第 5 章

定时器应用案例

基于 Proteus 硬件仿真工具和 Keil 程序设计软件，本章设计了微控制器中定时/计数器技术原理及应用相关的案例，包括计数器功能应用、定时器控制流水灯、定时器控制交通灯，每个案例提供了详尽的软硬件设计，包括仿真电路和完整的参考程序。

5.1 计数器功能的应用

5.1.1 案例概述

通过 8051 系列微控制器控制实现如下功能：

(1) 从外部输入 2 个计数脉冲到定时/计数器引脚 T0 后产生中断，在 T0 计数中断服务程序中，控制接在 P1 口的 8 个 LED 依次点亮 200ms，返回主程序；

(2) 从外部输入 5 个计数脉冲到定时/计数器引脚 T1 后产生中断，在 T1 计数中断服务程序中，控制接在 P2.1 引脚的蜂鸣器发出报警声 5 次，返回主程序；

(3) 定时/计数器 T0 和 T1 总的中断次数通过 P0 口输出到七段 BCD 数码管显示。

5.1.2 要求

(1) 了解中断的步骤及定时/计数器的工作方式；

(2) 关闭和启动定时器的条件和关闭和打开中断的条件；

(3) 计数初值的定义。

5.1.3 知识点

定时/计数器硬件原理、计数器功能应用编程。

5.1.4 电路原理图

案例电路原理图如图 5-1 所示。

在电路图中，利用虚拟元件脉冲信号发生器产生外部脉冲信号，然后送入定时/计数器的输入引脚 P3.4、P3.5。因为定时/计数器 T0、T1 设置为计数功能，用来检测 P3.4、P3.5 引脚接收的脉冲信号个数，当达到设置的计数值的个数后，产生计数器溢出中断，然后暂停主程序，转而执行计数器中断子程序。

