例 8.1 2N 分频系统设计一

1） 硬件电路

2N 分频系统要实现的功能是对 T0 脚输入的方波信号进行偶数次的分频，以获得频率低于 T0 输入的方波信号。

本设计的硬件电路非常简单，将实验板上的 250Hz 的方波信号输出与 ATmega16 的 T0 脚连接，作为 T/C0 计数器的外部输入。另外将 PA0 作为分频后的脉冲输出脚，用 PA0 控制一个 LED 的显示，通过 LED 的亮暗显示可以简单的观察方波的频率。当然最好的方式是使用示波器观察 PA0 的输出。

2） 软件设计

首先考虑使用 T/C0 的普通模式（WGM0[1:0]=0），采用 T0 上升沿触发（CS0[2:0]=111），并设置 TCNT0 的初值为 0xFF。当 T0 引脚输入电平出现一个上跳变时，T/C0 的 TCNT0 回到 0x00，并产生溢出中断（参考图 8-10），在溢出中断服务中重新设置 TCNT0 为 0xFF，并改变 PA0 口的输出电平（取反输出）。在 T0 引脚输入电平出现第二个上跳变时，又会产生中断，在中断服务程序中再次改变 PA0 的输出，这样在 PA0 上就得到 T0 的 2 分频输出信号。同理，如果将 TCNT0 的初值设置为 0xFE，则在 PA0 上得到 T0 的 4 分频输出信号，0xFD→6 分频，0xFC—>8 分频……，而当 TCNT0 的初值设置为 0x00 时，可实现最大 512 分频的输出。

下面，我们给出在 PA0 上输出 1Hz 的方波(LED 亮 0.5s，暗 0.5s)的设计和程序。由于 T0 输入的频率为 250Hz，所以分频系数为 250，因此 TCNT0 的初值=255-124（0x83），即 T/C0 计数 125 次时 PA0 的电平改变一次。

再次建议读者使用 CVAVR 中的程序生成向导功能来帮助你建立整个程序的框架，以及芯片的初始化部分的语句，可以省掉你过多的查看器件手册和考虑寄存器的设置值等。

图 8-14 是在 CVAVR 的程序生成向导中设置 T/C0 的对话窗口。选择 T/C0 的计数时钟源为 T0 的上升沿，工作方式为普通模式，允许溢出中断，TCNT0 的初值为 0x83。

利用 CVAVR 的程序生成向导，在它的帮助下生成一个程序框架后，然后再加入自己的程序和进行必要的修改。

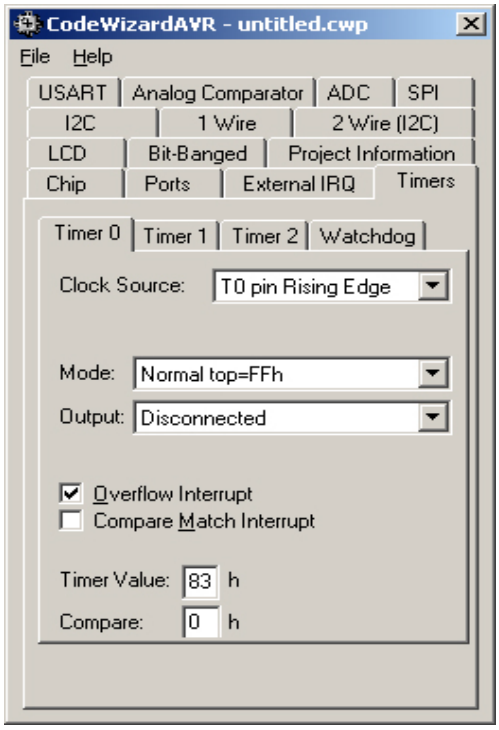

图 8-14 在程序生成向导中设置 T/C0

```
/*****************************************
File name          : Demo_8_1.c
Chip type          : ATmega16
Program type       : Application
Clock frequency    : 4.000000 MHz
Memory model       : Small
External SRAM size : 0
Data Stack size    : 256
*****************************************/

#include <mega16.h>

// Timer 0 溢出中断服务
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
    TCNT0=0x83;                 // 重新设置 TCNT0 的初值
    PORTA.0 = ~PORTA.0;         // PA0 取反输出
}

void main(void)
{
    PORTA=0x01;
    DDRA=0x01;                  // 设置 PA0 输出方式

    PORTB=0x01;
    DDRB=0x00;                  // 设置 PB0(T0)为输入方式

    // T/C0 初始化
    TCCR0=0x07;                 // T/C0 工作于普通模式，T0 上升沿触发
    TCNT0=0x83;
    OCR0=0x00;

    TIMSK=0x01;                 // 允许 T0 溢出中断

    #asm("sei")                 // 开放全局中断

    while (1)
    {
        // Place your code here
    };
}
```

