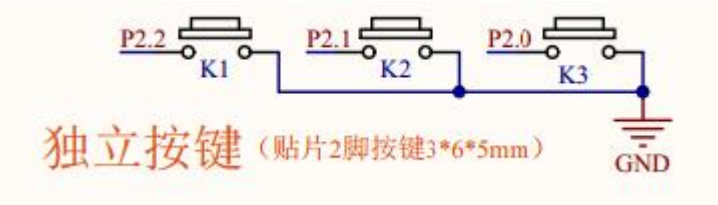
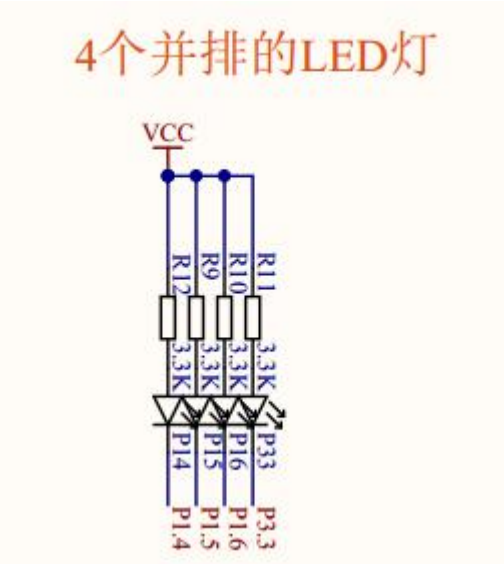
第一百一十一节： 工业自动化设备的开关信号的运动控制。

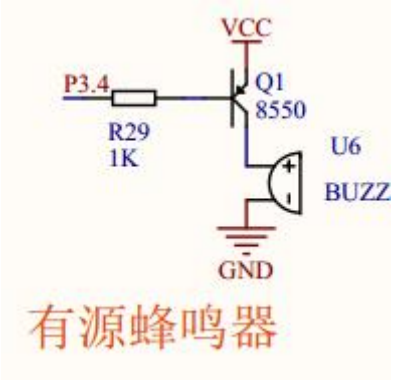
【111.1　开关信号的运动控制。】



上图 111.1.1　独立按键



上图 111.1.2　LED 电路



上图 111.1.3　有源蜂鸣器的电路

本节涉及的知识点有，switch 的过程控制，时间延时，开关感应器的软件滤波，工件计数器，以及整体的软件框架。

现在有一台设备，水平方向有一个滑块，能左右移动，滑块上安装了一个能垂直伸缩的“机械手”。按下启动按键后，滑块先从左边往右边移动，移到最右边碰到“右感应器”后，滑块上的“机械手”开始往下

移动 2 秒，移动 2 秒后开始原路返回，“机械手”向上移动，碰到“上感应器”后，滑块开始往左边移动，移动 3 秒后默认已经回到原位最左边，此时“计数器”累加 1，完成一次过程，如果再按下启动按键，继续重复这个过程。

这个设备用了 2 个气缸。1 个“水平气缸”驱动滑块水平方向的左右移动，当控制“水平气缸”的输出信号为 0 时往左边跑，当控制“水平气缸”的输出信号为 1 时往右边跑。另 1 个“垂直气缸”驱动“机械手”的上下移动，当控制“垂直气缸”的输出信号为 0 时往上边跑，当控制“垂直气缸”的输出信号为 1 时往下边跑。