案例通过 P0 口驱动 2 个 BCD 七段数码管显示计数器总中断次数。BCD 七段数码管

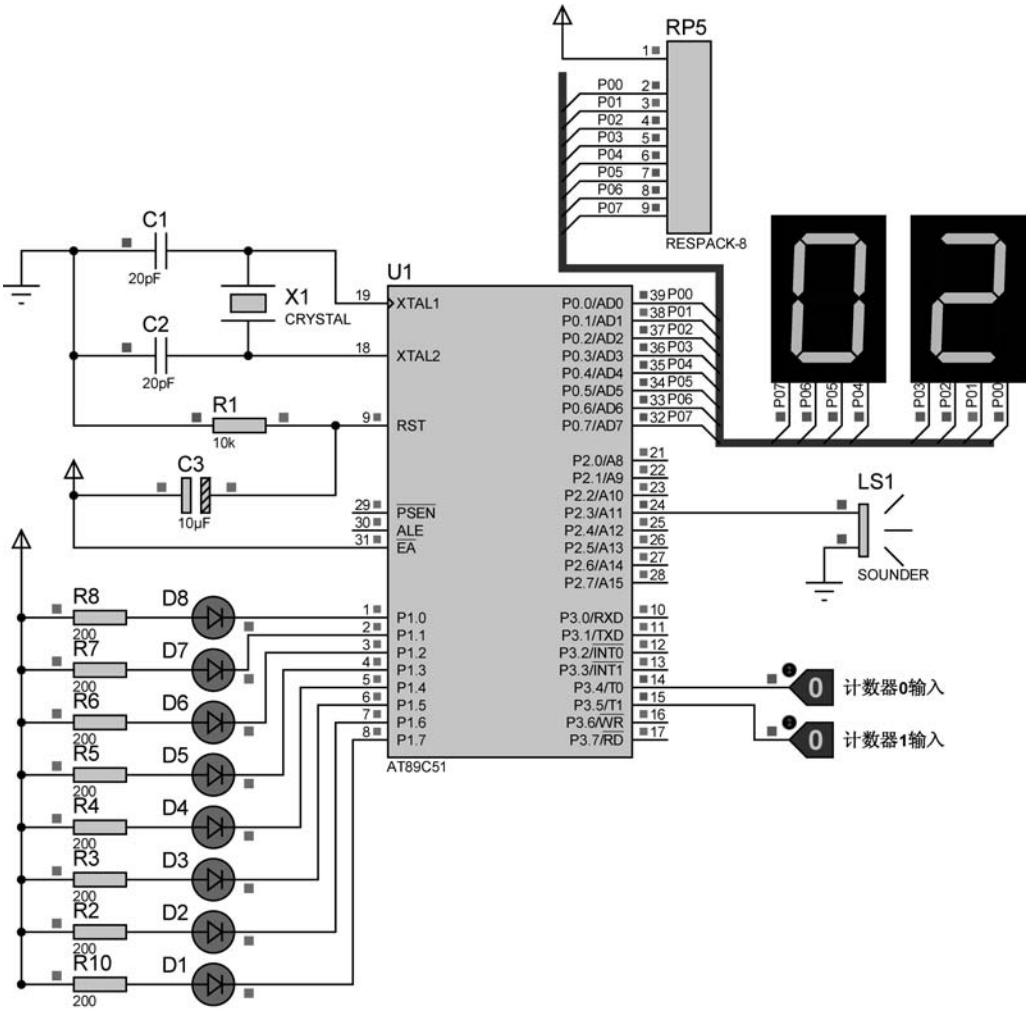

图 5-1　计数器应用电路图

的输入是 4 位二进制 BCD 码。BCD 七段数码管按 8421BCD 编码来点亮数码管显示数字，例如，当输入 8421 码 DCBA＝0000 时，数码管就显示 0；当 DCBA＝0100 时，应显示 4；DCBA＝0011 时显示 3；DCBA＝1111 时显示 F，以此类推。另外，微控制器通过 P2.3 引脚输出高电平驱动蜂鸣器发音。

5.1.5　案例应用程序

```
#include <intrins.h>
#include <REG51.H>
#define TRUE 1
#define uchar unsigned char
uchar t3 = 0;                                    //总中断次数
unsigned char ucTimes;
sbit   BEEP = P2^3;
void Play(unsigned char t);
```

```
void time(unsigned int ucMs);        //延时单位: ms

void main(void)
{
    TMOD = 0x55;                //设置定时器 0、1 为外部脉冲输入计数器工作,方式 1,16 位计数器
    TH0 = 0xFF; TL0 = 0xFE;          //设置计时器 0 的初值为 FFFEH,即 2 个计数脉冲产生中断
    TH1 = 0xFF; TL1 = 0xFB;          //设置计时器 1 的初值为 FFFBH,即 5 个计数脉冲产生中断
    TR0 = 1;                         //开启定时器 0
    TR1 = 1;                         //开启定时器 1
    IE = 0x8a;                       //打开定时器 0、1 中断
    P0 = t3;                         //总中断次数送 P1 口显示
    while(1){}                       //等待定时器 0、1 的中断
}

//定时器 0 的中断服务程序
void timer0(void) interrupt 1
{
    EA = 0;                                  //关总中断
    TR0 = 0;                                 //停止计时
    t3++;                                    //总中断次数加 1
    P0 = t3;                                 //总中断次数送 P0 口显示
    for(ucTimes = 0;ucTimes < = 8;ucTimes++)       //循环点亮 P1 口的 8 个灯
    {
        P1 = 0xff - (0x01 << ucTimes);               //亮灯需低电平驱动
        time(700);
    }
    TH0 = 0xFF; TL0 = 0xFE;
    TR0 = 1;
    EA = 1;
}

void timer1(void) interrupt 3
{
    EA = 0;                                  //总中断
    TR1 = 0;                                 //停止计时
    t3++;                                    //总中断次数加 1
    P0 = t3;                                 //总中断次数送 P0 口显示

    for(ucTimes = 0;ucTimes < 5;ucTimes++)
    {
        Play(2);
        time(200);
    }
    TH1 = 0xFF; TL1 = 0xFB;
    TR1 = 1;                                 //开启定时器
    EA = 1;                                  //开总中断
}

void DelayMS(unsigned int  x)
{
    uchar t;
```

```
    while(x--)
    {
        for(t=0;t<120;t++);
    }
}

void Play(unsigned char t)
{
    uchar i;
    for(i=0;i<100;i++)
    {
      BEEP=~BEEP;
      DelayMS(t);
    }
    BEEP=0;
}

void delay_5us(void)                    //延时 5μs
{
    _nop_();
    _nop_();
}

void delay_50us(void)                   //延时 50μs
{
    unsigned char i;
    for(i=0;i<4;i++)
    {
    delay_5us();
    }
}

void delay_100us(void)                  //延时 100μs
{
    delay_50us();
    delay_50us();
}

void time(unsigned int ucMs)            //延时单位: ms
{
    unsigned char j;
    while(ucMs>0){
    for(j=0;j<10;j++) delay_100us();
    ucMs--;
    }
}
```