上面的程序，就是先利用 CVAVR 的程序生成向导功能进行配置，然后在它生成的程序框架基础上完成的。

在程序中，主程序对 T/C0 进行初始化后进入一个无限的循环。在 T/C0 的中断服务程序

中，重新设置 TCNT0 的初值，并将 PA0 取反输出。这时 PA0 上可以获得 1Hz 的方波输出。

3） 思考与实践

- ✓ 为什么在中断服务程序中要重新设置 TCNT0 的初值？
- ✓ 如何计算 TCNT0 的初值，使得 PA0 输出 0.5Hz 的方波。

例 8.2 2N 分频系统设计二

1）硬件电路

同 2N 分频系统设计一。

2）软件设计

2N 分频系统设计一中使用了 T/C0 的普通模式，因此在中断服务程序中必须重新对 TCNT0 进行初始化。其实，更方便的方法是使用 T/C0 的 CTC 模式，利用 T/C0 的自动重装特性。当 T/C0 工作在 CTC 模式时，计数器 TCNT0 的值与 OCR0 的值比较，一旦相等，在下一次计数脉冲到来时，清另 TCNT0，并产生 T/C0 的比较匹配中断（参考图 8-13）。此时在比较匹配中断服务程序中改变 PA0 的输出即可。

```
/*****************************************
File name           : demo_8_2.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/

#include <mega16.h>

// Timer 0 比较匹配中断服务
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
      PORTA.0=~PORTA.0;      // PA0 取反输出
}

void main(void)
{
    PORTA=0x01;
    DDRA=0x01;

    PORTB=0x01;
    DDRB=0x00;

    // T/C0 初始化
    TCCR0=0x0F;              // T/C0 工作于 CTC 模式，T0 上升沿触发
    TCNT0=0x00;
    OCR0=0x7C;               // 设置 OCR0 的比较值为 124（0x7C）
```

```
    TIMSK=0x02;              // 允许 T/C0 的比较匹配中断
    #asm("sei")              // 开放全局中断

    while (1)
    {
          // Place your code here
    };
  }
```

3） 思考与实践

- ✓ 比较两种方式的特点。
- ✓ 如何利用 T/C0 实现 N 分频？

例 8.3 N 分频系统设计

1）硬件电路

同 2N 分频系统设计，但在 PA0 上得到 N 分频的输出。

2）软件设计

实际上，利用 T/C0 的比较匹配的特点，可以实现 N 分频的系统。

```
/*****************************************
File name           : demo_8_3.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************/

#include <mega16.h>

// Timer 0 溢出中断服务
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
      TCNT0=0xFB;            // 重新设置 TCNT0 的初值
      PORTA = ~PORTA;        // PA0 取反输出
}

// Timer 0 比较匹配中断服务
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
      PORTA = ~PORTA;        // PA0 取反输出
}

void main(void)
{
      PORTA=0x00;
```

```
    DDRA=0x01;

    PORTB=0x01;
    DDRB=0x00;

    // T/C0 初始化
    TCCR0=0x07;             // T/C0 工作于普通模式，T0 上升沿触发
    TCNT0=0xFB;
    OCR0=0xFD;              // 设置 OCR0 的比较值，>TCNT0 的初始值，<0xFF

    TIMSK=0x03;             // 允许 T/C0 的溢出和比较匹配中断
    #asm("sei")             // 开放全局中断

    while (1)
    {
        // Place your code here
    };
}
```

程序 demo_8_3.c 设置 T/C0 工作在普通模式，并结合比较匹配的特性，在比较匹配中断和溢出中断中都改变 PA0 的输出，在 PA0 上获得 5 分频的脉冲信号。

