第 2-3 讲：按键检测

1. 学习目的

1. 学习轻触按键和触摸按键硬件电路原理。
2. 学习 STC8A8K64D4 用作输入时相关寄存器的配置。
3. 掌握如何读取 GPIO 状态。
4. 掌握编写轻触按键和触摸按键检测程序。

2. 硬件电路设计

IK-64D4 开发板上设计了 4 个轻触按键和一个触摸按键，提供给用户作为按键开关使用。

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：

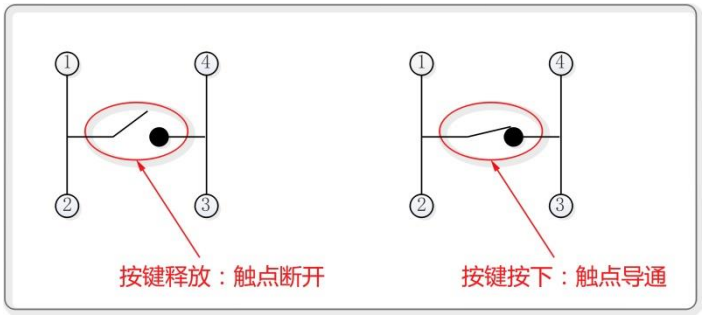

图 1：轻触开关原理

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。

IK-64D4 开发板上设计了 4 个轻触按键 Key1、Key2、Key3 和 Key4，分别连接到 STC8A8K64D4 的 GPIO P3.7、P3.6、P0.7 和 P0.5。轻触按键电路如下图所示，当按键处于释放状态时（按键没有按下），按键断开，由于上拉电阻的作用 GPIO 的输入为高电平。当按键按下时，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。由此，程序中我们可以根据读取的 GPIO 的状态是低电平还是高电平来判断对应按键是按下还是释放。

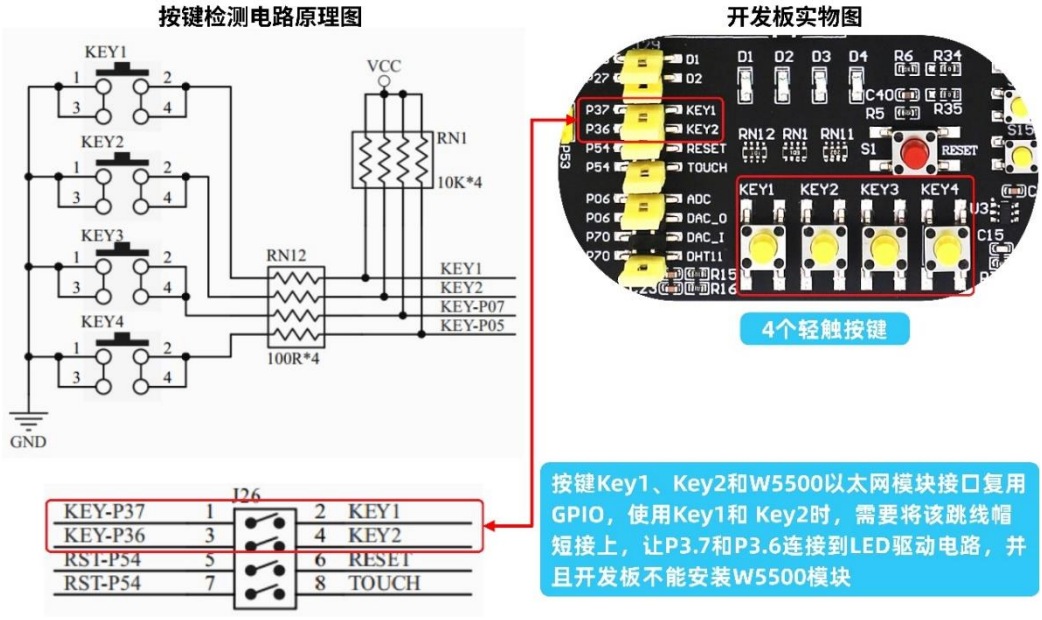

图 2：轻触按键电路

4 个轻触按键占用的单片机的引脚如下表：

表 1：轻触按键检测引脚分配

名称	引脚	说明
Key1	P3.7	非独立 GPIO，和以太网模块 W5500 接口复用。
Key2	P3.6	非独立 GPIO，和以太网模块 W5500 接口复用。
Key3	P0.7	独立 GPIO。
Key4	P0.5	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。