这个设备用了 2 个开关感应器。分别是“右感应器”和“上感应器”。当感应器没有被碰到的时候信号为 1，当感应器被碰到的时候信号为 0。

这个设备用了 1 个独立按键。控制运动的启动。

2 个气缸是输出信号，用 P1.4 和 P1.5 所控制的两个 LED 模拟。2 个开关感应器是输入信号，用 K2 和 K3 这两个独立按键模拟。1 个独立按键用 K1 按键。如上图。

```
#include "REG52.H"

#define KEY_VOICE_TIME   50
#define KEY_FILTER_TIME  25
#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 BeepOpen(void);
void BeepClose(void);

void GoLeft(void) ; //“水平气缸”往左跑
void GoRight(void); //“水平气缸”往右跑
void GoUp(void);    //“垂直气缸”往上跑
void GoDown(void);  //“垂直气缸”往下跑

void VoiceScan(void);
void SensorScan(void);  //开关感应器的消抖，在定时中断里调用处理
void KeyScan(void);
void KeyTask(void);
void RunTask(void);    //运动控制的任务函数

sbit P1_4=P1^4;  //水平气缸的输出
sbit P1_5=P1^5;  //垂直气缸的输出

sbit P3_4=P3^4;  //蜂鸣器的输出口
```

```
sbit KEY_INPUT1=P2^2;  //【启动】按键 K1 的输入口。

sbit SensorRight_sr=P2^1;   //右感应器的输入口
sbit SensorUp_sr=P2^0;      //上感应器的输入口

volatile unsigned char vGu8SensorRight=0;  //右感应器经过滤波后的当前电平状态。
volatile unsigned char vGu8SensorUp=0;  //上感应器经过滤波后的当前电平状态。

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

volatile unsigned char vGu8KeySec=0;

unsigned char Gu8RunStart=0;      //启动的总开关
unsigned char Gu8RunStatus=0;     //运动的状态，0 为停止，1 为运行

unsigned int  Gu16RunCnt=0;       //计数器
unsigned int  Gu16ReturnLeftTime=3000;   //水平往左跑的延时变量，默认为 3 秒
unsigned int  Gu16GoDownTime=2000;       //垂直往下跑的延时变量，默认为 2 秒

volatile unsigned char vGu8RunTimerFlag=0;   //用于控制运动过程中的延时的定时器
volatile unsigned int vGu16RunTimerCnt=0;

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
        KeyTask();    //按键的任务函数
        RunTask();    //运动控制的任务函数
    }
}

/* 注释一：
*  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 Su16SensorRight_H_Cnt=0;  //判断高电平的计时器
```

```
static unsigned int Su16SensorRight_L_Cnt=0;  //判断低电平的计时器

static unsigned int Su16SensorUp_H_Cnt=0;  //判断高电平的计时器
static unsigned int Su16SensorUp_L_Cnt=0;  //判断低电平的计时器

//右感应器的滤波
if(0==SensorRight_sr)
{
     Su16SensorRight_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!
     Su16SensorRight_L_Cnt++;
     if(Su16SensorRight_L_Cnt>=SENSOR_TIME)
     {
         Su16SensorRight_L_Cnt=0;
         vGu8SensorRight=0;   //此全局变量反馈经过滤波后“右感应器”当前电平的状态
     }

 }
 else
 {
     Su16SensorRight_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!
     Su16SensorRight_H_Cnt++;
     if(Su16SensorRight_H_Cnt>=SENSOR_TIME)
     {
         Su16SensorRight_H_Cnt=0;
         vGu8SensorRight=1;  //此全局变量反馈经过滤波后“右感应器”当前电平的状态
     }
 }

//上感应器的滤波
if(0==SensorUp_sr)
{
     Su16SensorUp_H_Cnt=0;
     Su16SensorUp_L_Cnt++;
     if(Su16SensorUp_L_Cnt>=SENSOR_TIME)
     {
         Su16SensorUp_L_Cnt=0;
         vGu8SensorUp=0;   //此全局变量反馈经过滤波后“上感应器”当前电平的状态
     }

 }
 else
 {
     Su16SensorUp_L_Cnt=0;
     Su16SensorUp_H_Cnt++;
```

```
            if(Su16SensorUp_H_Cnt>=SENSOR_TIME)
            {
                Su16SensorUp_H_Cnt=0;
                vGu8SensorUp=1;  //此全局变量反馈经过滤波后“上感应器”当前电平的状态
            }
        }


}

void T0_time() interrupt 1
{
    VoiceScan();
    KeyScan();
    SensorScan();  //用来识别和滤波开关感应器

    if(1==vGu8RunTimerFlag&&vGu16RunTimerCnt>0)  //用于控制运动延时的定时器
    {
        vGu16RunTimerCnt--;
    }

    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)
{
     //上电初始化气缸的开机位置
```

```
      GoLeft() ;     //“水平气缸”往左跑，上电初始化时滑块处于左边
      GoUp();        //“垂直气缸”往上跑，上电初始化时“机械臂”处于上方

}

void BeepOpen(void)