3） 思考与实践

- ✓ 请读者自己分析程序实现 N 分频的原理。
- ✓ 如果要实现 11 分频的输出，TCNT0 的初值应该如何计算，为什么同例 8.1 不同？
- ✓ 在 N 分频的系统中，OCR0 的值应该如何设置？
- ✓ 在 PA0 上输出的方波序列的占空比同例 8.1 和例 8.2 有何不同？
- ✓ 利用这个方法，能否在 PA0 上获得占空比可调的 PWM 波？

8.2.2 定时器应用设计

实际上不管定时/计数器是作为计数器使用还是作为定时器使用，其根本的工作原理并没有改变，都是对一个脉冲系列信号进行计数。通常所谓的定时器，更多的情况是指其计数脉冲信号来自芯片本身的内部。由于内部的计数脉冲信号的频率（周期）是已知的或固定的，因此用户可以根据需要来设定计数器脉冲计数的个数，以获得一个等间隔的定时中断。利用定时中断，可以方便的实现系统定时访问外设或处理事物，以及获得更加准确的延时等等。

同其他一些单片机类似，AVR 的定时/计数器的计数脉冲可以来自外部的引脚，也可以由从内部系统时钟获得，但 AVR 的定时/计数器在内部系统时钟和计数单元之间增加了一个可设置的预分频器，利用该预分频器，定时/计数器可以从内部系统中获得不同频率的计数脉冲信号。表 8.6 给出了系统时钟为 4MHz 时，ATmega16 芯片本身能够提供给 T/C0 的计数脉冲信号的最高计时精度和时宽范围。

表 8.1 T/C0 计时精度和时宽（系统时钟 4MHz）

分频系数	计时频率	最高计时精度 (TCNT0=255)	最宽时宽 (TCNT0=0)
1	4MHz	0.25us	64us
8	500KHz	2us	512us

64	62.5KHz	16us	4.096ms
256	15.625KHz	64us	16.384ms
1024	3906.25Hz	256us	65.536ms

从表中看出，在系统时钟为 4Mh 条件下，8 位的 T/C0 最高计时精度为 0.25us，而最长的时宽可达到 65.536ms。而如果使用 16 位的定时/计数器 T/C1 时，不需要使用辅助软件计数器，就可以非常方便的设计一个时间长达 16.777216 秒（精度为 256us）的定时器，这是其它的 8 位单片机所做不到的。

AVR 单片机的每一个定时/计数器都配备独立的、多达 10 位的预分频器，由软件设定分频系数，与 8/16 位定时/计数器配合，可以提供多种档次的定时时间。使用时可选取最接近的定时档次，即选 8/16 位定时/计数器与分频系数的最优组合，减少了定时误差。所以，AVR 定时/计数器的显著特点之一是：高精度和宽时范围，使得用户应用起来更加灵活和方便。

例 8.4 采用 T/C0 硬件定时器的时钟系统

1) 硬件电路

在第六章中的例 6.5,“六位 LED 数码管动态扫描控制显示设计(一)”中,使用调用 CVAVR 中软件延时函数的方法给出了一个使用 6 个数码管组成时钟系统。采用软件延时，时钟是不准确的，因为一旦系统中使用了中断，就可能打断延时程序的执行，使得延时时间的变化。

而在第七章中的例 7.2,“采用外部中断方式，用外部振荡源为基准的时钟系统”中，系统时钟的基准信号来自外部的标准方波信号源,这样尽管定时时间比采用软件延时方式要准确的多，但采用外部标准方波信号源增加了系统的成本。

实际上更加方便和简单的方式是采用系统本身的时钟信号，配合 T/C0 产生时钟系统的定时信号。下面给出采用 T/C0 硬件定时器实现的时钟系统设计。时钟系统的硬件电路仍旧与图 6-15 相同。

