第八十六节： 定时中断的“非阻塞”延时控制 LED 闪烁。

【86.1　定时中断应用的四大关键词。】

本节主要内容有四大个关键词：1ms，互斥量，volatile，switch。

（1）1ms。把定时中断设置为 1ms 中断一次，几乎是单片机界公认的“标配”。这个 1 ms 是系统时间的节拍来源，有了 1ms“标配”意识，你的程序在不同单片机平台上移植的时候会得心应手运用自如。

（2）互斥量。“主函数”与“定时中断函数”，本质上是两个独立进程在不断切换并行运行，两个进程之间不断切换，就会涉及到数据的安全保护，数据的安全保护主要是针对多字节的变量，比如 int 类型(2 个字节)，long 类型(4 个字节)。但是单字节的 char 变量不用额外保护，因为“字节”是变量中的最小单位（在不考虑“位”的情况下），这里的“最小单位不可分”就像“原子是最小单位不可分”一样，因此也有很多前辈把“互斥量”称为“原子锁”。为什么要用互斥量？因为，在多个线程同时访问同一个全局变量的时候，如果双方都是“读操作”，则不会出现问题，但是，如果双方都是“既有写操作也有读操作”的情况下，比如，我在主函数里正在修改（写操作）一个 unsigned int 类型的变量，unsigned int 类型的变量占用 2 个字节，在更改数据的时候至少需要 2 条指令，当我刚执行完第 1 条指令还没来得及执行第 2 指令的时候，突然来了一个定时中断，并且在定时中断函数里也对这个变量进行了修改（写操作）并且还进行了读取判断操作，这个瞬间就可能给程序带来了隐患。话说回来，互斥量到底有没有必要，其实还是有点争议的，我曾经为这个问题纠结过很久，毕竟，如果不用互斥量，这么微观的隐患到底存不存在，目前很难做一个“让故障重现”的实验去证明，最后，我是本着“宁可信其有不可信其无”的态度，把互斥量应用在了我的工作中。

（3）volatile。volatile 是一个前缀的修饰关键词，也是用来保护主函数与中断函数共用的全局变量的，只不过，volatile 是针对 C 编译器的，预防“C 编译器在优化代码的时候误伤一些重要的共享数据”，就像预防杀毒软软件用力过猛把一些合法软件当作病毒而误杀。加了 volatile 修饰的全局变量，就能提醒 C 编译器不要对这类特殊变量擅作主张去优化。

（4）switch。switch 是“非阻塞程序框架”的核心语句，在以 switch 为核心的框架下，进行不同步骤之间的程序跳转，是做大型裸机程序的常态。

【86.2　主函数与定时中断函数的程序框架。】

主函数与定时中断函数之间相互配合，主函数负责做什么，中断函数负责做什么，对于初学者来说可能是一头雾水，但是对于像我这种在单片机界深耕多年即将修炼成精的工程师来说，我心中是有很清晰的模板和套路的，这种模板和套路是经过多年沉淀下来的经验。比如，定时中断函数尽量放一些精简的计时器代码，一般不调用函数，但是“输入 IO 口的消抖动”(按键扫描)以及“蜂鸣器鸣叫”这两类特殊函数我是喜欢破例放在定时中断函数里调用的。定时中断如何产生时间，这个时间如何跟主函数关联起来，请看下面的框架代码：

```
volatile unsigned char vGu8TimeFlag=0;  //互斥量变量标志
volatile unsigned int vGu16TimeCnt=0;  //计时器变量

void main()
{
    vGu8TimeFlag=0;  //在“写操作”vGu16TimeCnt 全局变量之前，互斥量 vGu8TimeFlag 的“加锁”
    vGu16TimeCnt=1000;  //全局变量的赋值，就是“写操作”
```

```
    vGu8TimeFlag=1; //互斥量 vGu8TimeFlag 的“解锁”。同时也起到“启动计时器”的开关作用

    while(1)  //主循环
    {
         if(0==vGu16TimeCnt)  //时间变量为 0 则表示时间到了
         {
              ...在这里执行具体的功能代码
          }
   }
}


void T0_time() interrupt 1    //每 1ms 中断一次的定时中断函数
{
    if(1==vGu8TimeFlag&&vGu16TimeCnt>0) //判断 vGu8TimeFlag 是否等于 1，就是互斥量的判断。
    {
            vGu16TimeCnt--;  //“自减一”的操作
    }

}
```