2.2. 触摸按键

IK-64D4 开发板上设计了一路基于 TTP223 触摸检测芯片的触摸按键，TTP223 是电容式单键触摸按键 IC，电压输入范围为 2.0V~5.5V。TTP223 利用操作者的手指与触摸按键焊盘之间产生电荷电平来进行检测，通过监测电荷的微小变化来确定手指接近或者触摸到感应表面。没有任何机械部件，不会磨损，其感测部分可以放置到任何绝缘层（通常为玻璃或塑料材料）的后面，很容易制成与周围环境相密封的键盘。

触摸按键电路如下图所示。

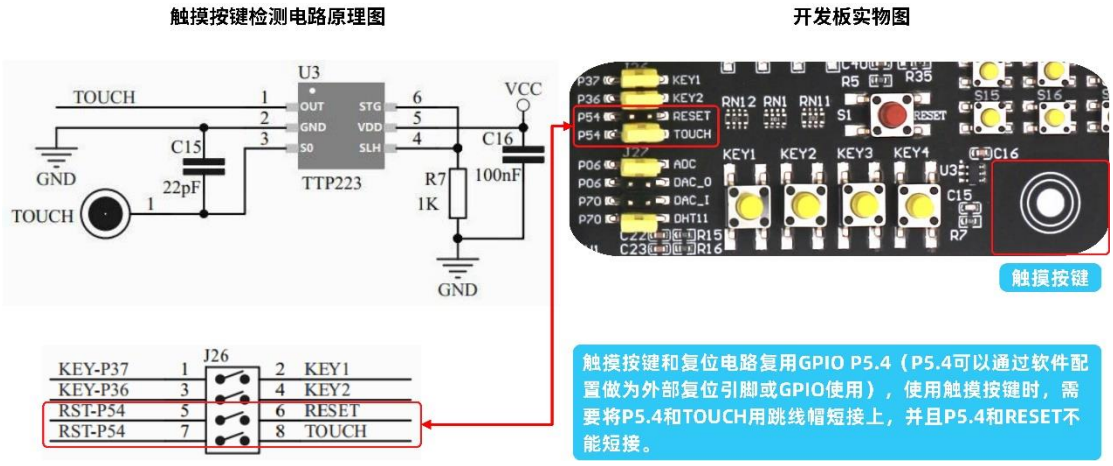

图 3：触摸按键电路

TTP223 的检测灵敏度可通过外部电容值(上图中的 C15)来调整。SLH 引脚用于设置 TTP223 的输出方式。

1) SLH = 0：触摸时，TTP223 的 OUT 引脚输出高电平。

2) SLH = 1：触摸时，TTP223 的 OUT 引脚输出地电平。

本电路中，SLH = 0，所以触摸时 OUT 引脚输出高电平，无触摸时 OUT 引脚输出低电平。注意一下，这和轻触按键的电路输出刚好是反的，轻触按键电路是按键按下时，电路输出低电平，无按键按下时电路输出高电平。

✧ 为什么触摸按键输出的信号不做成和轻触按键一样，也是按键时输出低电平，无按键时输出高电平？

这是因为：作为开发板，要方便用户测试，2 种不同类型的输出方式更方便我们使用，另外在后续的章节中我们还会用到按键信号的上升沿和下降沿，这时就可以通过轻触键和触摸按键来获取，而不需要另外接线。

触摸按键占用的 STC8A8K64D4 的引脚如下表：

表 2：触摸按键引脚分配

名称	引脚	说明
触摸按键	P5.4	和外部复位电路共用 IO

2.3. 按键检测电路需要考虑的因素

按键检测电路设计的时候，需要我们考虑两个方面：按键释放时 GPIO 口状态的确定和按键检测电路的保护以及按键消抖。

1. 按键释放时 GPIO 口状态的确定

按键检测电路中，当按键释放后要能保证 GPIO 口电平是确定的，即按键释放时 GPIO 口固定为高电平或低电平。所以一般按键电路中，都会通过上拉电阻或下拉电阻来保证当按键释放时 GPIO 处于固定的电平。如果单片机的 GPIO 有片内可配置的上/下拉电阻，也

