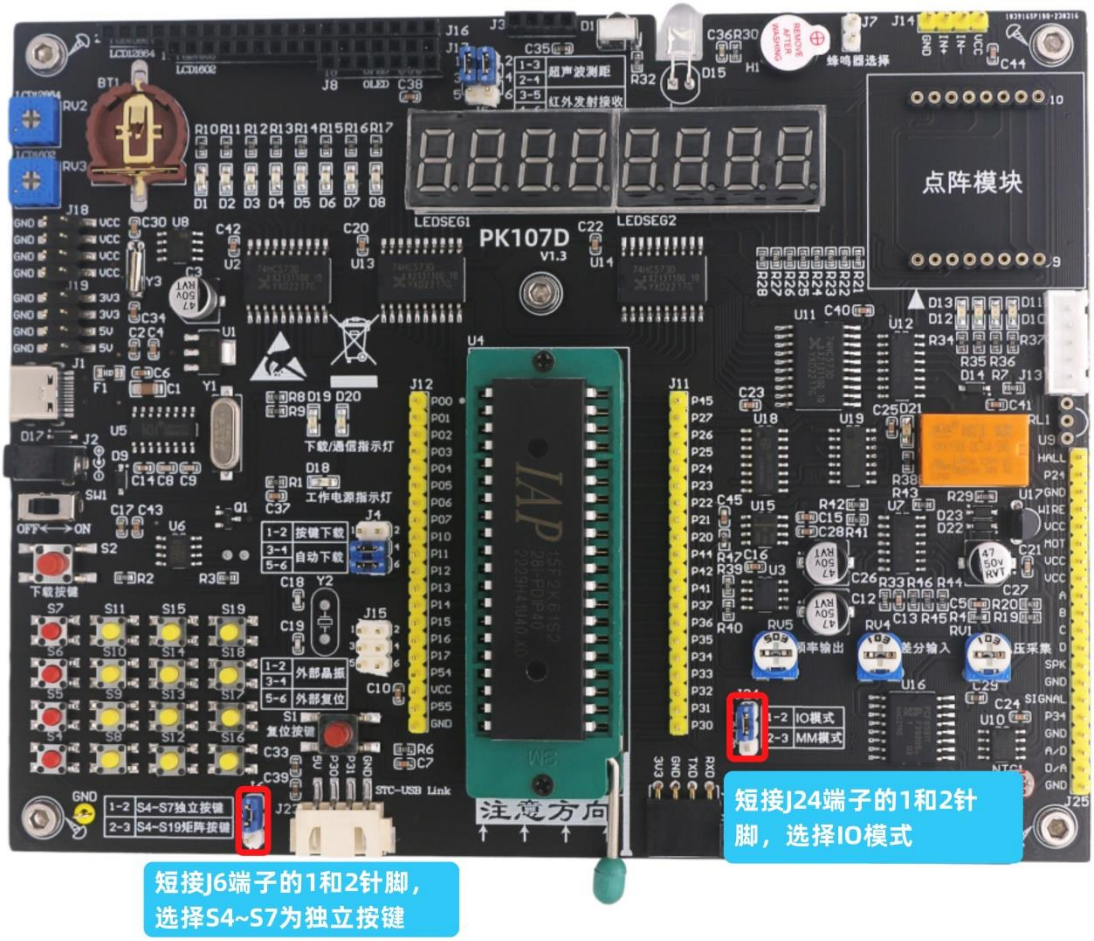


图 3：跳线帽短接

### 3.1.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-3-1：独立按键检测”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2. 双击“…\button\Project”目录下的工程文件“button.uvproj”。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“button.hex”位于工程的“…\button\project\Objects”目录下。
4. 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：12MHz。
5. 程序运行后，依次按下独立按键 S4、S5、S6 和 S7，可以观察到每按一次按键，对应编号的用户指示灯的状态翻转。