2) 软件设计

下面是一个采用 C 编写的系统源程序。

```
/*********************************************
File name            : demo_8_4.c
Chip type            : ATmega16
Program type         : Application
Clock frequency      : 4.000000 MHz
Memory model         : Small
External SRAM size   : 0
Data Stack size      : 256
*********************************************/

#include <mega16.h>
flash char led_7[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
flash char position[6]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};

char   time[3];                  // 时、分、秒计数单元
char   dis_buff[6];              // 显示缓冲区，存放要显示的 6 个字符的段码值
int    time_counter;             // 中断次数计数单元
char   posit;
bit    point_on, time_1s_ok;
```

```
void display(void)                  // 6 位 LED 数码管动态扫描函数
{
       PORTC = 0xff;
       PORTA = led_7[dis_buff[posit]];
       if (point_on && (posit==2||posit==4)) PORTA |= 0x80;
       PORTC = position[posit];
       if (++posit >=6 ) posit = 0;
}

// Timer 0 比较匹配中断服务
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
       display();                   // 调用 LED 扫描显示
       if (++time_counter>=500)
       {
           time_counter = 0;
           time_1s_ok = 1;
       }
}

void time_to_disbuffer(void)        // 时钟时间送显示缓冲区函数
{
       char i,j=0;
       for (i=0;i<=2;i++)
       {
           dis_buff[j++] = time[i] % 10;
           dis_buff[j++] = time[i] / 10;
       }
}

void main(void)
{
       PORTA=0x00;                  // 显示控制 I/O 端口初始化
       DDRA=0xFF;
       PORTC=0x3F;
       DDRC=0x3F;
       // T/C0 初始化
       TCCR0=0x0B;                  // 内部时钟，64 分频（4M/64=62.5KHz），CTC 模式
       TCNT0=0x00;
       OCR0=0x7C;                   // OCR0 = 0x7C(124),(124+1)/62.5=2ms
       TIMSK=0x02;                  // 允许 T/C0 比较匹配中断

       time[2] = 23; time[1] = 58; time[0] = 55;     // 设时间初值 23:58:55
```

```
        posit = 0;
        time_to_disbuffer();

        #asm("sei")                 // 开放全局中断

        while (1)
        {
            if (time_1s_ok)                     // 1秒到
            {
                time_1s_ok = 0;
                point_on = ~point_on;
                if (++time[0] >= 60)            // 以下时间调整
                {
                    time[0] = 0;
                    if (++time[1] >= 60)
                    {
                        time[1] = 0;
                        if (++time[2] >= 24) time[2] = 0;
                    }
                }
                time_to_disbuffer();            // 新调整好的时间送显示缓冲区
            }
        };
    }
```

该程序中的 LED 动态扫描，时间调整与例 7.2 相同，所不同的是使用了 T/C0 硬件定时。T/C0 工作在 CTC 模式，采用系统时钟经过 64 分频的信号作为计数器的计数脉冲。4M 系统时钟经过 64 分频后为 62.5KHz，周期为 16us。T/C0 的比较寄存器 OCR0 的值为 124（0x7C），因此 T/C0 每计数 125 次产生一次比较匹配中断，中断的间隔时间为 16*125=2ms。

在 T/C0 的比较匹配中断服务中，中断服务的内容同例 7.2，首先进行 LED 的扫描，即每位 LED 的扫描间隔（点亮时间）为 2 毫秒，然后中断次数计数器 time_counter 加 1。一旦 time_counter 加到 500，则置位 time_1s_ok，表示 1 秒时间到。

主程序中检测秒标志 time_1s_ok，当秒标志置位，则进行时间的调整，然后将新的时间送显示缓冲区中。

3) 思考与实践

该程序同第六章的例 6.5、第七章的例 7.2 有许多地方相同或类似，读者可以对三个程序进行全面的分析和比较，例如它们各自的优点和缺点以及 MCU 的利用率等。

例 8.5 基于 T/C2 硬件方式的 2N 分频方波发生器

1) 硬件电路

在本章的例 8.1 和 8.2 中，分别利用 T/C0 两种不同工作模式实现了 2N 分频的功能。在本例中，我们给出一个基于 ATmega16 本身系统时钟，利用 T/C2 构成的硬件方式的 2N 分频方波发生器的设计。

在这个设计中，T/C2 工作在 CTC 模式下，并利用其比较匹配输出的性能，直接由 OC2（PD7）输出 2N 分频的方波。具体工作原理参见本章对 T/C 结构中比较匹配输出的介绍和表 8.2。

