

MSP430 定时器的使用

南京航空航天大学 魏小龙

MSP430 系列单片机的定时器相当丰富。有基本定时器 BT、定时器 TA、定时器 TB、看门狗定时器 WDT 等。其中看门狗主要用于程序的完善性控制等方面；基本定时器除了可以用于定时之外，还可以用于液晶显示的时序控制；TA、TB 基本相同，而且功能复杂，这里将详细讲解，并使用之实现时钟。

看门狗的目的在于阻止程序跑飞，其原理在于：看门狗定时器设置一定时时间，比如 250 毫秒，这个时间是所有用户程序一定能在该时间内执行完该程序的一个时间，设置好这个定时时间之后，所有用户程序就必须在这个设定的时间内将看门狗计数器的值清零，使计数器重新计数，如果 CPU 执行程序正确，则看门狗计数器始终能在规定的时间内被用户程序清零而始终不能计数到 250 毫秒，而当 CPU 执行程序跑飞（PC 值指向用户程序以外），看门狗计数器得不到用户程序清零，能计数到 250 毫秒，发生溢出，导致 CPU 复位，这样 CPU 又会重新运行用户程序。所以使用看门狗时，用户软件必须周期性地在 WDTCTL 的 CNTCL 位写“1”，使得看门狗计数器复位以防止其超过设定的定时时间。

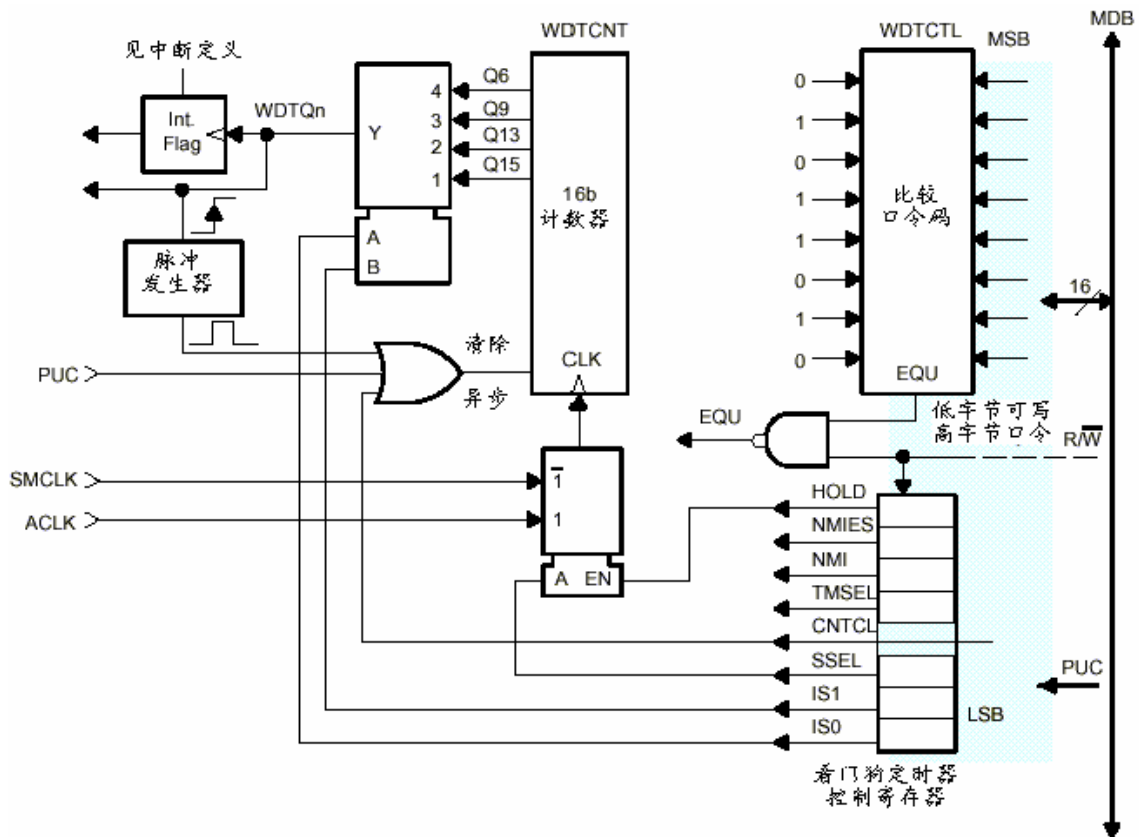


图 1 看门狗定时器的结构原理

BT 是 MSP430X3XX、MSP430F4XX 系列器件中的模块。它也是一个定时器，它通常向其它外围模块提供低频控制信号。BT 可以是两个 8 位定时器也可以是一个 16 位定时器，它有两个计数单元(BTCNT1、BTCNT2)与一个控制单元(BTCTL)。通过控制寄存器 BTCTL 的设置，用户可以方便地使用 BT。

