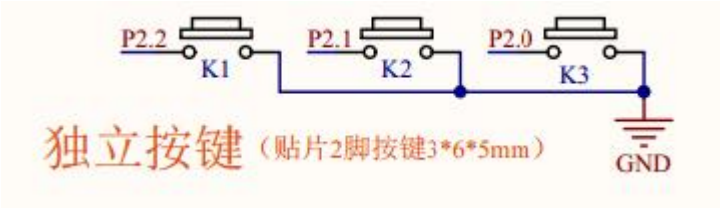
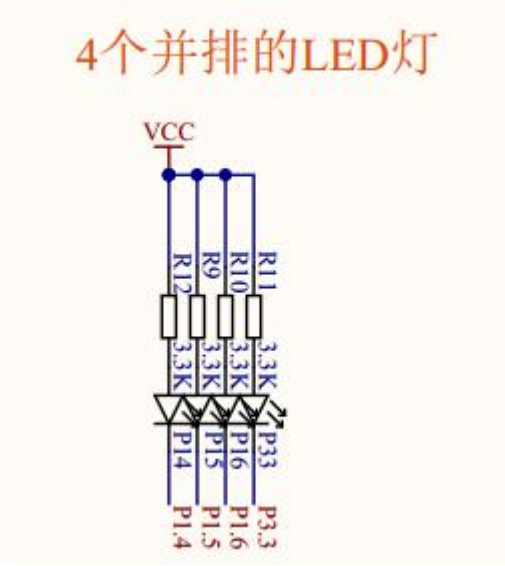
第九十三节： 独立按键鼠标式的单击与双击。

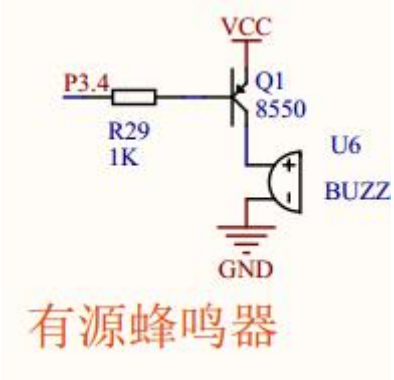
【93.1 鼠标式的单击与双击。】



上图 93.1.1 独立按键电路



上图 93.1.2 LED 电路



上图 93.1.3 有源蜂鸣器电路

鼠标的左键，可以触发单击，也可以触发双击。双击的规则是这样的，两次单击，如果第 1 次单击与第 2 次单击的时间比较“短”的时候，则这两次单击就构成双击。编写这个程序的最大亮点是如何控制好第 1

