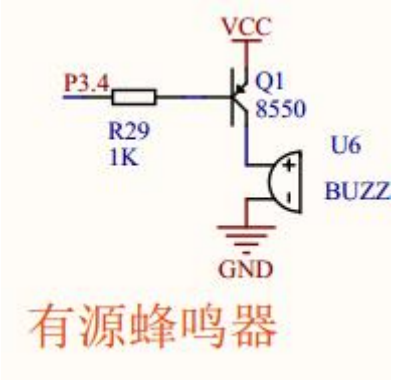
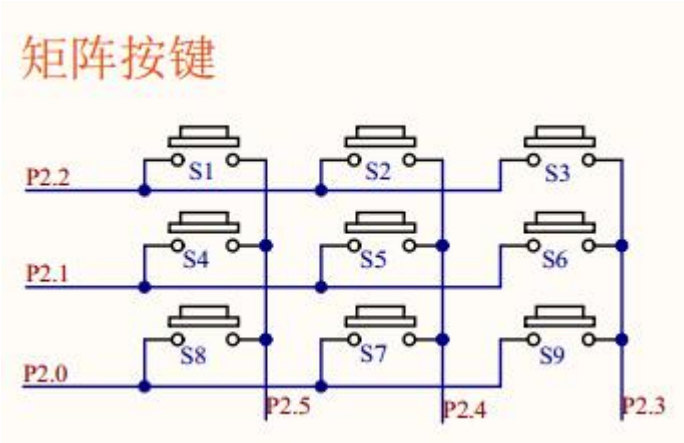
第九十九节： “行列扫描式”矩阵按键的单个触发（原始版）。

【99.1 “行列扫描式”矩阵按键。】



上图 99.1.1 有源蜂鸣器电路



上图 99.1.2 3*3 矩阵按键的电路

上图是 3*3 的矩阵按键电路，其它 4*4 或者 8*8 的矩阵电路原理是一样的，编程思路也是一样的。相对独立按键，矩阵按键因为采用动态行列扫描的方式，能更加节省 IO 口，比如 3*3 的 3 行 3 列，1 行占用 1 根 IO 口，1 列占用 1 根 IO 口，因此 3*3 矩阵按键占用 6 个 IO 口（3+3=6），但是能识别 9 个按键（3*3=9）。同理，8*8 矩阵按键占用 16 个 IO 口（8+8=16），但是能识别 64 个按键（8*8=64）。

矩阵按键的编程原理。如上图 3*3 矩阵按键的电路，行 IO 口（P2.2，P2.1，P2.0）定为输入，列 IO 口（P2.5，P2.4，P2.3）定为输出。同一时刻，列输出的 3 个 IO 口只能有 1 根是输出 L（低电平），其它 2 根必须全是 H（高电平），然后依次轮番切换输出状态，列输出每切换一次，就分别读取一次行输入的 3 个 IO 口，这样一次就能识别到 3 个按键的状态，如果列连续切换 3 次就可以读取全部 9 个按键的状态。列的 3 种输出状态分别是：（P2.5 为 L，P2.4 为 H，P2.3 为 H），（P2.5 为 H，P2.4 为 L，P2.3 为 H），（P2.5 为 H，P2.4 为 H，P2.3 为 L）。为什么列输出每切换一次就能识别到 3 个按键的状态？因为，首先要明白一个前提，在没有任何按键“被按下”的时候，行输入的 3 个 IO 口因为内部上拉电阻的作用，默认状态都是 H 电平。并且，H 与 H 相互短接输出为 H，H 与 L 相互短接输出 L，也就是，L（低电平）的优先级最大，任何 H（高电平）碰到 L（低电平）输出的结果都是 L（低电平）。L（低电平）就像数学乘法运算里的数字 0，任何数跟 0 相乘必然等于 0。多说一句，这个“L 最高优先级”法则是有前提的，就是 H（高电平）的产生必须是纯粹

靠上拉电阻拉高的 H（高电平）才行，比如刚好本教程所用的 51 单片机内部 IO 口输出的 H（高电平）是依靠内部的上拉电阻产生，如果是其它“非上拉电阻产生的高电平”与“低电平”短接就有“短路烧坏芯片”的风险，这时就需要额外增加“三极管开漏式输出”电路或者外挂“开漏式输出集成芯片”电路。继续回到正题，为什么列输出每切换一次就能识别到 3 个按键的状态？举个例子，比如当列输出状态处于（P2.5 为 L，P2.4 为 H，P2.3 为 H）下，我们读取行输入的 P2.2 口，行输入的 P2.2 与列输出 P2.5，P2.4，P2.3 的“交叉处”有 3 个按键 S1，S2，S3，此时，如果 P2.2 口是 L（低电平），那么必然是 S1“被按下”，因为想让 P2.2 口是 L，只有 S1 有这个能力，而如果 S1 没有“被按下”，另外两个 S2，S3 即使“被按下”，P2.2 口也是 H 而绝对不会为 L，因为 S2，S3 的列输出 P2.4 为 H，P2.3 为 H，H 与 H 相互短接输出的结果必然为 H。

