

## 第八十九节： 跑马灯的三种境界。

### 【89.1 跑马灯的三种境界。】

跑马灯也称为流水灯，排列的几个 LED 依次循环的点亮和熄灭，给人“跑动起来”的感觉，故称为“跑马灯”。实现跑马灯的效果，编程上有三种思路，分别代表了跑马灯的三种境界，分别是：移位阻塞，移位非阻塞，状态切换非阻塞。

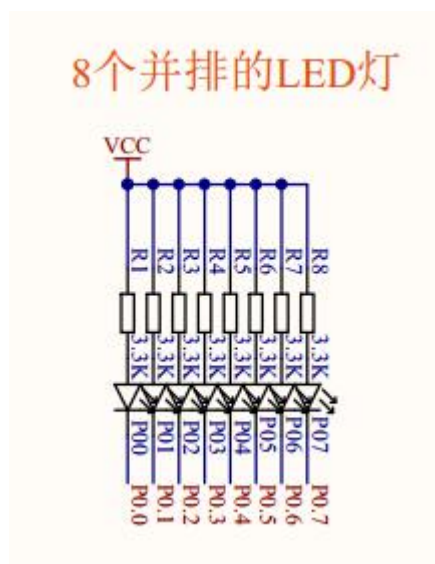


图 89.1.1 灌入式驱动 8 个 LED

本节用的是 8 个 LED 灯依次挨个熄灭点亮，如上图所示。

### 【89.2 移位阻塞。】

移位阻塞，“移位”用的是 C 语言的左移或者右移语句，“阻塞”用的是 delay 延时。代码如下：

```
#include "REG52.H"

void TO_time();
void SystemInitial(void) ;
void Delay(unsigned long u32DelayTime) ;
void PeripheralInitial(void) ;
void LedTask(void);

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
```

```

    {
        LedTask();
    }
}

void T0_time() interrupt 1
{
    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)
{
}

//跑马灯的任务程序
void LedTask(void)
{
    static unsigned char Su8Data=0x01; //加 static 修饰的局部变量，每次进来都会保留上一次值。
    static unsigned char Su8Cnt=0;      //加 static 修饰的局部变量，每次进来都会保留上一次值。

    P0=Su8Data; //Su8Data 的 8 个位代表 8 个 LED 的状态，0 为点亮，1 为熄灭。
    Delay(10000); //阻塞延时
    Su8Data=Su8Data<<1; //左移一位
    Su8Cnt++; //计数器累加 1
    if(Su8Cnt>=8) //移位大于等于 8 次后，重新赋初值

```

```

    {
        Su8Cnt=0;
        Su8Data=0x01; //重新赋初值，继续下一次循环移动
    }
}

```

分析总结：这是第 1 种境界的跑马灯，这种思路虽然实现了跑马灯的效果，但是因为“阻塞延时”，整个程序显得僵硬机械，缺乏多任务并行的框架。

### 【89.3 移位非阻塞。】

移位非阻塞，“移位”用的是 C 语言的左移或者右移语句，“非阻塞”用的是定时中断衍生出来的软件定时器。代码如下：

```

#include "REG52.H"

void TO_time();
void SystemInitial(void) ;
void Delay(unsigned long u32DelayTime) ;
void PeripheralInitial(void) ;
void LedTask(void);

#define BLINK_TIME_1 1000

volatile unsigned char vGu8TimeFlag_1=0;
volatile unsigned int vGu16TimeCnt_1=0;

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
        LedTask();
    }
}

void TO_time() interrupt 1
{
    if(1==vGu8TimeFlag_1&&vGu16TimeCnt_1>0) //软件定时器
    {
        vGu16TimeCnt_1--;
    }
}

```

```

    }

    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)
{
}

//跑马灯的任务程序
void LedTask(void)
{
    static unsigned char Su8Data=0x01; //加 static 修饰的局部变量，每次进来都会保留上一次值。
    static unsigned char Su8Cnt=0;      //加 static 修饰的局部变量，每次进来都会保留上一次值。

    if(0==vGu16TimeCnt_1)      //时间到
    {
        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1; //重装定时的时间
        vGu8TimeFlag_1=1;

        P0=Su8Data; //Su8Data 的 8 个位代表 8 个 LED 的状态，0 为点亮，1 为熄灭。
        Su8Data=Su8Data<<1; //左移一位
        Su8Cnt++; //计数器累加 1
        if(Su8Cnt>=8) //移位大于等于 8 次后，重新赋初值
        {

```

```

        Su8Cnt=0;
        Su8Data=0x01; //重新赋初值，继续下一次循环移动
    }
}
}

```

分析总结：这是第 2 种境界的跑马灯，这种思路虽然实现了跑马灯的效果，也用到了多任务并行处理的基本元素“软件定时器”，但是因为还停留在“移位”语句的阶段，此时的程序并没有超越跑马灯本身，跑马灯还是跑马灯，处于“看山还是山”的境界。

#### 【89.4 状态切换非阻塞。】

状态切换非阻塞，“状态切换”用的是 switch 语句中根据特定条件进行步骤切换，“非阻塞”用的是定时中断衍生出来的软件定时器。代码如下：

```

#include "REG52.H"

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

#define BLINK_TIME_1 1000

sbit P0_0=P0^0;
sbit P0_1=P0^1;
sbit P0_2=P0^2;
sbit P0_3=P0^3;
sbit P0_4=P0^4;
sbit P0_5=P0^5;
sbit P0_6=P0^6;
sbit P0_7=P0^7;

volatile unsigned char vGu8TimeFlag_1=0;
volatile unsigned int vGu16TimeCnt_1=0;

void main()
{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)

