第八十二节： Delay 阻塞延时控制 LED 闪烁。

【82.1　“阻塞”与“非阻塞”。】

做项目写程序，大框架大思路上就是在“阻塞”与“非阻塞”这两种模式下不断切换。“阻塞”可以理解成“单任务处理”模式，“非阻塞”可以理解成“多任务并行处理”模式。“阻塞”的优点是它全神贯注不分心地专注于当下这一件事，它等待某个事件的响应速度是最快的，同时省去了“来回切换、反复扫描”的额外开销，而且在编程思路上不用太费脑力只需“记流水账式”的编程即可，但是它的缺点是当下只能干一件事，其它事情无法兼顾，做不到多任务并行处理。而“非阻塞”恰恰相反，它的有优点就是“阻塞”的缺点，它的缺点就是“阻塞”的优点，对于“非阻塞”本节暂时不多讲。在实际项目中，有时候“大 阻塞”中分支了 N 个“小 非阻塞”，也有时候“大 非阻塞”中分支了 N 个“小 阻塞”。能在“阻塞”与“非阻塞”之间运用自如者，谓之神。

“阻塞等待”是指单片机在某个死循环里（比如“while(1)”这类）一直不断循环地在等待某个标志变量的状态，如果这个标志变量满足条件才会跳出这个死循环然后才能干其它的事情，否则一直在死循环里死等，给人一种全神贯注心无旁骛的感觉，

“阻塞延时”是指单片机在产生“延时时间”的时候做不了别的事，延时多久它就要被“阻塞”多久，只有延时过后它才能解脱去干别的事。比如，在编程上，常用 for 循环产生 N 个空指令来达到产生“延时时间”的目的，这种编程方式就是最常见的“阻塞延时”。

【82.2　Delay 阻塞延时的一个例子。】

现在利用“Delay 阻塞延时”编写一个练习程序，让一个 LED 灯闪烁。例子如下：

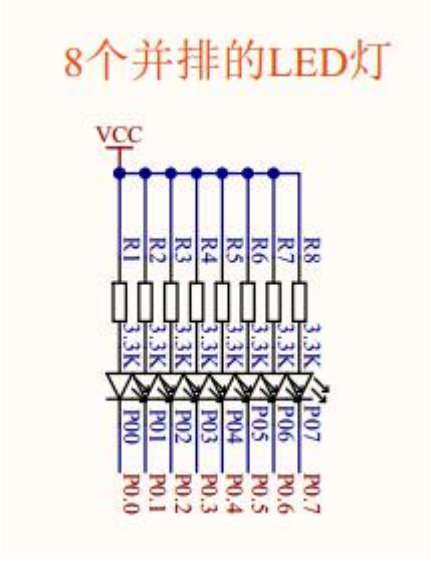

图 82.2.1　灌入式驱动 8 个 LED

```
#include "REG52.H"
void Delay(unsigned long u32DelayTime); //函数的声明
sbit P0_0=P0^0;  //利用 sbit 和符号“^”的组合，把变量名字 P0_0 与 P0.0 引脚关联起来
```

```
void Delay(unsigned long u32DelayTime) //产生“阻塞延时”的延时函数
{
   static unsigned long i; //函数在频繁调用时，加 static 可以省去一条额外的初始化语句的开销。
   for(i=0;i<u32DelayTime;i++);
}
void main()
{
   while(1)
   {
       //第（1）步
       P0_0=0;  //LED 灯亮。

       //第（2）步
      Delay(5000);  //这里就是阻塞延时，时间就越长，“亮”持续的时间就越长。

       //第（3）步
       P0_0=1;  //LED 灯灭。

       //第（4）步
      Delay(5000);  //这里就是阻塞延时，时间就越长，“灭”持续的时间就越长。

       //第（5）步：这里已经触碰到主循环 while(1)的“底线”，所以接着跳转到第（1）步继续循环
   }
}
```

【82.3　累加型和累减型的两种 Delay 函数，哪家强？】

上述 82.2 例子中，用到一个 Delay 函数，该函数内部的 for 循环用的是“累加型”的，比如：

```
   void Delay(unsigned long u32DelayTime)
   {
        static unsigned long i;       //“累加型”函数内部多开销了一个变量 i。
        for(i=0;i<u32DelayTime;i++);  //因为这里的“i++”是加法运算，所以称为“累加型”。
   }
```

现在在跟大家分享一种“累减型”的 Delay 函数，例子如下：

```
   void Delay(unsigned long u32DelayTime)
   {
                                          //“累减型”函数内部节省了一个变量 i。
        for(;u32DelayTime>0;u32DelayTime--);  //“u32DelayTime--”意味着“累减型”。
   }
```

仔细对比“累加型”和“累减型”，会发现在实现同样“阻塞延时”的功能下，因为“累减型”巧妙的

借用了函数入口的局部变量 u32DelayTime 来充当 for 循环的变量，而省去了一个 i 变量。因此，“累减型”比“累加型”强一点。

【82.4　Delay 函数让初学者容易犯的错误。】

初学者刚接触 Delay 函数，常常容易犯的错误就是忽略了 for 循环变量的类型，for 循环变量的类型决定了你能输入的数值范围，比如上面例子中用到的是 unsigned long 变量，因此可以最大输入 Delay(4294967295)。如果是 unsigned int 变量，最大可以输入 Delay(65535)。如果是 unsigned char 变量，最大可以输入 Delay(255)。

【82.5　Delay 内部的 for 循环嵌套可产生无穷长的时间。】

刚才讲到，如果用最大的变量类型 unsigned long ，最大的输入是 Delay(4294967295)，那么问题来，难道 Delay 函数的阻塞延时的时间有最大极限？其实不存在最大极限，理论上，你要多大的延时都可以，只需要在 Delay 函数内部用上 for 循环的嵌套，就可以产生“乘法级”的无穷长的时间，例子如下：

```
void Delay(unsigned long u32DelayTime)
{
    static unsigned long i;
    static unsigned long k;
    for(i=0;i<u32DelayTime;i++)
    {
       for(k=0;k<5000;k++); //内部嵌套的 for 循环，意味着乘法的关系 u32DelayTime 的 5000 倍!
    }
}
```

【82.6　“阻塞延时”与“非阻塞延时”的各自应用范围。】

“阻塞延时”一般应用在两个地方，一个是上电初始化进入主循环之前的延时，另一个是进入主循环之后，跟外部驱动芯片通信时候产生的时钟节拍小延时，而这个类延时一般是低于 1ms 的小延时。

“非阻塞延时”在项目中是被大量应用的，进入主循环之后，只要大于或等于 1ms 的延时，大多数都采样“非阻塞延时”，因为进入“任务框架级”的层面，只有“非阻塞延时”才能保证项目可以继续“多任务并行处理”。“非阻塞延时”的方式后续章节会讲到。

综上所述，1ms 是“阻塞延时”与“非阻塞延时”的一个分解线，1ms 这个时间不是绝对的，只是一个经验值。