可以使用片内上/下拉电阻而不用在单片机外部增加上/下拉电阻，使用 GPIO 检测按键时只需通过软件打开上拉/下拉电阻就可以了。

2. 按键检测电路保护

设计电路时，可以考虑在按键和 STC8A8K64D4 单片机的 GPIO 之间串接了一个小阻值的电阻，这个电阻的作用如下：

- 保护 IO，若 IO 口不小心被配置成了推挽输出，按下按键可能会损坏 IO，串接电阻后，即使出现这种情况，也不会损坏 IO。
- 降低按键动作时产生的抖动峰值电压，防止抖动电压对单片机产生影响。虽然很多单片机在 IO 上都有电压钳位设计，可以承受一定范围内的电压抖动，但是通过串接一个小阻值电阻既能让抖动峰值电压更低而又不影响按键检测电路的性能，而且花费的代价也很小（仅串接一个电阻），无疑是比较合算的。

3. 按键硬件消抖

对于按键硬件上的消抖，一般常用的方式是在按键上并接一个容值约 0.1uF 左右电容，利用电容两端的电压不能突变的特性，消除抖动时产生的毛刺电压。虽然电容可以起到消除抖动的作用，但是在考虑按键灵敏度的情况下，电容是无法完全消除抖动的，消除抖动还需要软件的配合。

本电路中，没有采用电容消抖的方式，在整体设计上，我们是使用软件完成消抖的。

3. 软件设计

3.1. GPIO 配置

STC8A8K64D4 提供的 7 个用于操作 GPIO 的寄存器中，和输入相关的寄存器有端口模式配置寄存器、端口上拉电阻控制寄存器、端口施密特触发控制寄存器、端口数字信号输入使能控制寄存器和端口数据寄存器。使用 GPIO 进入输入检测的时候，首先需要配置 GPIO 输入相关的寄存器，包含下面 4 个配置：

- 配置 GPIO 的工作模式为：本例中配置为准双向口。
- 配置上拉电阻：是否开启 GPIO 的上拉电阻要根据实际情况确定，如硬件电路上已经设计了外部上拉电阻，则无需开启。本例中需要开启外部上拉电阻（硬件电路上没有外部上拉）。
- 端口施密特触发控制寄存器：需要使能，端口施密特触上电复位后默认使能，因此，通常代码里面没有配置这个。
- 端口数字信号输入使能控制寄存器：需要使能，端口施密特触上电复位后默认使能。

配置完成后，即可通过读取端口数据寄存器来获取 GPIO 当前的状态是高电平还是低电平。

这几个配置项中，GPIO 工作模式配置在前文已经描述过，这里不再赘述，端口施密特触发控制寄存器和端口数字信号输入使能控制寄存器通常无需配置，使用默认值即可，这里，我们主要看一下如下使能 GPIO 的上拉电阻。

配置 GPIO 上拉电阻的寄存器如下表所示，STC8A8K64D4 共有 8 个端口 P0~P7，因此，

下表中只有 P0PU~P7PU 寄存器是 STC8A8K64D4 拥有的。

表 3：端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻

端口上拉电阻控制寄存器中的各个位的值对应端口中的 GPIO 的上拉电阻的开启或关闭（值为 1：开启上拉电阻，值为 0：关闭上拉电阻），如 P0PU 中的位 0 的值对应 P0.0 的上拉电阻的开启或关闭。端口上拉电阻控制寄存器是不支持位寻址的，也就是只能使用寄存器 P0PU~P7PU 访问，另外，需要特别注意的是端口上拉电阻控制寄存器为扩展 RAM 区特殊功能寄存器，访问前需先将 P_SW2 寄存器的最高位（EAXFR）置 1，否则无法访问。

