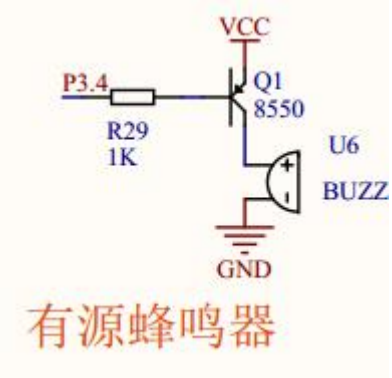
第九十一节： 蜂鸣器的“非阻塞”驱动。

【91.1　蜂鸣器的硬件电路简介。】



上图 91.1.1　PNP 三极管驱动有源蜂鸣器

蜂鸣器有两种，一种是有源蜂鸣器，一种是无源蜂鸣器。有源蜂鸣器的驱动最简单，只要通电就一直响，断电就停，跟驱动 LED 灯一样。无源蜂鸣器则不一样，无源蜂鸣器一直断电不响，奇怪的是一直通电也不响，只有“通，关，通，关...”反复通电关电的状态，才会持续发生稳定的声音，此方式称为脉冲驱动方式，或者 PWM 驱动方式。本教程用的是有源蜂鸣器。

蜂鸣器的驱动电路也有两种常用的方式，一种是 NPN 三极管驱动，一种是 PNP 三极管驱动。NPN 三极管驱动电路，单片机输出“1”（高电平）蜂鸣器导通，输出“0”（低电平）蜂鸣器关闭。而 PNP 三极管驱动电路恰恰相反，单片机输出“0”（低电平）蜂鸣器导通，输出“1”（高电平）蜂鸣器关闭。本教程所用的是 PNP 三极管驱动电路，如上图。

【91.2　“非阻塞”驱动程序。】

“驱动层”是相对“应用层”而言。“应用层”发号施令，“驱动层”负责执行。一个好的“驱动层”必须给“应用层”提供快捷便利的调用接口，此接口可以是函数或者全局变量。本节驱动蜂鸣器所用的是全局变量 vGu16BeepTimerCnt。“应用层”只需给 vGu16BeepTimerCnt 赋值，就可以控制蜂鸣器发声，赋值越大，发声越长，500 代表发声 500ms，1000 代表发声 1000ms，具体细节实现，则由“驱动层”的驱动函数负责执行，驱动函数放在定时中断函数里定时扫描。为什么不把驱动函数放到 main 函数的循环里去？因为放在定时中断里，能保证蜂鸣器的声音长度是一致的，如果放在 main 循环里，声音的长度有可能在某些项目中受到某些必须一气呵成的任务干扰，得不到及时响应，影响声音长度的一致性。下面代码实现的功能是，单片机只要一上电，蜂鸣器就发出一次 1000ms 长度的“嘀”声音。

```
#include "REG52.H"

#define BEEP_TIME  1000   //控制蜂鸣器发声的长度，此处是 1000ms

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

```
void BeepOpen(void);   //蜂鸣器发声
void BeepClose(void);  //蜂鸣器关闭
void VoiceScan(void);  //蜂鸣器的驱动函数，放在定时中断里

sbit P3_4=P3^4;  //控制蜂鸣器的 IO 口。0 代表发声，1 代表关闭。

volatile unsigned char vGu8BeepTimerFlag=0;
volatile unsigned int vGu16BeepTimerCnt=0;  //控制蜂鸣器发声长度的计时器

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();     //此函数内部有“应用层”的赋值操作，控制上电的声音长度。
    while(1)
    {
       ;
    }
}

void T0_time() interrupt 1
{
    VoiceScan();  //蜂鸣器的驱动函数

    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)
{
    vGu8BeepTimerFlag=0;
    vGu16BeepTimerCnt=BEEP_TIME;  //“应用层”只需赋值，一上电，蜂鸣器发出 1000ms 长度的声音。
    vGu8BeepTimerFlag=1;

}



//蜂鸣器发声
void BeepOpen(void)
{
    P3_4=0;  //0 代表发声
}

//蜂鸣器关闭
void BeepClose(void)
{
    P3_4=1;  //1 代表关闭
}

//蜂鸣器的驱动函数，放在定时中断函数里每定时 1ms 扫描一次。
void VoiceScan(void)
{
      //Su8Lock 的作用是避免 BeepOpen()被重复扫描影响效率，发声时只执行一次此函数即可。
      //同时，也巧妙借用 else 结构，实现逻辑顺序分解成“先发声，下一次再开始定时”的两个步骤。

      static unsigned char Su8Lock=0;

      if(1==vGu8BeepTimerFlag&&vGu16BeepTimerCnt>0)
      {
          if(0==Su8Lock)
          {
               Su8Lock=1;  //进入触发声音后就自锁起来
               BeepOpen(); //发声，此处封装成函数，为了今后代码的移植性。
          }
          else  //巧妙借用 else 结构，实现先发声，下一次中断再开始计时的逻辑顺序。比如，
          {     //如果赋值 1，就能确保有 1ms 的计时发声。

               vGu16BeepTimerCnt--;     //定时器自减，控制蜂鸣器发声的时间长度

               if(0==vGu16BeepTimerCnt)
               {
```

```
                    Su8Lock=0;      //关闭声音后，及时解锁，为下一次触发做准备
                    BeepClose();  //关闭声音，此处封装成函数，为了今后代码的移植性。
                }

            }
        }
}
```