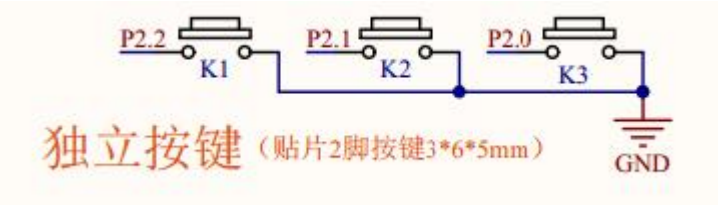
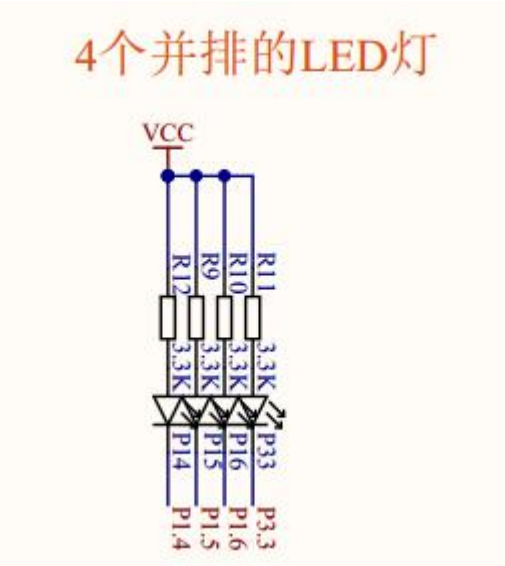
第一百零七节： 开关感应器的识别与软件滤波。

【107.1　开关感应器的识别与软件滤波。】



上图 107.1.1　独立按键模拟开关感应器



上图 107.1.2　LED 电路

什么叫开关感应器？凡是只能输出 0 和 1 这两种状态的感应器都可以统称为开关感应器。前面花了大量的章节讲按键，按键的识别主要是识别电平变化状态的“下降沿”，程序代码中有 1 个特别的变量标志叫“自锁标志”，还有 1 个用来消除抖动的“计时器”。本节讲的开关感应器跟按键很相似，差别在于，开关感应器是识别电平变化状态的“电平”，程序代码中没有“自锁标志”，但是多增加了 1 个用来消除抖动的“计时器”，也就是一共有两个用来消除抖动的“计时器”，这两个“计时器”相互“清零”相互“抗衡”，从而实现了开关感应器的“消抖”处理，专业术语也叫“软件滤波”。消抖的时间跟按键差不多，我的经验值是 20ms 到 30ms 之间，我平时在项目中喜欢用 20ms。

在显示框架方面，除了之前讲过 Gu8DisplayUpdate 这类“显示刷新变量”，本节介绍另外一种常用的显示框架，原理是“某数值跟上一次对比，如果发生了变化（两数值不一样），则自动刷新显示,并及时记录当前值”。

本节例程实现的功能如下：用 K1 独立按键模拟开关感应器，K1 独立按键“没有被按下”时是高电平，单片机识别到这种“高电平”，就让 P1.4 所在的 LED 灯发亮；K1 独立按键“被按下”时是低电平，单片机识别到这种“低电平”，就让 P1.4 所在的 LED 灯熄灭。

```
#include "REG52.H"
```

```
#define SENSOR_TIME  20    //开关感应器的“滤波”时间

void T0_time();
void SystemInitial(void) ;
void Delay(unsigned long u32DelayTime) ;
void PeripheralInitial(void) ;

void VoiceScan(void);
void SensorScan(void);
void DisplayTask(void);   //显示的任务函数（LED 显示状态）

sbit P1_4=P1^4;
sbit Sensor_K1_sr=P2^2;   //开关感应器 K1 所在的引脚

volatile unsigned char vGu8Sensor_K1=0;  //K1 开关感应器的当前电平状态。

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
        DisplayTask();   //显示的任务函数（LED 显示状态）
    }
}

/* 注释一：
* 后缀为_Last 这类“对比上一次数值发生变化而自动刷新显示”在“显示框架”里是很常见的，
* 目的是，既能及时刷新显示，又能避免主函数“不断去执行显示代码”而影响程序效率。
*/

void DisplayTask(void)   //显示的任务函数（LED 显示状态）
{
   // Su8Sensor_K1_Last 初始化取值 255，只要不为 0 或者 1 就行，目的是让上电就发生第一次刷新。
    static unsigned char Su8Sensor_K1_Last=255;  //记录 K1 开关感应器上一次的电平状态。

    if(Su8Sensor_K1_Last!=vGu8Sensor_K1)  //如果当前值与上一次值不一样，就自动刷新
    {
        Su8Sensor_K1_Last=vGu8Sensor_K1;  //及时记录最新值，避免主函数“不断去执行显示代码”

        if(0==vGu8Sensor_K1) //如果当前电平状态为“低电平”，LED 熄灭
        {
              P1_4=1;  //LED 熄灭
```

```
        }
        else  //如果当前电平状态为“高电平”，LED 发亮
        {
              P1_4=0;  //LED 发亮
        }
    }
}

/* 注释二：
*  本节破题的关键：
*  两个“计时器”相互“清零”相互“抗衡”，从而实现了开关感应器的“消抖”处理，
*  专业术语也叫“软件滤波”。这种滤波方式，不管是从“高转成低”，还是“低转成高”，
*  如果在某个瞬间出现干扰抖动，某个计数器都会及时被“清零”，从而起到非常高效的消抖滤波作用。
*/

void SensorScan(void)  //此函数放在定时中断里每 1ms 扫描一次，用来识别和滤波开关感应器
{
      static unsigned int Su16Sensor_K1_H_Cnt=0;  //判断高电平的计时器
      static unsigned int Su16Sensor_K1_L_Cnt=0;  //判断低电平的计时器

      if(0==Sensor_K1_sr)
      {
          Su16Sensor_K1_H_Cnt=0;  //在判断低电平的时候，高电平的计时器被清零，巧妙极了!
          Su16Sensor_K1_L_Cnt++;
          if(Su16Sensor_K1_L_Cnt>=SENSOR_TIME)
          {
              Su16Sensor_K1_L_Cnt=0;
              vGu8Sensor_K1=0;   //此全局变量反馈当前电平的状态
          }

      }
      else
      {
          Su16Sensor_K1_L_Cnt=0;   //在判断高电平的时候，低电平的计时器被清零，巧妙极了!
          Su16Sensor_K1_H_Cnt++;
          if(Su16Sensor_K1_H_Cnt>=SENSOR_TIME)
          {
              Su16Sensor_K1_H_Cnt=0;
              vGu8Sensor_K1=1;  //此全局变量反馈当前电平的状态
          }
      }
}

void T0_time() interrupt 1
```

```
{
    SensorScan();  //开关感应器的识别与软件滤波处理

    TH0=0xfc;
    TL0=0x66;
}


void SystemInitial(void)
{
    TMOD=0x01;
    TH0=0xfc;
    TL0=0x66;
    EA=1;
    ET0=1;
    TR0=1;
}

void Delay(unsigned long u32DelayTime)
{
    for(;u32DelayTime>0;u32DelayTime--);
}

void PeripheralInitial(void)
{

}
```