在 CTC 模式下利用比较匹配输出单元产生波形输出时，应设置 OC2 的输出方式为触发方式（COM2[1:0]=1)。OC2 输出波形的最高频率为 $f_{OC2}=f_{clk\_I/O}/2$（OCR2=0x00)。其它频率输出由下式确定，式中 K 的取值为 1、8、32、64、128、256 或 1024。

$$f_{OC\,2}=\frac{f_{\text{clk\_I/O}}}{2K\,(1+OCR\;2)}$$

2） 软件设计

```
/*********************************************
File name            : demo_8_5.c
Chip type            : ATmega16
Program type         : Application
Clock frequency      : 4.000000 MHz
Memory model         : Small
External SRAM size   : 0
Data Stack size      : 256
*********************************************/

#include <mega16.h>

void main(void)
{
    DDRD=0x80;                  // 设置 PD7 为输出方式，即 OC2 方波输出脚
    // T/C2 初始化
    TCCR2=0x19;           //内部时钟，1 分频（4M)，CTC 模式，OC2 的输出方式为触发方式
    TCNT2=0x00;
    OCR2=52;

    while (1)
    { };
}
```

上面的程序非常简单，仅在初始化时对 C/T2 进行设置，并设置 PD7 脚作为 OC2 的输出，然后进入一个无限的循环。该程序没有使用任何中断，在对 C/T2 做了必要的初始化设置后，就不再对 T/C2 做其它的处理，但程序运行后，在 PD7 的引脚上输出了一个 50%的 37.736KHz 的方波序列信号。因此，这种方式的波形发生器是基于 T/C2 硬件方式的，不需要软件的干预。通过选择不同的计数频率的时钟信号，以及配合 OCR2 不同的值就可以获得更多频率的方波输出信号。

3） 思考与实践

- ✓ 将该程序同例 8.1 和 8.2 进行比较，读者可以对三个程序进行全面的分析，例如它们各自的优点和缺点以及 MCU 的利用率等。
- ✓ 如果系统需要一个频率为 50Hz 左右 50%占空比的方波信号，那么应该 T/C2、OCR2 如何设置和计算？理论上的输出频率是多少？与 50Hz 的误差是多少（假定系统时钟频率为 4M)。

8.3　PWM 脉宽调制波的产生和应用

8.3.1 PWM 脉宽调制波

PWM 是脉冲宽度调制的简称。实际上，PWM 波也是一个连续的方波，但在一个周期中，其高电平和低电平的占空比是不同的。一个典型 PWM 的波形如图 8-15 所示。

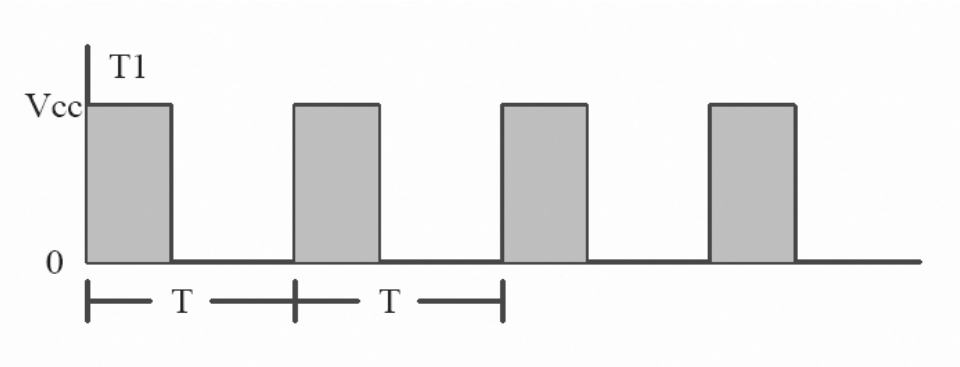

图 8-15 典型的 PWM 波

在图中，T 是 PWM 波的周期，T1 是高电平的宽度，Vcc 是高电平值。当该 PWM 波经过一个积分器后（低通滤波器）后，我们可以得到其输出的平均电压为：

$$V = \frac{Vcc * T1}{T}$$

式中，T1/T 称为 PWM 波的占空比。当控制调节和改变 T1 的宽度，即改变 PWM 的占空比，就可以的到不同的平均电压输出。因此在实际应用中，常利用 PWM 波的输出，实现 D/A 转换，调节电压或电流控制改变马达的转速，实现变频控制等功能。

