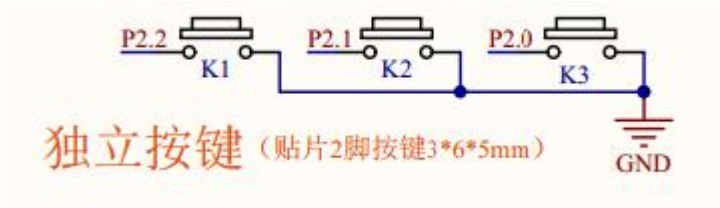
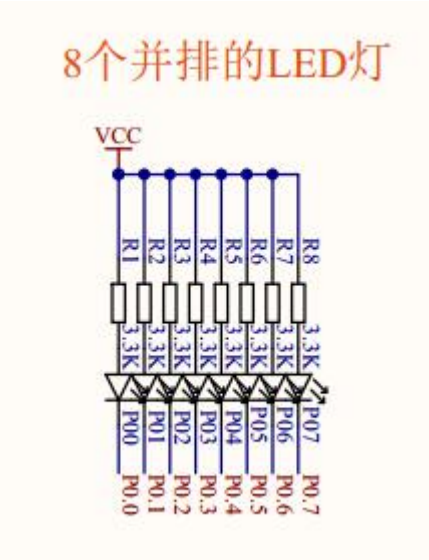
第九十八节： 独立按键按住不松手的“先加速后均速”的触发。

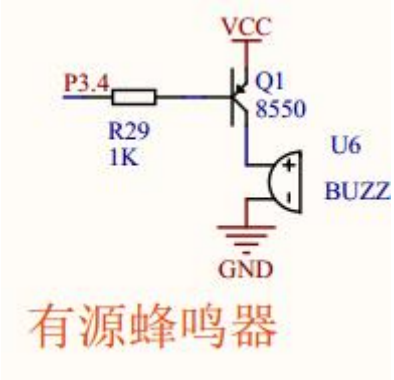
【98.1 “先加速后均速”的触发。】



上图 98.1.1 独立按键电路



上图 98.1.2 灌入式驱动 8 个 LED



上图 98.1.3 有源蜂鸣器电路

当“连续加”或者“连续减”的数据范围很大的时候，就需要按键的加速与匀速相结合的触发方式。“加速”是指按住按键不松手，按键刚开始触发是从慢到快的渐进过程，当“加速”到某个特别快的速度的时候，

就“不再加速”，而是以该“恒定高速”进行“连续匀速”触发。这种触发方式，“加速”和“匀速”是相辅相成的，缺一不可。为什么？假如没有“加速”只有“匀速”，那么刚按下按键就直接以最高速的“匀速”进行，就会跑过头，缺乏微调功能；而假如没有“匀速”只有“加速”，那么按下按键不松手后，速度就会一直不断飙升，最后失控过冲。

本节例程实现的功能如下：

（1）要更改一个“设置参数”（一个全局变量），参数的范围是 0 到 800。

（2）8 个受该“设置参数”控制的跑马灯在某一时刻只有 1 个 LED 亮，每触发一次 K1 按键，该“设置参数”就自减 1，直到减到最小值 0 为止；相反，每触发一次 K2 按键，该“设置参数”就自加 1，直到加到最大值 800 为止。

（3）LED 灯实时显示该“设置参数”的范围状态：

只有第 0 个 LED 灯亮：0<=“设置参数”<100。

只有第 1 个 LED 灯亮：100<=“设置参数”<200。

只有第 2 个 LED 灯亮：200<=“设置参数”<300。

只有第 3 个 LED 灯亮：300<=“设置参数”<400。

只有第 4 个 LED 灯亮：400<=“设置参数”<500。

只有第 5 个 LED 灯亮：500<=“设置参数”<600。

只有第 6 个 LED 灯亮：600<=“设置参数”<700。

只有第 7 个 LED 灯亮：700<=“设置参数”<=800。

（4）按键每“单击”一次蜂鸣器就鸣叫一次，但是，当按键“从单击进入连击”后，蜂鸣器就不鸣叫。

```
#include "REG52.H"

#define KEY_VOICE_TIME   50

#define KEY_SHORT_TIME  25      //按键单击的“滤波”时间
#define KEY_ENTER_CONTINUITY_TIME    300  //按键“从单击进入连击”的间隔时间
#define KEY_CONTINUITY_INITIAL_TIME    80  //按键“连击”起始的预设间隔时间
#define KEY_SUB_DT_TIME     8      //按键在“加速”时每次减小的时间。
#define KEY_CONTINUITY_MIN_TIME   10  //按键时间减小到最后的“匀速”间隔时间。



#define BUS_P0    P0     //8 个 LED 灯一一对应单片机的 P0 口总线

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);
```