5.1.6 案例分析

案例代码中主要有 3 个函数：主函数 main()、计数器 0 中断函数 timer0()和计数器 1

中断函数 timer1(),3 个函数是并行关系。主函数和两个计数器中断函数实现不同的功能，可以直观地理解 3 个函数的并行性以及中断函数 timer0()和 timer1()执行的随机性。

通过本案例，可以将计数器的中断工作方式、计数初值、计数溢出中断标识位、中断号与对应的 C51 语言程序控制语句关联起来，更好地理解定时计数器硬件结构和工作原理。

另外，在计数器 1 中断函数 timer1()中要实现 P2.3 引脚驱动蜂鸣器发出 5 次报警声。通过改变 P2.3 引脚输出高低电平信号的间隔时间控制报警声的频率，具体实现代码如下：

```
void Play(unsigned char t)
{
    uchar i;
    for(i = 0;i < 100;i++)
    {
        BEEP = ～BEEP;
        DelayMS(t);
    }
    BEEP = 0;
```

通过改变 P2.3 引脚输出高低电平信号的次数控制报警声的长短，具体实现代码如下：

```
for(ucTimes = 0;ucTimes < 5;ucTimes++)
     {
           Play(2);
           time(200);
     }
}
```

5.2　定时器控制流水灯

5.2.1　案例概述

通过 8051 系列微控制器控制实现如下功能：使用定时器 T1 的中断来控制 P1 连接的 8 个 LED 灯的流水闪烁，即依次轮流控制每个灯点亮 50ms，实现流水闪烁效果。