一个 PWM 方波的参数有频率、占空比和相位（在一个 PWN 周期中，高低电平转换的起始时间），其中频率和占空比为主要的参数。图 8-16 为 3 个占空比都为 2/3 的 PWM 波形，尽管他们输出的平均电压是一样的，但其中（b）的频率比（a）高一倍，相位相同；而（c）与(a)的频率相同，但相位不同。

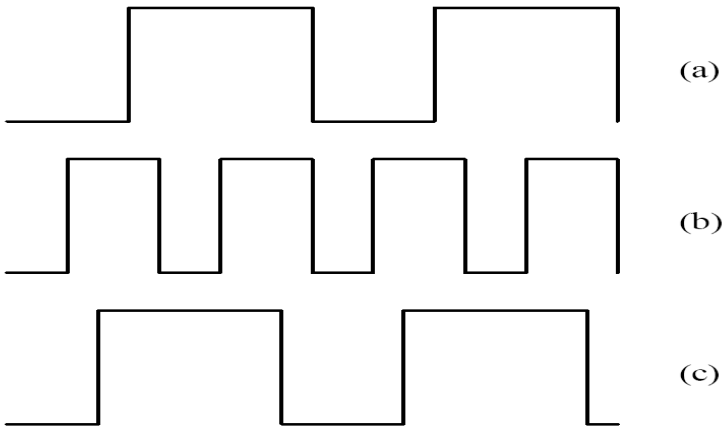

图 5.9 占空比相同，频率与相位不同的 PWM

在实际应用中，除了要考虑如何正确的控制和调整 PWM 波的占空比，获得达到要求的平均电压的输出外，还需要综合的考虑 PWM 的周期、PWM 波占空比调节的精度（通常用 BIT 位表示）、积分器的设计等。而且这些因素相互之间也是互相牵连的。根据 PWM 的特点，在使用 AVR 定时/计数器设计输出 PWM 时应注意以下几点：

- ✓ 首先应根据实际的情况，确定需要输出的 PWM 波的频率范围。这个频率与控制的对象有关。如输出 PWM 波用于控制灯的亮度，由于人眼不能分辨 42Hz 以上的频率，所以 PWM 的频率应高于 42Hz，否则人眼会察觉到灯的闪烁。PWM 波的频率越高，经过积分器输出的电压也越平滑。
- ✓ 同时还要考虑占空比的调节精度。同样，PWM 波占空比的调节精度越高，经过积分器输出的电压也越平滑。但占空比的调节精度与 PWM 波的频率是一对矛盾，在相同的系统时钟频率时，提高占空比的调节精度，将导致 PWM 波频率的降低。
- ✓ PWM 波本身还是数字脉冲波，其中含有大量丰富的高频成分，因此在实际使用中，还需要一个好的积分器电路，如采用有源低通滤波器，或多阶滤波器等，能将高频成分有效的去除掉，从而获得比较好的模拟变化信号。

ATmega16 的 T/C0 和 T/C2 都具备产生 PWM 波的功能，由于它们计数器的长度为 8 位，所以是固定 8 位精度的 PWM 波发生器。即 PWM 的调节精度为 8 位，而且 PWM 波的周期也只能取决于系统时钟和分频系数（即作为计数器计数脉冲的频率）。T/C0、T/C2 的工作模式中有快速 PWM 模式（WGMn[1:0]=3）和相位可调 PWM 模式（WGMn[1:0]=1）两种（参见图 8-8 和图 8-9）。

快速 PWM 模式可以的到比较高频率、相位固定的 PWM 输出，适合一些要求输出 PWM 频率较高，频率相位固定的应用。快速 PWM 模式中，计数器仅工作在单程正向计数方式，计数器的上限值决定 PWM 的频率，而比较匹配寄存器 OCRn 的值决定了占空比的大小。快速 PWM 频率的计算公式为：