次单击与第 2 次单击的时间间隔。

程序例程要实现的功能是：（1）单击改变 LED 灯的显示状态，单击一次 LED 从原来“灭”的状态变成“亮”的状态，或者从原来“亮”的状态变成“灭”的状态，依次循环切换。（2）双击则蜂鸣器发出“嘀”的一声。代码如下：

```
#include "REG52.H"

#define KEY_VOICE_TIME   50     //按键触发后发出的声音长度
#define KEY_FILTER_TIME  25     //按键滤波的“稳定时间”25ms
#define KEY_INTERVAL_TIME  250  //连续两次单击之间的最大有效时间 250ms

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

void BeepOpen(void);
void BeepClose(void);
void LedOpen(void);
void LedClose(void);

void VoiceScan(void);
void KeyScan(void);    //按键识别的驱动函数，放在定时中断里
void SingleKeyTask(void);   //单击按键任务函数，放在主函数内
void DoubleKeyTask(void);   //双击按键任务函数，放在主函数内

sbit P3_4=P3^4;       //蜂鸣器
sbit P1_4=P1^4;       //LED

sbit KEY_INPUT1=P2^2;  //K1 按键识别的输入口。

volatile unsigned char vGu8BeepTimerFlag=0;
volatile unsigned int vGu16BeepTimerCnt=0;

unsigned char Gu8LedStatus=0; //记录 LED 灯的状态，0 代表灭，1 代表亮
volatile unsigned char vGu8SingleKeySec=0;  //单击按键的触发序号
volatile unsigned char vGu8DoubleKeySec=0;  //双击按键的触发序号

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
```

```
    {
       SingleKeyTask();    //单击按键任务函数
       DoubleKeyTask();    //双击按键任务函数
    }
}

void T0_time() interrupt 1
{
    VoiceScan();
    KeyScan();    //按键识别的驱动函数

    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)
{
/* 注释一：
* 把 LED 的初始化放在 PeripheralInitial 而不是放在 SystemInitial，是因为 LED 显示内容对上电
* 瞬间的要求不高。但是，如果是控制继电器，则应该把继电器的输出初始化放在 SystemInitial。
*/

    //根据 Gu8LedStatus 的值来初始化 LED 当前的显示状态，0 代表灭，1 代表亮
    if(0==Gu8LedStatus)
    {
        LedClose();  //关闭 LED
    }
    else
```

```
    {
        LedOpen();   //点亮 LED
    }
}

void BeepOpen(void)
{
    P3_4=0;
}

void BeepClose(void)
{
    P3_4=1;
}
void LedOpen(void)
{
    P1_4=0;
}

void LedClose(void)
{
    P1_4=1;
}

void VoiceScan(void)
{

      static unsigned char Su8Lock=0;

      if(1==vGu8BeepTimerFlag&&vGu16BeepTimerCnt>0)
      {
          if(0==Su8Lock)
          {
               Su8Lock=1;
               BeepOpen();
          }
          else
          {

               vGu16BeepTimerCnt--;

               if(0==vGu16BeepTimerCnt)
               {
                   Su8Lock=0;
```

```
                    BeepClose();
                }

            }
        }
}

/* 注释二：
* 双击按键扫描的详细过程：
* 第一步：平时没有按键被触发时，按键的自锁标志,去抖动延时计数器一直被清零。
*         如果之前已经有按键触发过 1 次单击，那么启动时间间隔计数器 Su16KeyIntervalCnt1，
*         在 KEY_INTERVAL_TIME 这个允许的时间差范围内，如果一直没有第 2 次单击触发，
*         则把累加按键触发的次数 Su8KeyTouchCnt1 也清零，上一次累计的单击数被清零，
*         就意味着下一次新的双击必须重新开始累加两次单击数。
* 第二步：一旦有按键被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值 KEY_FILTER_TIME 时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO 口突然瞬间触发成高电平，这个时候马上把延时计数器 Su16KeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰，以后凡是用到开关感应器的时候，
*         都可以用类似这样的方法去干扰。
* 第三步：如果按键按下的时间超过了阀值 KEY_FILTER_TIME，马上把自锁标志 Su8KeyLock1 置 1，
*         防止按住按键不松手后一直触发。与此同时，累加 1 次按键次数，如果按键次数累加有 2 次，
*         则认为触发双击按键，并把编号 vGu8DoubleKeySec 赋值。
* 第四步：等按键松开后，自锁标志 Su8KeyLock1 及时清零解锁，为下一次自锁做准备。并且累加间隔时间，
*         防止两次按键的间隔时间太长。如果连续 2 次单击的间隔时间太长达到了 KEY_INTERVAL_TIME
*         的长度，立即清零当前按键次数的计数器，这样意味着上一次的累加单击数无效，下一次双击
*         必须重新累加新的单击数。
*/

void KeyScan(void)  //此函数放在定时中断里每 1ms 扫描一次
{
   static unsigned char Su8KeyLock1;           //1 号按键的自锁
   static unsigned int  Su16KeyCnt1;           //1 号按键的计时器
   static unsigned char Su8KeyTouchCnt1;       //1 号按键的次数记录
   static unsigned int  Su16KeyIntervalCnt1;   //1 号按键的间隔时间计数器

   //1 号按键
   if(0!=KEY_INPUT1)//IO 是高电平，说明按键没有被按下，这时要及时清零一些标志位
   {
       Su8KeyLock1=0; //按键解锁
       Su16KeyCnt1=0; //按键去抖动延时计数器清零，此行非常巧妙。
       if(Su8KeyTouchCnt1>=1) //之前已经有按键触发过一次，再来一次就构成双击
       {
           Su16KeyIntervalCnt1++; //按键间隔的时间计数器累加
           if(Su16KeyIntervalCnt1>=KEY_INTERVAL_TIME) //达到最大允许的间隔时间
```

```
            {
                Su16KeyIntervalCnt1=0; //时间计数器清零
                Su8KeyTouchCnt1=0;     //清零按键的按下的次数
            }
        }
    }
    else if(0==Su8KeyLock1)//有按键按下，且是第一次被按下。此行如有疑问，请看第 92 节的讲解。
    {
       Su16KeyCnt1++; //累加定时中断次数
       if(Su16KeyCnt1>=KEY_FILTER_TIME) //滤波的“稳定时间”KEY_FILTER_TIME，长度是 25ms。
       {
          Su8KeyLock1=1;  //按键的自锁,避免一直触发
          Su16KeyIntervalCnt1=0;   //按键有效间隔的时间计数器清零
          Su8KeyTouchCnt1++;       //记录当前单击的次数
          if(1==Su8KeyTouchCnt1)   //只按了 1 次
          {
               vGu8SingleKeySec=1;     //单击任务
          }
          else if(Su8KeyTouchCnt1>=2)  //连续按了两次以上
          {
               Su8KeyTouchCnt1=0;    //统计按键次数清零
               vGu8SingleKeySec=1;   //单击任务
               vGu8DoubleKeySec=1;   //双击任务
          }
       }
    }

}

void SingleKeyTask(void)    //单击按键任务函数，放在主函数内
{
    if(0==vGu8SingleKeySec)
    {
        return;//按键的触发序号是 0 意味着无按键触发，直接退出当前函数，不执行此函数下面的代码
    }

    switch(vGu8SingleKeySec) //根据不同的按键触发序号执行对应的代码
    {
       case 1:     //单击任务
            //通过 Gu8LedStatus 的状态切换，来反复切换 LED 的“灭”与“亮”的状态
            if(0==Gu8LedStatus)
            {
                Gu8LedStatus=1; //标识并且更改当前 LED 灯的状态。0 就变成 1。
                LedOpen();   //点亮 LED
```

```
            }
            else
            {
                Gu8LedStatus=0; //标识并且更改当前 LED 灯的状态。1 就变成 0。
                LedClose();  //关闭 LED
            }

            vGu8SingleKeySec=0;  //响应按键服务处理程序后，按键编号必须清零，避免一致触发
            break;
    }

}

void DoubleKeyTask(void)    //双击按键任务函数，放在主函数内
{
    if(0==vGu8DoubleKeySec)
    {
        return; //按键的触发序号是 0 意味着无按键触发，直接退出当前函数，不执行此函数下面的代码
    }

    switch(vGu8DoubleKeySec) //根据不同的按键触发序号执行对应的代码
    {
       case 1:     //双击任务

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //触发双击后，发出“嘀”一声
            vGu8BeepTimerFlag=1;
            vGu8DoubleKeySec=0;  //响应按键服务处理程序后，按键编号必须清零，避免一致触发
            break;
    }

}
```