5.2.2　要求

(1) 学习使用微控制器定时器的工作原理、应用和编程方法；

(2) 熟悉微控制器中定时器中断处理程序编写。

5.2.3　知识点

定时器硬件结构、工作方式、定时器中断方式的应用。

5.2.4　电路原理图

案例控制电路如图 5-2 所示，8 个 LED 灯连接在微控制器的 P1 口的 8 个引脚上，限流作用的电阻阻值设定为 200Ω，灯采用灌电流连接方式连接。

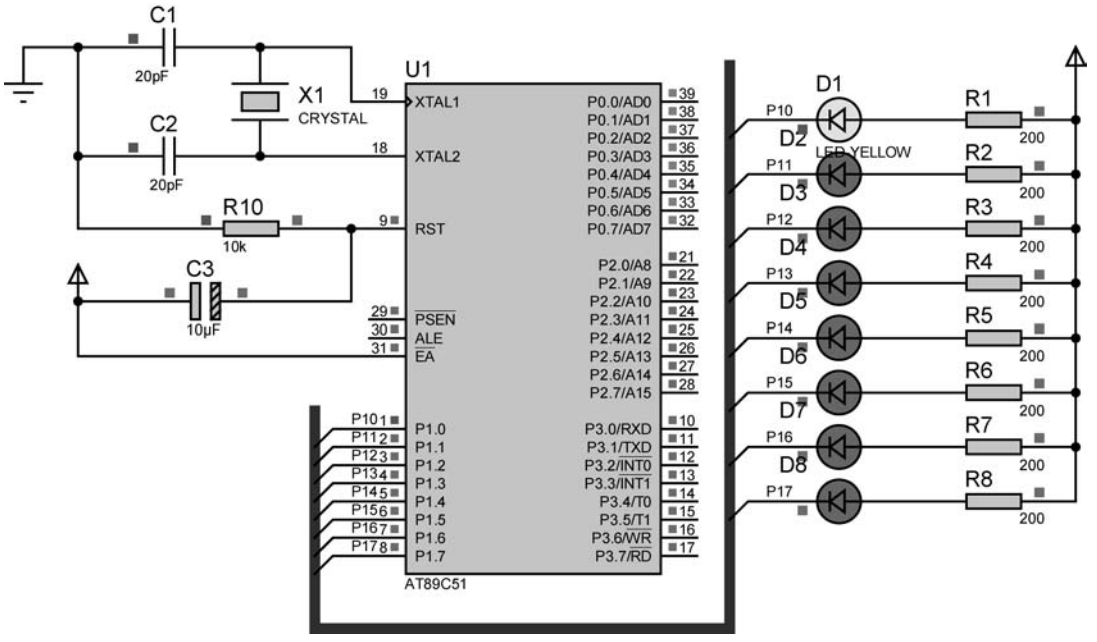

图 5-2 定时器控制流水灯电路图

5.2.5 案例应用程序

```
#include <REG51.H>
unsigned char kkk;
unsigned char Count = 0;
void main(void)
{
    TMOD = 0x10;                          //定时器 T0 的工作方式 1
    TH1 = (65536 - 60536)/256;            //定时器 T0 的高 8 位的初值
    TL1 = (65536 - 60536) % 256;          //定时器 T0 的低 8 位的初值
    TR1 = 1;                              //启动定时器 T0
    kkk = 0xfe;
    P1 = kkk;                             //流水灯初始状态控制
    EA = 1;                               //开总中断
    ET1 = 1;                              //定时器 T0 中断允许
    while(1) ;                            //无限循环,等待定时器溢出中断
}

void Time1(void) interrupt 3 using 0      //定时器 T0 的中断服务程序
{
     TH1 = (65536 - 60536)/256;           //重新设置定时器 T0 的高 8 位的初值
     TL1 = (65536 - 60536) % 256;         //重新设置定时器 T0 的低 8 位的初值

     if(++Count!= 10) return;             // 50ms(10×5ms)延时切换
     Count = 0;
     kkk = kkk << 1;                      //数据左移 1 位
     kkk = kkk|1;                         //数据末位置 1
```

```
    if(kkk == 0xff)kkk = 0xfe;              //设置流水灯的控制数据初始值
    P1 = kkk;                               //更新流水灯状态控制数据
}
```

5.2.6　案例分析

案例中，需要控制每个 LED 灯点亮 50ms，定时由定时器硬件实现。程序中设置定时器每次定时 5ms，为了简化初值计算，其初值设置如下：

```
TH1 = (65536 - 60536)/256;              //重新设置定时器 T0 的高 8 位的初值
TL1 = (65536 - 60536) % 256;            //重新设置定时器 T0 的低 8 位的初值
```

50ms 的时间实现通过定时器中断函数 Time1()中的两条语句实现，即

```
if(++Count!= 10) return;                //50ms(10 × 5ms)延时切换
Count = 0;
```

每当定时器硬件 5ms 定时到就产生溢出中断，暂停主程序 main()函数后，CPU 执行中断函数 Time1()，通过 if(++Count!=10) 判断 10 次 5ms 延时是否完成。具体方法是：++Count 后变量 Count 的值不等于 10，则 return 返回到调用此语句所在函数的地方，即返回到调用定时器 1 中断函数 Timer1()的地方，就是主程序的 while(1)处，即回到 main()函数继续等待下一次 50ms 定时中断发生。这个过程重复 10 次后，即延时 10 次 5ms 之后(此时，P1 口的引脚状态保持 50ms，即流水灯状态保持了 50ms)，不再 return 返回到主程序的 while(1)处，而是顺序执行中断函数后续语句：

```
Count = 0;
kkk = kkk << 1;                         //数据左移 1 位
kkk = kkk|1;                            //数据末位置 1
if(kkk == 0xff)kkk = 0xfe;              //设置流水灯的控制数据初始值
P1 = kkk;                               //更新流水灯状态控制数据
```

通过这段代码，首先 Count=0，为下次 50ms 定时做准备。然后更新 P1 口的引脚状态值，从而更新流水灯的状态。

5.3　定时器控制交通灯

5.3.1　案例概述

通过 8051 系列微控制器控制十字路口模拟交通灯的变化，分别实现如下功能：

(1) 东西向绿灯与南北向红灯亮 4s；

(2) 东西向黄灯开始闪烁 6 次，绿灯关闭；

(3) 东西向红灯与南北向绿灯亮 4s；

(4) 南北向黄灯开始闪烁 6 次，绿灯关闭；

(5) 如此往复。

其中定时功能采用微控制器的定时器硬件模块实现。

5.3.2 要求

(1) 掌握微控制器定时器的原理、工作方式和应用编程;