PWM 频率 = 系统时钟频率/（分频系数 * 256）

相位可调 PWM 模式的输出频率比较低，因为此时计数器工作在双向计数方式。同样计数器的上限值决定了 PWM 的频率，比较匹配寄存器的值决定了占空比的大小。PWM 频率的计算公式为：

PWM 频率 = 系统时钟频率/（分频系数 * 510）

相位调整 PWM 模式适合在要求输出 PWM 频率较低，频率固定应用中。由于计数器工作在双向模式，当调整占空比时（改变 OCR0 的值），PWM 的相位也相应的跟着变化（Phase Correct），但由于其产生的 PWM 波形是对称的，更加适合在电机控制中使用。

T/C0、T/C2 两种 PWM 模式都是固定 8 位的 PWM，计数器 TCNTn 的上限值为固定的 0xFF（8 位 T/C），而比较匹配寄存器 OCRn 的值与计数器上限值之比即为占空比。

8.3.2 基于比较匹配输出的脉冲宽度调制 PWM

利用 ATmega16 定时/计数器的 PWM 模式，与比较匹配寄存器相配合，能直接生成占空比可变的方波信号，即脉冲宽度调制输出 PWM 信号。快速 PWM 模式工作的基本原理是：定时/计数器在计数过程中，内部硬件电路会将计数值（TCNTn）和比较寄存器（OCRn）中的值进行比较，当两个值相匹配（相等）时，能自动置位（清另）一个固定引脚的输出电平（OCnx），而当计数器的值到达最大值时，则自动将该引脚的输出电平（OCnx）清另（置位）（参考图 8-8）。因此，在程序中改变比较寄存器中的值（通常在溢出中断服务中），定时/计数器就能自动产生不同占空比的方波（PWM）信号输出了。

例 8.6 利用 T/C0 的 PWM 功能产生正弦波

1） 硬件电路

在本例中将 T/C0 的 PWM 方式来产生一个 1KHz 左右的正弦波，正弦波的幅度为 0-Vcc。

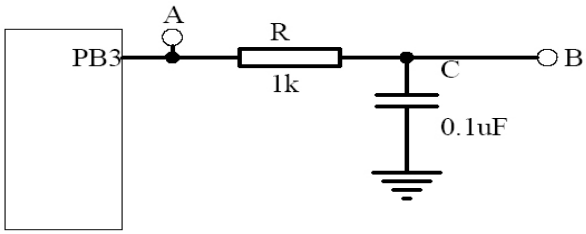

图 8-16 采用 T/C0 的 PWM 功能产生正弦波

图 8-16 为电原理图。PB3 为 ATmega16 的 T/C0 匹配输出脚 OC0，该脚上的 PWM 波经过由电阻 R 和电容 C 构成的简单滤波电路滤掉高频进行平滑后，在 B 点获得模拟的正弦波信号。

2） 软件设计

首先按照下面的公式建立一个正弦波样本表，样本表将一个正弦波周期分为 128 个点，每点按 8 位量化（1 对应最低幅度，255 对应最高幅值 Vcc）：

$$f(x) = 128 + 127 * \sin(2\pi x/127) \quad x \in [0 \cdots 127]$$

如果在一个正弦波周期中采用 128 个样点，那么对应 1KHz 的正弦波 PWM 的频率为 128KHz。实际上，根据采样频率至少为信号频率的 2 倍的采样定理来计算，PWM 的频率的理论值应大于 2KHz。考虑到尽量提高 PWM 的输出精度，实际设计中使用的 PWM 频率为 16KHz，即一个正弦波周期中输出 16 个正弦波样本值。这意味着在 128 点的正弦波样本表中，要每隔 8 点取出一点作为 PWM 的输出。