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

sbit P3_4=P3^4;       //蜂鸣器

sbit KEY_INPUT1=P2^2;  //K1 按键识别的输入口。
sbit KEY_INPUT2=P2^1;  //K2 按键识别的输入口。

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

unsigned int Gu16SetData=0; //“设置参数”。范围从 0 到 800。LED 灯反映该当前值的范围状态
unsigned char Gu8DisplayUpdate=1; //显示的刷新标志

volatile unsigned char vGu8KeySec=0;  //按键的触发序号
volatile unsigned char vGu8ShieldVoiceFlag=0;  //屏蔽声音的标志

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



/* 注释一：
* Gu8DisplayUpdate 这类“显示刷新变量”在“显示框架”里是很常见的，而且屡用屡爽。
* 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）
{
    if(1==Gu8DisplayUpdate)  //需要刷新一次显示
    {
        Gu8DisplayUpdate=0;  //及时清零，避免主函数“不断去执行显示代码”而影响程序效率

        if(Gu16SetData<100)
        {
            BUS_P0=~(1<<0);  //第 0 个灯亮
        }
        else if(Gu16SetData<200)
```

```
        {
            BUS_P0=~(1<<1);  //第 1 个灯亮
        }
        else if(Gu16SetData<300)
        {
            BUS_P0=~(1<<2);  //第 2 个灯亮
        }
        else if(Gu16SetData<400)
        {
            BUS_P0=~(1<<3);  //第 3 个灯亮
        }
        else if(Gu16SetData<500)
        {
            BUS_P0=~(1<<4);  //第 4 个灯亮
        }
        else if(Gu16SetData<600)
        {
            BUS_P0=~(1<<5);  //第 5 个灯亮
        }
        else if(Gu16SetData<700)
        {
            BUS_P0=~(1<<6);  //第 6 个灯亮
        }
        else
        {
            BUS_P0=~(1<<7);  //第 7 个灯亮
        }
    }
}

/* 注释二：
* 按键“先加速后均速”的识别过程：
* 第一步：每次按下去触发一次单击按键，如果按下去到松手的时间不超过 1 秒，则不会进入连续加速触发
          模式。
* 第二步：如果按下去不松手的时间超过 1 秒，则进入连续加速触发模式。按键触发节奏不断加快，蜂鸣器
*          鸣叫的节奏也不断加快。直到它们都到达某个极限值，然后以此极限值间隔匀速触发。
*/

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;
   static unsigned int  Su16KeyContinuityCnt1;  //连击计数器
   static unsigned int  Su16KeyContinuityTime1=KEY_CONTINUITY_INITIAL_TIME;  //动态时间阀值
```

```
static unsigned char Su8KeyLock2;
static unsigned int  Su16KeyCnt2;
static unsigned int  Su16KeyContinuityCnt2;  //连击计数器
static unsigned int  Su16KeyContinuityTime2=KEY_CONTINUITY_INITIAL_TIME;  //动态时间阀值