(2) 掌握通过定时器硬件实现模拟交通灯的延时的时间控制方法。

5.3.3 知识点

定时器硬件原理、定时器功能的应用编程。

5.3.4 电路原理图

定时器控制十字路口模拟交通灯电路如图 5-3 所示。

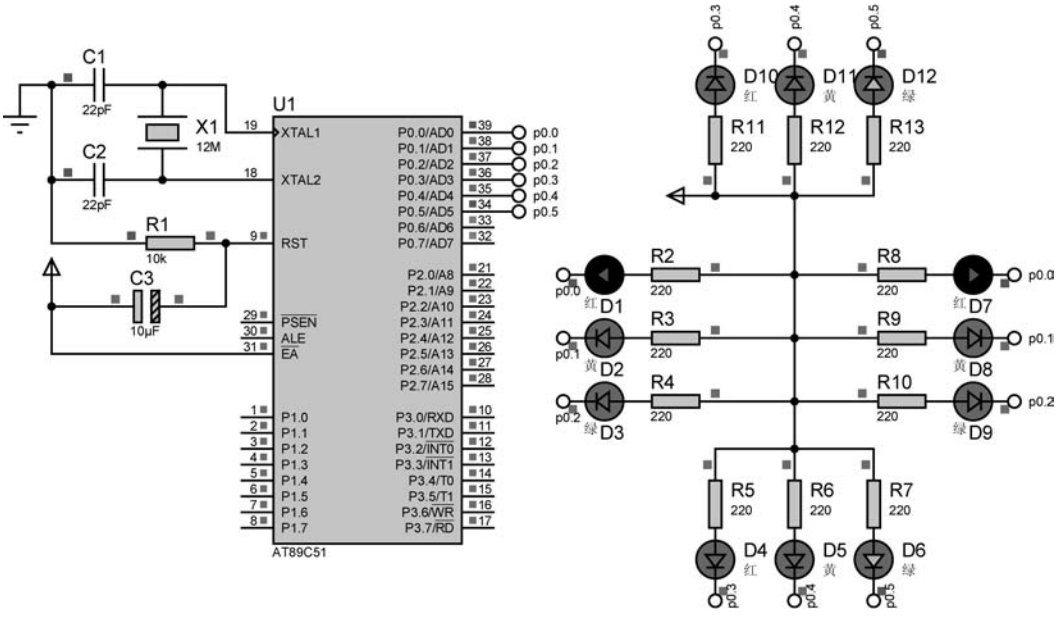

图 5-3 交通灯电路

案例中,东西向的红、黄、绿 3 个灯分别由微控制器的 P0.0、P0.1、P0.2 3 个引脚控制。南北向的红、黄、绿 3 个灯分别由微控制器的 P0.3、P0.4、P0.5 3 个引脚控制。另外,所有 LED 灯都是通过灌电流连接方式连接的。