■ 开启上拉电阻示例：使能 P3.6 的上拉电阻

代码清单：使能 P3.6 的上拉电阻

```
1.  P_SW2 |= 0x80;          //将 EAXFR 位置 1，使能访问 XFR（扩展 RAM 区特殊功能寄存器）
2.  P3PU |= 0x40;           //将 P3PU 的位 6 设置为 1，使能 P3.6 的上拉电阻
3.  P_SW2 &= 0x7F;          //将 EAXFR 位置 0，关闭访问 XFR
```

3.2. 轻触按键检测实验

✧ 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：轻触按键检测”。

3.2.1. 实验内容

1. 配置连接按键 Key1、Key2、Key3 和 Key4 的 GPIO P3.7、P3.6、P0.7 和 P0.5 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 Key1~ Key4 按下后，分别翻转指示灯 D1~D4 的状态。

3.2.2. 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 个轻触按键 Key1、Key2、Key3 和 Key4 的 GPIO P3.7、P3.6、P0.7 和 P0.5，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_P37   P37         //用户按键 Key1 用 GPIO 口 P3.7
2.  #define BUTTON2_P36   P36         //用户按键 Key2 用 GPIO 口 P3.6
3.  #define BUTTON3_P07   P07         //用户按键 Key3 用 GPIO 口 P0.7
4.  #define BUTTON3_P05   P05         //用户按键 Key4 用 GPIO 口 P0.5
```

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 KEY1 按下
8.  #define BUTTON2_PRESSED       2           //按键 KEY2 按下
9.  #define BUTTON3_PRESSED       3           //按键 KEY3 按下
10. #define BUTTON4_PRESSED       4           //按键 KEY4 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 个用户按键(KEY1、KEY2、KEY3 和 KEY4)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.  *********************************************************************/
1.  u8 buttons_scan(u8 mode)
2.  {
3.      static u8 btn_up=1;    //标志变量
```

```
4.
5.      if(mode==1)             //支持连按
6.      {
7.          btn_up=1;           //变量 Key_up 会被重新赋值为 1
8.      }
9.
10.     //读取按键 KEY1、KEY2、KEY3 和 KEY4 连接的 IO 口电平是否为低电平
11.     if(btn_up&&((BUTTON1_P37 == BUTTON_PRESSED ) || (BUTTON2_P36 == BUTTON_PRESSED ) ||
12.         (BUTTON3_P07 == BUTTON_PRESSED ) || (BUTTON4_P05 == BUTTON_PRESSED )))
13.     {
14.         delay_ms(10);                           //软件延时 10ms，消抖
15.         btn_up=0;                               //变量 btn_up 清零
16.         //KEY1 按键按下(P3.7 为低电平)，返回按键有效(BUTTON3_PRESSED)
17.         if(BUTTON1_P37 == BUTTON_PRESSED)
18.         {
19.           return BUTTON1_PRESSED;
20.         }
21.         //KEY2 按键按下(P3.6 为低电平)，返回按键有效(BUTTON4_PRESSED)
22.         else if(BUTTON2_P36 == BUTTON_PRESSED)
23.         {
24.           return BUTTON2_PRESSED;
25.         }
26.         //KEY3 按键按下(P0.7 为低电平)，返回按键有效(BUTTON5_PRESSED)
27.         else if(BUTTON3_P07 == BUTTON_PRESSED)
28.         {
29.           return BUTTON3_PRESSED;
30.         }
31.         //KEY4 按键按下(P0.5 为低电平)，返回按键有效(BUTTON5_PRESSED)
32.         else if(BUTTON4_P05 == BUTTON_PRESSED)
33.         {
34.           return BUTTON4_PRESSED;
35.         }
36.     }
37.     else if((BUTTON1_P37 == BUTTON_RELEASED )&&(BUTTON2_P36 == BUTTON_RELEASED ) &&
38.             (BUTTON3_P07 == BUTTON_RELEASED )&&(BUTTON4_P05 == BUTTON_RELEASED ))
39.     {
40.       btn_up=1;               //变量 key_up 置位
41.     }
42.     return BUTTONS_RELEASED; //返回无按键按下
43. }
```