本节例程实现的功能：9 个矩阵按键，每按下 1 个按键都触发一次蜂鸣器鸣叫。

```
#include "REG52.H"

#define KEY_VOICE_TIME   50

#define KEY_SHORT_TIME  20     //按键去抖动的“滤波”时间

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

void BeepOpen(void);
void BeepClose(void);

void VoiceScan(void);
void KeyScan(void);
void KeyTask(void);

sbit P3_4=P3^4;       //蜂鸣器

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;



volatile unsigned char vGu8KeySec=0;  //按键的触发序号
```

```
void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
        KeyTask();       //按键的任务函数
    }
}

/* 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，不再重复多讲。
*/

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;

   switch(Su8KeyStep)
   {
     case 1:   //按键扫描输出第一列低电平
          COLUMN_OUTPUT1=0;
          COLUMN_OUTPUT2=1;
          COLUMN_OUTPUT3=1;

          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++;  //如果没有按键按下，切换到下一个运行步骤
        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
     }
     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)
             {
                 Su16KeyCnt=0;
                 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
                 vGu8KeySec=1;  //触发 1 号键 对应 S1 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)
             {
                 Su16KeyCnt=0;
                 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
                 vGu8KeySec=2;  //触发 2 号键 对应 S2 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)
             {
                 Su16KeyCnt=0;
                 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
                 vGu8KeySec=3;  //触发 3 号键 对应 S3 键
             }
         }

     }
     break;
```

```
case 4:   //按键扫描输出第二列低电平
     COLUMN_OUTPUT1=1;
     COLUMN_OUTPUT2=0;
     COLUMN_OUTPUT3=1;

     Su16KeyCnt=0;  //延时计数器清零
     Su8KeyStep++;  //切换到下一个运行步骤
     break;

case 5:     //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 6:
     if(1==ROW_INPUT1&&1==ROW_INPUT2&&1==ROW_INPUT3)
     {
        Su8KeyStep++;  //如果没有按键按下，切换到下一个运行步骤
        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
     }
     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)
             {
                 Su16KeyCnt=0;
                 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
                 vGu8KeySec=4;  //触发 4 号键 对应 S4 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)
             {
                 Su16KeyCnt=0;
```

```
                    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
                    vGu8KeySec=5;  //触发 5 号键 对应 S5 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)
                {
                    Su16KeyCnt=0;
                    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
                    vGu8KeySec=6;  //触发 6 号键 对应 S6 键
                }
            }

        }
        break;
    case 7:   //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 8:     //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 9:
        if(1==ROW_INPUT1&&1==ROW_INPUT2&&1==ROW_INPUT3)
        {
           Su8KeyStep=1;  //如果没有按键按下，返回到第一步，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
        }
        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)
                {
                    Su16KeyCnt=0;
                    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
                    vGu8KeySec=7;  //触发 7 号键 对应 S7 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)
                {
                    Su16KeyCnt=0;
                    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
                    vGu8KeySec=8;  //触发 8 号键 对应 S8 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)
                {
                    Su16KeyCnt=0;
                    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
                    vGu8KeySec=9;  //触发 9 号键 对应 S9 键
                }
            }

        }
        break;
    }

}

void KeyTask(void)    //按键任务函数，放在主函数内
{
    if(0==vGu8KeySec)
    {
        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(vGu8KeySec) //根据不同的按键触发序号执行对应的代码
{
   case 1:     //S1 触发的任务

        vGu8BeepTimerFlag=0;
        vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
        vGu8BeepTimerFlag=1;

        vGu8KeySec=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;

   case 2:     //S2 触发的任务

        vGu8BeepTimerFlag=0;
        vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
        vGu8BeepTimerFlag=1;

        vGu8KeySec=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;

   case 3:     //S3 触发的任务

        vGu8BeepTimerFlag=0;
        vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
        vGu8BeepTimerFlag=1;

        vGu8KeySec=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;

   case 4:     //S4 触发的任务

        vGu8BeepTimerFlag=0;
        vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
        vGu8BeepTimerFlag=1;

        vGu8KeySec=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;

   case 5:     //S5 触发的任务

        vGu8BeepTimerFlag=0;
        vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
        vGu8BeepTimerFlag=1;
```

```
            vGu8KeySec=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;

        case 6:     //S6 触发的任务

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
            vGu8BeepTimerFlag=1;

            vGu8KeySec=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;

        case 7:     //S7 触发的任务

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
            vGu8BeepTimerFlag=1;

            vGu8KeySec=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;

        case 8:     //S8 触发的任务

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
            vGu8BeepTimerFlag=1;

            vGu8KeySec=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;

        case 9:     //S9 触发的任务

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
            vGu8BeepTimerFlag=1;

            vGu8KeySec=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 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)
{

}

void BeepOpen(void)
{
    P3_4=0;
}

void BeepClose(void)
{
    P3_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();
            }

        }
    }
}
```