5.3.5 案例应用程序

```
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
sbit R_A = P0^0;                    //东西向指示灯
sbit Y_A = P0^1;
sbit G_A = P0^2;
sbit R_B = P0^3;                    //南北向指示灯
sbit Y_B = P0^4;
```

```
sbit G_B = P0^5;
uchar TC = 0,FC = 0,K = 1;              //定义延时倍数、闪烁次数、操作类型变量

//定时器 1 中断函数
void Timer1()  interrupt  3
{
    TL1 = -50000/256;
    TH1 = -50000 % 256;                 //50ms 定时的初值
    switch(K)
    {
        case 1: //东西向绿灯与南北向红灯亮 4s
                R_A = 1;Y_A = 1;G_A = 0;                    //P0^2 = 0,灯亮
                R_B = 0;Y_B = 1;G_B = 1;
                if(++TC!= 80) return;                       //4s(80×50ms)切换
                TC = 0;
                K = 2;
                break;
        case 2: //东西向黄灯开始闪烁,绿灯关闭
                if(++TC!= 10) return;                       //延时 10×50ms
                TC = 0;
                Y_A = ~Y_A;G_A = 1;
                if(++FC!= 12) return;                       //闪烁 6 次
                FC = 0;
                K = 3;
                break;

        case 3: //东西向红灯与南北向绿灯亮 4s
                R_A = 0;Y_A = 1;G_A = 1;
                R_B = 1;Y_B = 1;G_B = 0;
                if(++TC!= 50) return;                       //4s(80×50ms)切换
                TC = 0;
                K = 4;
                break;

        case 4: //南北向黄灯开始闪烁,绿灯关闭
                if(++TC!= 10) return;                       //延时 10×50ms
                TC = 0;
                Y_B = ~Y_B;G_B = 1;
                if(++FC!= 12) return;                       //闪烁 6 次
                FC = 0;
                K = 1;
                break;
    }
}

//主程序
void main()
{
    TMOD = 0x10;                                            //T1 方式 1
    TL1 = -50000/256;                                       //初值设置,方式 1 最多 56ms
    TH1 = -50000 % 256;
```

```
    IE = 0x88;
    TR1 = 1;
    while(1);
}
```

5.3.6 案例分析

案例代码中主要有两个函数：主函数 main()和定时器 1 的中断函数 timer1()。主函数完成定时器 1 的初始化工作，交通灯亮灭的 4 种效果控制代码由函数 timer1()完成。

案例中，为了简化定时器初值的十六进制转换计算，初值高 8 位采取取商计算，即 TH0＝(65536－50000)/256；初值低 8 位采取取余计算，即 TL0＝(65536－50000)％256。因此，定时器初值为：

```
TL1 = -50000/256;
TH1 = -50000 % 256;
```

交通灯中的黄灯闪烁 6 次的实现在 switch 语句的 case 2 分支中，具体代码如下：

```
if(++FC!= 12) return;              //闪烁 6 次
FC = 0;
Y_A = ~Y_A; G_A = 1;
```

每次执行语句＋＋FC 后判断变量 FC 的值。如果 FC 的值不等于 12，则返回到调用此语句所在函数的地方，也就是返回到调用定时器 1 中断函数 timer1()的地方，即主程序的 while(1)处。此时黄灯状态为灭。

当定时器定时时间达到 50ms 后就产生溢出，随后程序重新进入中断函数 timer1()，再次执行 case 2 分支，重新执行如下代码：

```
if(++TC!= 10) return;              //延时 10×50ms
TC = 0;
Y_A = ~Y_A;G_A = 1;
```

当定时器硬件定时 50ms 溢出过程达到 10 次，即延时 10 个 50ms 后，才继续执行代码 Y_A＝～Y_A 和 G_A＝1，实现 Y_A 取反，此时黄灯状态为亮。从而实现黄灯亮灭一次，实现闪烁的效果。

直到上述过程执行 12 次后，FC＝12，不再 return 返回到主程序的 while(1)处，而是顺序执行后边语句(FC＝0；K＝3；)，为下次黄灯闪烁 6 次做准备，并切换交通灯的操作类型为 3。另外，switch 分支结构中的 case 4 分支实现的原理与 case 2 原理相同。

交通灯中的红灯或绿灯亮 4s 的实现在 switch 语句的 case 1 分支中，具体代码如下：

```
if(++TC!= 80) return;
TC = 0;
```

每次执行语句＋＋TC 后判断变量 TC 的值。如果 TC 的值不等于 80，则返回到调用此语句所在函数的地方，也就是返回到调用定时器 1 中断函数 timer1()的地方，即主程序的 while(1)处。

当定时器定时时间达到50ms后就产生溢出，随后程序重新进入中断函数timer1()，再次执行case 1分支，重新执行如下代码：

```
if(++TC!= 80) return;
TC = 0;
```

当定时器硬件定时50ms溢出过程达到80次，即延时80次50ms后，不再return返回到主程序的while(1)处，而是顺序执行TC＝0，为下次延时4s做准备。此时已经延时80×50ms，实现绿灯或红灯持续亮4s的效果。另外，switch分支结构中的case 3分支实现的原理与case 1原理相同。