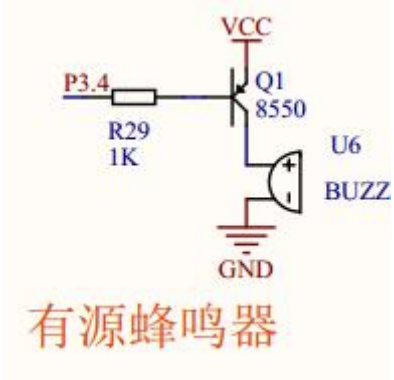
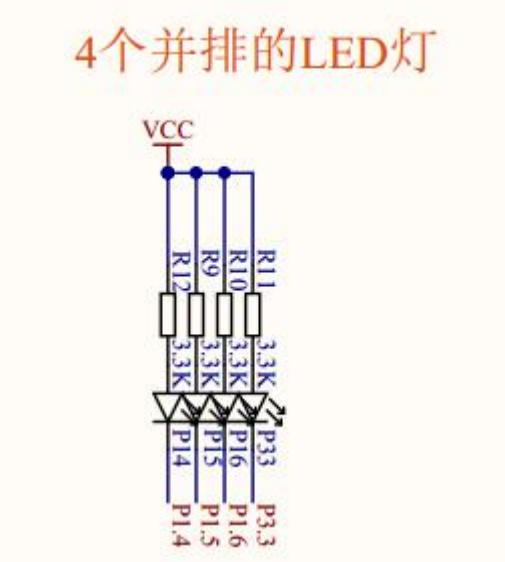
第一百零一节： 矩阵按键鼠标式的单击与双击。

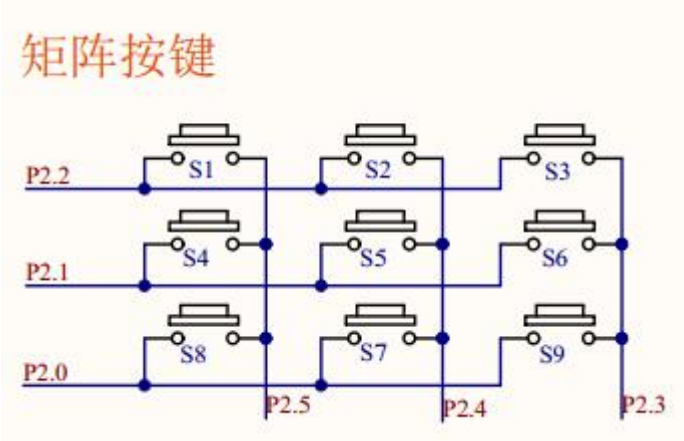
【101.1　矩阵按键鼠标式的单击与双击。】



上图 101.1.1　有源蜂鸣器电路



上图 101.1.2　LED 电路



上图 101.1.3　3*3 矩阵按键的电路

矩阵按键与前面章节独立按键的单击与双击的处理思路是一样的，本节讲矩阵按键的单击与双击，也算是重温之前章节讲的内容。

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：以 S1 按键为例，（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
#define KEY_SHORT_TIME  20     //按键去抖动的“滤波”时间
#define KEY_INTERVAL_TIME  80  //连续两次单击之间的最大有效时间。因为是矩阵，80 不一定是 80ms

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);
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 ROW_INPUT1=P2^2;  //第 1 行输入口。
sbit ROW_INPUT2=P2^1;  //第 2 行输入口。
sbit ROW_INPUT3=P2^0;  //第 3 行输入口。

sbit COLUMN_OUTPUT1=P2^5;  //第 1 列输出口。
sbit COLUMN_OUTPUT2=P2^4;  //第 2 列输出口。
sbit COLUMN_OUTPUT3=P2^3;  //第 3 列输出口。

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
    }
}

/* 注释一：
*  矩阵按键扫描的详细过程：
*  先输出某 1 列低电平，其它 2 列输出高电平，延时等待 2ms 后（等此 3 列输出同步稳定），
*  再分别判断 3 行的输入 IO 口，  如果发现哪一行是低电平，就说明对应的某个按键被触发。
*  依次循环切换输出的 3 种状态，并且分别判断输入的 3 行，就可以检测完 9 个按键。矩阵按键的
*  去抖动处理方法跟我前面讲的独立按键去抖动方法是一样的。
*/

/* 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_SHORT_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 Su16KeyCnt
*         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_SHORT_TIME，马上把自锁标志 Su8KeyLock 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 Su8KeyLock 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 Su8KeyLock=0;
   static unsigned int  Su16KeyCnt=0;
   static unsigned char Su8KeyStep=1;

   static unsigned char Su8ColumnRecord=0;  //用来切换当前列的输出

   static unsigned char Su8KeyTouchCnt1;       //S1 按键的次数记录
   static unsigned int  Su16KeyIntervalCnt1;   //S1 按键的间隔时间计数器