{
    P3_4=0;
}

void BeepClose(void)
{
    P3_4=1;
}

void GoLeft(void) //“水平气缸”往左跑
{
    P1_4=0;
}

void GoRight(void) //“水平气缸”往右跑
{
    P1_4=1;
}

void GoUp(void)  //“垂直气缸”往上跑
{
    P1_5=0;
}

void GoDown(void) //“垂直气缸”往下跑
{
    P1_5=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();
               }

          }
      }
}

void KeyScan(void)  //此函数放在定时中断里每 1ms 扫描一次
{
   static unsigned char Su8KeyLock1;
   static unsigned int  Su16KeyCnt1;

   //【启动】按键 K1 的扫描识别
   if(0!=KEY_INPUT1)
   {
      Su8KeyLock1=0;
      Su16KeyCnt1=0;
   }
   else if(0==Su8KeyLock1)
   {
      Su16KeyCnt1++;
      if(Su16KeyCnt1>=KEY_FILTER_TIME)
      {
         Su8KeyLock1=1;
         vGu8KeySec=1;    //触发 1 号键
      }
   }

}

/* 注释二：
*  在 KeyTask 中只改变 Gu8RunStart 的值，用于总启动开关。而运动状态 Gu8RunStatus 是在运动函数
```

```
*  RunTask 中改变，用于对外反馈实时的运动状态。
*/

void KeyTask(void)    //按键的任务函数，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:     //1 号按键。【启动】按键 K1

            if(0==Gu8RunStatus) //根据当前运动的状态来决定“总开关”是否能受按键的控制
            {
                Gu8RunStart=1;   //总开关“打开”。
            }

            vGu8BeepTimerFlag=0;
            vGu16BeepTimerCnt=KEY_VOICE_TIME;  //触发按键后，发出固定长度的声音
            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;

       default:
            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;

    }
}

/* 注释三：
*  本节故意引入三个变量：计数器 Gu16RunCnt，左延时 Gu16ReturnLeftTime，下延时 Gu16GoDownTime。
*  在人机界面的场合，这三个变量可以用来扩展实现设置参数的功能。比如，如果有数码管，可以通过
*  显示 Gu16RunCnt 的数值来让客户看到当前设备的计数器。如果有数码管和按键，可以通过切换到某个
*  界面下，修改 Gu16ReturnLeftTime 和 Gu16GoDownTime 的数值，让客户对设备进行延时参数的设置。
*/

void RunTask(void)    //运动控制的任务函数，放在主函数内
{
    static unsigned char Su8RunStep=0; //运行的步骤
```

```
//当总开关处于“停止”并且“步骤不为0”时，强制把步骤归零。
if(0!=Su8RunStep&&0==Gu8RunStart)
{
      Su8RunStep=0; //步骤归零
}

switch(Su8RunStep) //屡见屡爱的 switch 又来了
{
   case 0:
       if(1==Gu8RunStart) //总开关“打开”
        {
            Gu8RunStatus=1;  //及时设置 Gu8RunStatus 的运动状态为“运行”

            GoRight() ;       //“水平气缸”往右跑。P1.4 的 LED 灯“灭”。

            Su8RunStep=1;  //切换到下一步
        }
       break;
   case 1:
       if(0==vGu8SensorRight) //直到碰到了“右感应器”（按下 K2），“机械臂”才往下移动。
        {
            GoDown();  //“垂直气缸”往下跑。P1.5 的 LED 灯“灭”。
            vGu8RunTimerFlag=0;
            vGu16RunTimerCnt=Gu16GoDownTime; //向下移动 3 秒的延时赋值
            vGu8RunTimerFlag=1;   //启动定时器

            Su8RunStep=2;  //切换到下一步
        }
       break;
   case 2:
       if(0==vGu16RunTimerCnt) //当定时的 3 秒时间到，“机械臂”才往上移动，开始原路返回。
        {
            GoUp();    //“垂直气缸”往上跑。P1.5 的 LED 灯“亮”。
            Su8RunStep=3;  //切换到下一步
        }
       break;
   case 3:
       if(0==vGu8SensorUp) //直到碰到了“上感应器”（按下 K3），滑块才往左移动。
        {
            GoLeft() ; //“水平气缸”往左跑。P1.4 的 LED 灯“亮”。
            vGu8RunTimerFlag=0;
            vGu16RunTimerCnt=Gu16ReturnLeftTime; //向左移动 2 秒的延时赋值
            vGu8RunTimerFlag=1;   //启动定时器
```

```
                Su8RunStep=4;  //切换到下一步
            }

            break;
        case 4:
            if(0==vGu16RunTimerCnt) //当定时的 2 秒时间到，完成一次过程。
            {
                Gu16RunCnt++;   //计数器加 1，统计设备运行的次数
                Gu8RunStatus=0;  //及时设置 Gu8RunStatus 的运动状态为“停止”
                Gu8RunStart=0;  //总开关“关闭”，为下一次启动作准备
                Su8RunStep=0;   //步骤变量清零，为下一次启动作准备
            }
            break;
    }
}
```