```

```

    {
        LedTask();
    }
}

void T0_time() interrupt 1
{
    if(1==vGu8TimeFlag_1&&vGu16TimeCnt_1>0) //软件定时器
    {
        vGu16TimeCnt_1--;
    }

    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)
{
}

//跑马灯的任务程序
void LedTask(void)
{
    static unsigned char Su8Step=0;    //加 static 修饰的局部变量，每次进来都会保留上一次值。

    switch(Su8Step)
    {

```

```

case 0:
    if(0==vGu16TimeCnt_1)  //时间到
    {

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;  //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=1;  //第 0 个灯熄灭
        P0_1=0;
        P0_2=0;
        P0_3=0;
        P0_4=0;
        P0_5=0;
        P0_6=0;
        P0_7=0;

        Su8Step=1;  //切换到下一个步骤，精髓语句！
    }
    break;

case 1:
    if(0==vGu16TimeCnt_1)  //时间到
    {

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;  //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=1;  //第 1 个灯熄灭
        P0_2=0;
        P0_3=0;
        P0_4=0;
        P0_5=0;
        P0_6=0;
        P0_7=0;

        Su8Step=2;  //切换到下一个步骤，精髓语句！
    }
    break;

case 2:
    if(0==vGu16TimeCnt_1)  //时间到

```

```

{

    vGu8TimeFlag_1=0;
    vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间
    vGu8TimeFlag_1=1;

    P0_0=0;
    P0_1=0;
    P0_2=1;    //第 2 个灯熄灭
    P0_3=0;
    P0_4=0;
    P0_5=0;
    P0_6=0;
    P0_7=0;

    Su8Step=3;    //切换到下一个步骤，精髓语句！
}
break;

case 3:
    if(0==vGu16TimeCnt_1)    //时间到
    {

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=0;
        P0_2=0;
        P0_3=1;    //第 3 个灯熄灭
        P0_4=0;
        P0_5=0;
        P0_6=0;
        P0_7=0;

        Su8Step=4;    //切换到下一个步骤，精髓语句！
    }
    break;

case 4:
    if(0==vGu16TimeCnt_1)    //时间到
    {

```



```

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=0;
        P0_2=0;
        P0_3=0;
        P0_4=1;    //第 4 个灯熄灭
        P0_5=0;
        P0_6=0;
        P0_7=0;

        Su8Step=5;    //切换到下一个步骤，精髓语句！
    }
    break;

case 5:
    if(0==vGu16TimeCnt_1)    //时间到
    {

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=0;
        P0_2=0;
        P0_3=0;
        P0_4=0;
        P0_5=1;    //第 5 个灯熄灭
        P0_6=0;
        P0_7=0;

        Su8Step=6;    //切换到下一个步骤，精髓语句！
    }
    break;

case 6:
    if(0==vGu16TimeCnt_1)    //时间到
    {

        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间

```

```

        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=0;
        P0_2=0;
        P0_3=0;
        P0_4=0;
        P0_5=0;
        P0_6=1;    //第 6 个灯熄灭
        P0_7=0;

        Su8Step=7;    //切换到下一个步骤，精髓语句！
    }
    break;

case 7:

    if(0==vGu16TimeCnt_1)    //时间到
    {
        vGu8TimeFlag_1=0;
        vGu16TimeCnt_1=BLINK_TIME_1;    //重装定时的时间
        vGu8TimeFlag_1=1;

        P0_0=0;
        P0_1=0;
        P0_2=0;
        P0_3=0;
        P0_4=0;
        P0_5=0;
        P0_6=0;
        P0_7=1;    //第 7 个灯熄灭

        Su8Step=0;    //返回到第 0 个步骤重新开始往下走，精髓语句！
    }
    break;
}
}

```

分析总结：这是第 3 种境界的跑马灯，很多初学者咋看此程序，表示不理解，人家一条赋值语句就解决 8 个 LED 一次性显示的问题，你非要拆分成 8 条按位赋值的语句，人家只用一个判断就实现了 LED 灯移动显示的功能，你非要整出 8 个步骤的切换，况且，整个程序的代码量明显增加了很多，这个程序好在哪？其实，我这么做是用心良苦呀。这个程序的代码量虽然增多了，但是仔细一看，并没有影响运行的效率。之所以把 8 个 LED 灯拆分成一个一个的 LED 灯单独赋值显示，是因为，在我眼里，这个 8 个 LED 灯代表的不仅仅是 LED 灯，而是 8 个输出信号！这 8 个输出信号未来驱动的可能是不同的继电器，气缸，电机，大炮，导弹，以及

它们的各种千变万化的组合逻辑，拆分之后程序框架就有了无限可能的扩展性。之所以整出 8 个步骤的切换，也是同样的道理，为了增加程序框架无限可能的扩展性。这个程序虽然表面看起来繁琐，但是仔细一看它是“多而不乱”，非常富有“队形感”。因此可以这么说，这个看似繁琐的跑马灯程序，其实背后蕴藏了编程界的大智慧，它已经突破了“看山还是山”的境界。