   switch(Su8KeyStep)
   {
     case 1:
          if(0==Su8ColumnRecord)  //按键扫描输出第一列低电平
          {
              COLUMN_OUTPUT1=0;
              COLUMN_OUTPUT2=1;
              COLUMN_OUTPUT3=1;
          }
          else if(1==Su8ColumnRecord)  //按键扫描输出第二列低电平
          {
              COLUMN_OUTPUT1=1;
              COLUMN_OUTPUT2=0;
              COLUMN_OUTPUT3=1;
          }
          else     //按键扫描输出第三列低电平
          {
              COLUMN_OUTPUT1=1;
              COLUMN_OUTPUT2=1;
              COLUMN_OUTPUT3=0;
          }
          Su16KeyCnt=0;  //延时计数器清零
          Su8KeyStep++;  //切换到下一个运行步骤
          break;

     case 2:     //延时等待 2ms 后（等此 3 列输出同步稳定）。不是按键的去抖动延时。
          Su16KeyCnt++;
          if(Su16KeyCnt>=2)
          {
             Su16KeyCnt=0;
             Su8KeyStep++;     //切换到下一个运行步骤
```

```
        }
        break;

    case 3:
        if(1==ROW_INPUT1&&1==ROW_INPUT2&&1==ROW_INPUT3)
        {
           Su8KeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描!!!!!!
           Su8KeyLock=0;  //按键自锁标志清零
           Su16KeyCnt=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，启动间隔时间的计数器
          {
              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，溢出无效
              {
                  Su16KeyIntervalCnt1=0; //时间计数器清零
                  Su8KeyTouchCnt1=0;     //清零按键的按下的次数，因为间隔时间溢出无效
              }
          }

           Su8ColumnRecord++;  //输出下一列
           if(Su8ColumnRecord>=3)
           {
              Su8ColumnRecord=0; //依次输出完第 3 列之后，继续从第 1 列开始输出低电平
           }
        }
        else if(0==Su8KeyLock)  //有按键按下，且是第一次触发
        {
            if(0==ROW_INPUT1&&1==ROW_INPUT2&&1==ROW_INPUT3)
            {
                Su16KeyCnt++;  //去抖动延时计数器
                if(Su16KeyCnt>=KEY_SHORT_TIME)
                {
                    Su8KeyLock=1;//自锁置 1,避免一直触发,只有松开按键,此标志位才会被清零

                    if(0==Su8ColumnRecord)  //第 1 列输出低电平
                    {
                         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，触发 1 号键 对应 S1 键
                         }
                         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，触发 1 号键 对应 S1 键
                    vGu8DoubleKeySec=1;   //双击任务，触发 1 号键 对应 S1 键
                }
            }
            else if(1==Su8ColumnRecord)  //第 2 列输出低电平
            {
                vGu8SingleKeySec=2;  //触发 2 号键 对应 S2 键
            }
            else if(2==Su8ColumnRecord)  //第 3 列输出低电平
            {
                vGu8SingleKeySec=3;  //触发 3 号键 对应 S3 键
            }
        }

    }
    else if(1==ROW_INPUT1&&0==ROW_INPUT2&&1==ROW_INPUT3)
    {
        Su16KeyCnt++;  //去抖动延时计数器
        if(Su16KeyCnt>=KEY_SHORT_TIME)
        {
            Su8KeyLock=1;//自锁置 1,避免一直触发,只有松开按键,此标志位才会被清零

            if(0==Su8ColumnRecord)  //第 1 列输出低电平
            {
                vGu8SingleKeySec=4;  //触发 4 号键 对应 S4 键
            }
            else if(1==Su8ColumnRecord)  //第 2 列输出低电平
            {
                vGu8SingleKeySec=5;  //触发 5 号键 对应 S5 键
            }
            else if(2==Su8ColumnRecord)  //第 3 列输出低电平
            {
                vGu8SingleKeySec=6;  //触发 6 号键 对应 S6 键
            }
        }
    }
    else if(1==ROW_INPUT1&&1==ROW_INPUT2&&0==ROW_INPUT3)
    {
        Su16KeyCnt++;  //去抖动延时计数器
        if(Su16KeyCnt>=KEY_SHORT_TIME)
        {
            Su8KeyLock=1;//自锁置 1,避免一直触发,只有松开按键,此标志位才会被清零
```

```
                    if(0==Su8ColumnRecord)  //第 1 列输出低电平
                    {
                        vGu8SingleKeySec=7;  //触发 7 号键 对应 S7 键
                    }
                    else if(1==Su8ColumnRecord)  //第 2 列输出低电平
                    {
                        vGu8SingleKeySec=8;  //触发 8 号键 对应 S8 键
                    }
                    else if(2==Su8ColumnRecord)  //第 3 列输出低电平
                    {
                        vGu8SingleKeySec=9;  //触发 9 号键 对应 S9 键
                    }
                }
            }

        }
        break;
   }

}

void SingleKeyTask(void)    //按键单击的任务函数，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:     //S1 按键的单击任务

           //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，避免一直触发
            break;

        default:  //其它按键触发的单击

            vGu8SingleKeySec=0;  //响应按键服务处理程序后，按键编号必须清零，避免一直触发
            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:     //S1 按键的双击任务

            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;
    }

}

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)
{
/* 注释三：
* 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();
                }

          }
      }
}
```