程序中使用 ATmega16 的 8 位 T/C0，工作模式为快速 PWM 模式输出，系统时钟为 4MHz，分频系数为 1，其可以产生 PWM 波的频率为： 4000000Hz / 256 = 15625Hz。每 16 次输出构成一个周期正弦波，那么正弦波的频率为 15625/16=976.56Hz。PWM 由 OC0（PB3）引脚输出。参考程序如下。

```
/*********************************************
File name           : demo_8_6.c
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*********************************************/

#include <mega16.h>

flash char auc_SinParam[128] = {
128,134,140,147,153,159,165,171,177,182,188,193,198,204,208,213,
218,222,226,230,233,237,240,242,245,247,249,251,252,253,254,254,
254,254,253,252,251,250,248,246,244,241,238,235,232,228,224,220,
215,211,206,201,196,191,185,179,174,168,162,156,150,144,137,131,
125,119,112,106,100,94,88,82,77,71,65,60,55,50,45,41,
36,32,28,24,21,18,15,12,10,8,6,5,4,3,2,2,
2,2,3,4,5,7,9,11,14,16,19,23,26,30,34,38,
43,48,52,57,63,68,74,79,85,91,97,103,109,116,122,128};    // 128 点正弦波样本值
```

```
char x_SW = 8,X_LUT = 0;

// T/C0 溢出中断服务
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
        X_LUT += x_SW;                  // 新样点指针
        if (X_LUT > 127) X_LUT -= 128;  // 样点指针调整
        OCR0 = auc_SinParam[X_LUT];     // 取样点指针到比较匹配寄存器
        // OCR0+=1;
}

void main(void)
{
        DDRB=0x08;                      // PB3 输出方式，作为 OC0 输出 PWM 波
        // Timer/Counter 0 initialization
        // Clock source: System Clock
        // Clock value: 4000.000 kHz
        // Mode: Fast PWM top=FFh
        // OC0 output: Non-Inverted PWM
        TCCR0=0x69;
        OCR0=128;

        TIMSK=0x01;                     // 允许 T/C0 溢出中断
        #asm("sei")                     // 开放全局中断

        while (1)
        {};
}
```

程序中，在每次的计数器溢出中断的服务中取出一个正弦波的样点值到比较匹配寄存器中，用于调整下一个 PWM 的脉冲宽度，这样在 PB3 引脚上输出了按正弦波调制的 PWM 方波。当 PB3 的输出通过低通滤波器后，便得到一个 976.56Hz 的正弦波了。

如要得到更精确的 1KHz 的正弦波，可使用 ATmega16 的 T/C1，选择工作模式 10，设置 ICR1=250 为计数器的上限值（将在以后的章节中给出）。

图 8-17 为程序运行后使用示波器在 B 点测量的实际波形。B 点的波形轮廓非常接近正弦波，频率为 976.7Hz，幅度在 0-5v 之间。由于滤波电路由非常简单的 RC 电路构成，因此还能够看出在正弦波的轮廓上的高频成分。如果用示波器在 A 点测量的话，可以观察到一个占空比变化的序列方波。

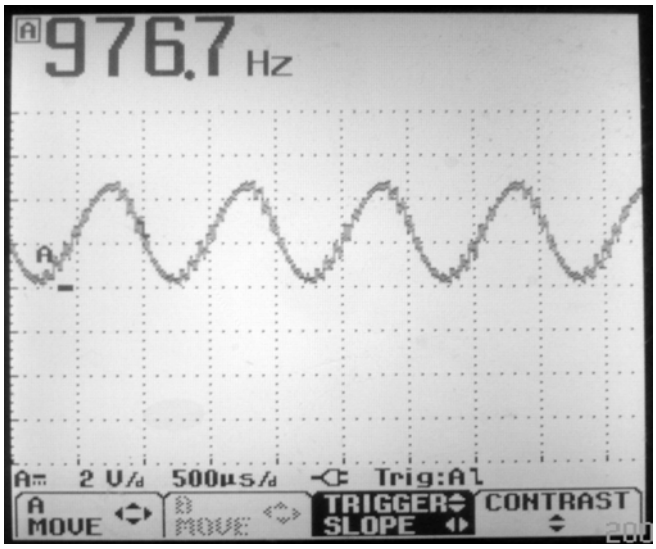

图 8-17 实际测量到的正弦波输出波形

3） 思考与实践

- ✓ 根据 T/C0 的 PWM 模式，再结合本例，详细描述快速 PWM 的工作原理。
- ✓ 减小或增加程序中变量 x_SW 的值时，B 点输出的波形有何变化？如何进行理论的计算？

如果将 T/C0 中断服务程序中的语句清除到，用 OCR0+=1 代替时，分析 B 点输出的波形，以及频率，并用示波器观察。