主函数中配置连接按键的 GPIO 为准双向口，因为开发板轻触按键检测硬件电路上设计

有外部上拉电阻，因此，无需要打开 GPIO 的片内上拉电阻。之后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 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口（指示灯 D1 和 D2）
    P7M1 &= 0xF9;   P7M0 &= 0xF9;     //设置 P7.1~P7.2 为准双向口（指示灯 D4 和 D3）
    P3M1 &= 0x3F;   P3M0 &= 0x3F;     //设置 P3.6~P3.7 为准双向口（按键 KEY2 和 KEY1）
    P0M1 &= 0x5F;   P0M0 &= 0x5F;     //设置 P0.5，P0.7 为准双向口（按键 KEY4 和 KEY3）

//如果按键电路上没有外部上拉电阻，需要开启 GPIO 的片内上拉。
//开发板的按键电路设计了上拉电阻，因此，无需开启片内上拉
//  P_SW2 |= 0x80;                  //将 EAXFR 位置 1，以访问在 XDATA 区域的扩展 SFR
//  P0PU |= 0xA0;                   //开启 P0.5、P0.7 的上拉电阻
//  P3PU |= 0xC0;                   //开启 P3.6、P3.7 的上拉电阻
//  P_SW2 &= 0x7F;                  //将 EAXFR 位置 0，恢复访问 XRAM

  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 KEY1 按下
    {
      led_toggle(LED_1);                //用户指示灯 D1 状态翻转
    }
    else if(temp == BUTTON2_PRESSED)    //按键 KEY2 按下
    {
      led_toggle(LED_2);                //用户指示灯 D2 状态翻转
    }
    else if(temp == BUTTON3_PRESSED)    //按键 KEY3 按下
    {
      led_toggle(LED_3);                //用户指示灯 D3 状态翻转
    }
    else if(temp == BUTTON4_PRESSED)    //按键 KEY4 按下
    {
```

```
39.         led_toggle(LED_4);              //用户指示灯 D3 状态翻转
40.       }
41.     }
42. }
```

3.2.3. 硬件连接

本实验需要使用 LED 指示灯和按键，因此需要用跳线帽短接复用引脚的指示灯（D1 和 D2）和按键（KEY1 和 KEY2），而指示灯 D3 和 D4 以及按键 KEY3 和 KEY4 是独立引脚，没有和其他电路复用引脚，是没有短接跳线帽的操作的。

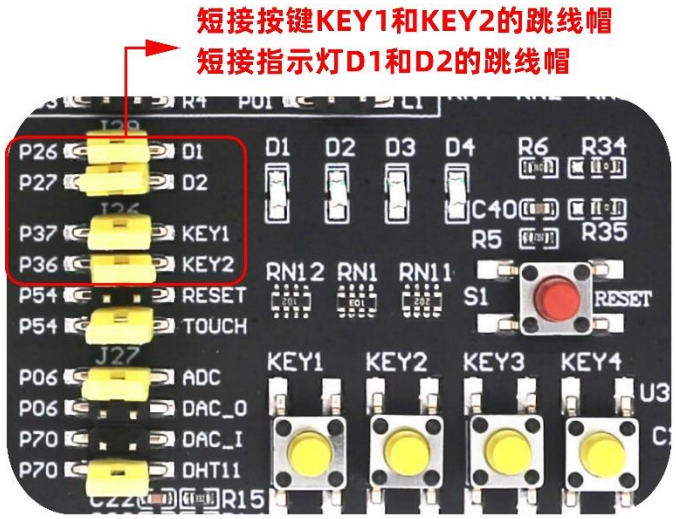

图 4：跳线帽短接

3.2.4. 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：轻触按键检测”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（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：24MHz。
5. 程序运行后，依次按下用户按键 KEY1、KEY2、KEY3 和 KEY4，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。

3.3. 触摸按键检测实验

✧ 注：本节的实验是在“实验 2-3-1：轻触按键实验”的基础上修改，本节对应的实验源码是：“实验 2-3-2：触摸按键检测”。

3.3.1. 实验内容

在“实验 2-3-1：轻触按键实验”的基础上增加触摸按键的检测代码，主要包含下面两个方面。

1. 配置连接触摸按键的 GPIO P5.4 为准双向口。
2. 主循环中检测触摸按键状态并使用软件消抖，当检测到触摸按键按下后，点亮 4 个 LED 指示灯。

3.3.2. 代码编写

1. 引脚定义和配置

在“button.h”文件中按键引脚定义的地方加入触摸按键的引脚定义，代码清单如下。

代码清单：定义连接按键的引脚