//K1 按键
if(0!=KEY_INPUT1)//单个 K1 按键没有按下，及时清零一些标志。
{
   Su8KeyLock1=0; //按键解锁
   Su16KeyCnt1=0;  //去抖动延时计数器清零，此行非常巧妙，是全场的亮点。
   Su16KeyContinuityCnt1=0;  //连击计数器
   Su16KeyContinuityTime1=KEY_CONTINUITY_INITIAL_TIME;  //动态时间阀值。重装初始值。
}
else if(0==Su8KeyLock1)//单个按键 K1 被按下
{
   Su16KeyCnt1++; //累加定时中断次数，每一次累加额度是 1ms
   if(Su16KeyCnt1>=KEY_SHORT_TIME) //按键的“滤波”时间 25ms
   {
         Su8KeyLock1=1;      //“自锁”
         vGu8KeySec=1;       //触发一次 K1 按键
         Su16KeyCnt1=0;      //清零，为了下一步用来累加“从单击进入连击的间隔时间 300ms”
   }
}
else if(Su16KeyCnt1<=KEY_ENTER_CONTINUITY_TIME)//按住不松手累加到 300ms
{
   Su16KeyCnt1++; //累加定时中断次数，每一次累加额度是 1ms
}
else  //按住累加到 300ms 后仍然不放手，这个时候进入有节奏的连续触发
{
    Su16KeyContinuityCnt1++; //连击计数器开始累加，每一次累加额度是 1ms
    if(Su16KeyContinuityCnt1>=Su16KeyContinuityTime1)  //按住没松手，每隔一会就触发一次
    {
         Su16KeyContinuityCnt1=0; //清零，为了继续连击。
         vGu8KeySec=1;       //触发一次 K1 按键
         vGu8ShieldVoiceFlag=1;  //把当前按键触发的声音屏蔽掉
         if(Su16KeyContinuityTime1>=KEY_SUB_DT_TIME)
         {
             //此数值不断被减小，按键的触发速度就不断变快
             Su16KeyContinuityTime1=Su16KeyContinuityTime1-KEY_SUB_DT_TIME;//变快节奏
         }

         if(Su16KeyContinuityTime1<KEY_CONTINUITY_MIN_TIME) //最小间隔时间就是“高速匀速”
         {
```

```
            Su16KeyContinuityTime1=KEY_CONTINUITY_MIN_TIME;
        }

    }

 }

//K2 按键
if(0!=KEY_INPUT2)
{
   Su8KeyLock2=0;
   Su16KeyCnt2=0;
   Su16KeyContinuityCnt2=0;
   Su16KeyContinuityTime2=KEY_CONTINUITY_INITIAL_TIME;
}
else if(0==Su8KeyLock2)
{
   Su16KeyCnt2++;
   if(Su16KeyCnt2>=KEY_SHORT_TIME)
   {
         Su8KeyLock2=1;
         vGu8KeySec=2;       //触发一次 K2 按键
         Su16KeyCnt2=0;
   }
}
else if(Su16KeyCnt2<=KEY_ENTER_CONTINUITY_TIME)
{
   Su16KeyCnt2++;
}
else
{
    Su16KeyContinuityCnt2++;
    if(Su16KeyContinuityCnt2>=Su16KeyContinuityTime2)
    {
         Su16KeyContinuityCnt2=0;
         vGu8KeySec=2;       //触发一次 K2 按键
         vGu8ShieldVoiceFlag=1;
         if(Su16KeyContinuityTime2>=KEY_SUB_DT_TIME)
         {
             Su16KeyContinuityTime2=Su16KeyContinuityTime2-KEY_SUB_DT_TIME;
         }

         if(Su16KeyContinuityTime2<KEY_CONTINUITY_MIN_TIME)
         {
```

```
                Su16KeyContinuityTime2=KEY_CONTINUITY_MIN_TIME;
            }
        }
    }
}

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:     //K1 触发的任务
            if(Gu16SetData>0)
            {
                Gu16SetData--;     //“设置参数”
                Gu8DisplayUpdate=1;  //刷新显示
            }

            if(0==vGu8ShieldVoiceFlag) //声音没有被屏蔽
            {
                vGu8BeepTimerFlag=0;
                vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
                vGu8BeepTimerFlag=1;
            }

            vGu8ShieldVoiceFlag=0;  //及时把屏蔽标志清零，避免平时正常的单击声音也被淹没。
            vGu8KeySec=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;

       case 2:     //K2 触发的任务
            if(Gu16SetData<800)
            {
                Gu16SetData++;       //“设置参数”
                Gu8DisplayUpdate=1;  //刷新显示
            }

            if(0==vGu8ShieldVoiceFlag) //声音没有被屏蔽
            {
                vGu8BeepTimerFlag=0;
                vGu16BeepTimerCnt=KEY_VOICE_TIME;  //发出“嘀”一声
```

```
                vGu8BeepTimerFlag=1;
            }

            vGu8ShieldVoiceFlag=0;  //及时把屏蔽标志清零，避免平时正常的单击声音也被淹没。
            vGu8KeySec=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)
{

}

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();
               }

          }
      }
}
```