分析：上述代码中，vGu8TimeFlag 是一箭双雕，既起到互斥量的作用，也起到了计数器 vGu16TimeCnt 开始计时的启动开关作用。

【86.3　练习例程。】

现在根据上述程序框架，编写一个 LED 灯闪烁的程序。

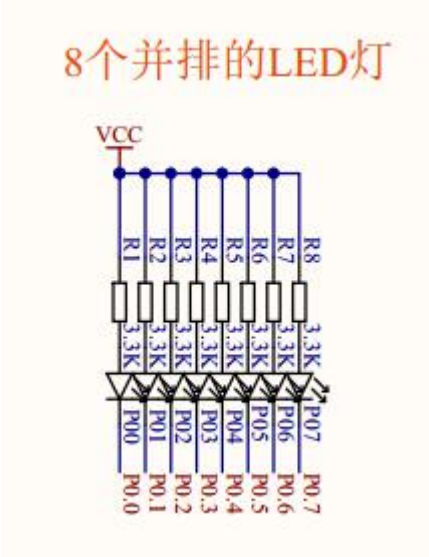

图 86.3.1　灌入式驱动 8 个 LED

```
#include "REG52.H"

#define  BLINK_TIME   500   //时间是500ms

sbit P0_0=P0^0;

volatile unsigned char vGu8TimeFlag=0;  //互斥量变量标志
volatile unsigned int vGu16TimeCnt=0;  //计时器变量

unsigned char Gu8Step=0;  //switch的切换步骤
void main()
{
    TMOD=0x01;  //设置定时器0为工作方式1
    TH0=0xfc;   //产生1ms中断的TH0初始值
    TL0=0x66;   //产生1ms中断的TL0初始值
    EA=1;       //开总中断
    ET0=1;      //允许定时0的中断
    TR0=1;      //启动定时0的中断


    while(1)  //主循环
    {
       switch(Gu8Step)
       {
           case 0:
               if(0==vGu16TimeCnt)  //时间到
               {
                    P0_0=0;   //LED灯亮
                    vGu8TimeFlag=0;  //互斥量“加锁”
                    vGu16TimeCnt=BLINK_TIME;  //计时器的写操作。设定计时的长度
                    vGu8TimeFlag=1;  //互斥量“解锁”，同时蕴含了计时器“启动”的动作

                    Gu8Step=1;  //切换到case 1这个步骤
                }
               break;

           case 1:

              if(0==vGu16TimeCnt)  //时间到
               {
                    P0_0=1;   //LED灯灭。
                    vGu8TimeFlag=0;  //互斥量“加锁”
                    vGu16TimeCnt=BLINK_TIME;  //计时器的写操作。设定计时的长度
                    vGu8TimeFlag=1;  //互斥量“解锁”，同时蕴含了计时器“启动”的动作
```

```
                    Gu8Step=0;  //切换到 case 0 这个步骤，依次循环
                }
                break;
        }
    }
}


void T0_time() interrupt 1    //定时器 0 的中断函数，每 1ms 单片机自动执行一次此函数
{
    if(1==vGu8TimeFlag&&vGu16TimeCnt>0) //判断 vGu8TimeFlag 是否等于 1，就是互斥量的判断
    {
        vGu16TimeCnt--;  //“自减一”的操作
    }

    TH0=0xfc;   //重装初值，不能忘
    TL0=0x66;   //重装初值，不能忘
}
```

【86.4　解决闪烁出现不规则“非对称感”现象的方法。】

上述例子，实验现象应该是 LED 闪烁很有规则的每 1s 闪烁一次，但是也有一部分初学者可能会遇到闪烁出现不规则“非对称感”的现象，这个问题的解决办法如下：在 keil2 的 project 下拉菜单下，选择 Options for Target 选项，弹出的窗口中，切换到 Target 选项，在 Memory Model 选项中选择 small:variables in Data。

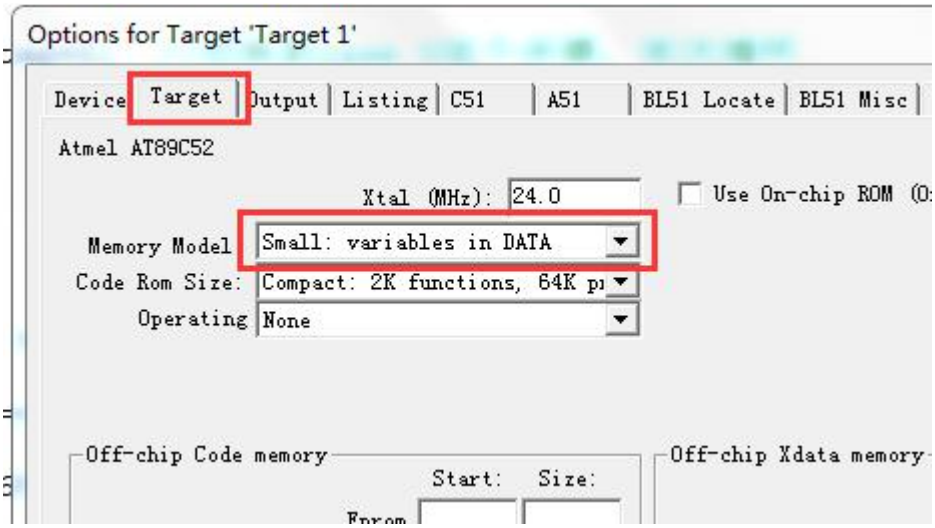

图 86.4.1　设置窗口