```
1.  //省略无关的代码
2.  #define TOUCH_BUTTON_P54    P54     //触摸按键用 GPIO 口 P5.4
```

2. 触摸按键扫描函数

触摸按键的检测方式和轻触按键一样，不同的是轻触按键检测到低电平时认为按键按下，触摸按键检测到高电平时认为按键按下，代码如下。

代码清单：按键检测相关常量定义

```
1.  /*************************************************************************
2.  功能描述：读取开发板上的触摸按键的状态
3.  参     数：无
4.  返 回 值：有按键按下，返回 BUTTON_PRESSED，否则返回 BUTTONS_RELEASED(没有按键按下)
5.   *************************************************************************/
6.  u8 touch_button_scan(void)
7.  {
8.     //读取触摸按键用引脚 P5.4 是否是高电平 (用手指触摸按键感应区域，引脚为高电平)
9.     if(TOUCH_BUTTON_P54 == 1)
10.    {
11.       delay_ms(10);                    //软件延时 10ms，软件消抖
12.       if(TOUCH_BUTTON_P54== 1)         //检测触摸按键用引脚 P5.4 是否依然是高电平
13.       {
14.          return BUTTON_PRESSED;        //返回按键按下
15.       }
16.     }
17.     return BUTTON_RELEASED;    //返回无按键按下
18. }
```

主函数中配置连接触摸按键的 GPIO P5.4 为准双向口，之后在主循环里面调用触摸按键扫描函数 touch_button_scan()查询是否有按键按下，如果有按键按下则点亮 4 个 LED 指示灯。

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;
    //省略了无关的代码
    P5M1 &= 0xEF;   P5M0 &= 0xEF;     //设置 P5.4 为准双向口（触摸按键）

    while(1)
    {
        //省略了无关的代码
        temp = touch_button_scan();         //读取开发板触摸按键检测值
        if(temp == BUTTON_PRESSED)          //如果触摸按键按下
        {
            leds_on();                      //点亮 4 个用户指示灯
        }
    }
}
```

3.3.3. 硬件连接

本实验需要使用 LED 指示灯、4 个轻触按键和触摸按键，因此需要用跳线帽短接复用引脚的指示灯（D1 和 D2）、按键（KEY1 和 KEY2）和触摸按键（TOUCH），而指示灯 D3 和 D4 以及按键 KEY3 和 KEY4 是独立引脚，没有和其他电路复用引脚，是没有短接跳线帽的操作的。

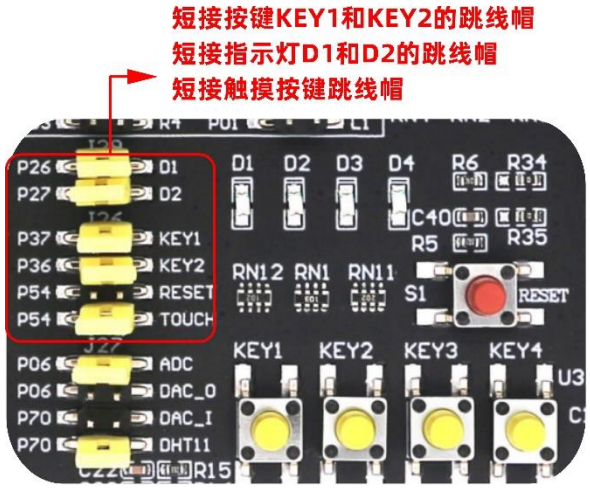

图 5：跳线帽短接

3.3.4. 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 2-3-2：触摸按键检测”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（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“…\touch_button\Project”目录下的工程文件“touch_button.uvproj”。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“touch_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：24MHz。
5. 程序运行后，按下用户按键 KEY1~KEY4 可以翻转对应编号的指示灯状态，用手指去接触触摸按键，4 个 LED 指示灯 D1~D4 全部点亮。