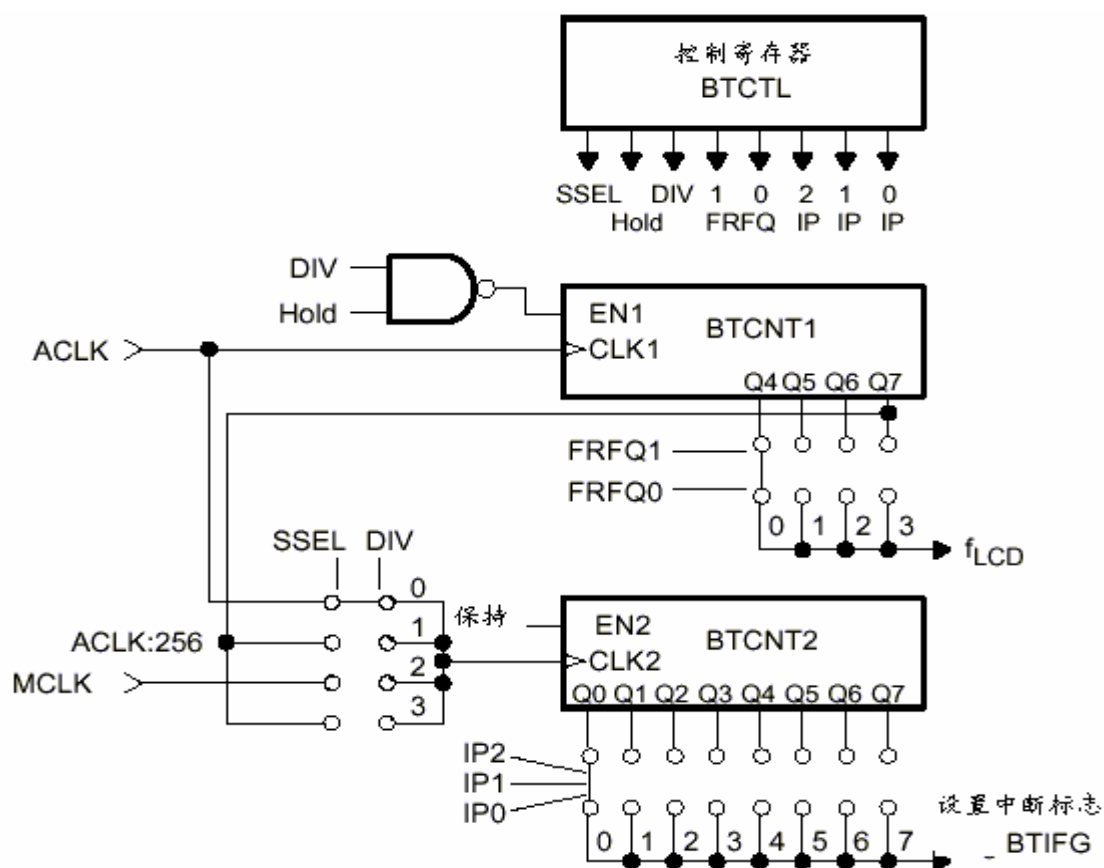


图2 定时器BT的结构原理

TA、TB 基本相同，这里将详细讲解定时器 A

定时器 A 是 MSP430 所有系列都有的模块，是一个用途非常广泛的通用 16 位定时/计数器。它有以下一些特点：

- 16 位计数器，四种工作模式；
- 多种可选的计数器时钟源；
- 多个具有可配置输入端的捕获/比较寄存器；
- 有 8 种输出模式的多个可配置的输出单元；

Timer_A 可支持同时进行的多种时序控制、多个捕获/比较功能、多种输出波形 (PWM)，也可以是几种功能的组合，每个捕获/比较寄存器可以以硬件方式支持实现串行通讯。

Timer_A 具有中断能力。中断可由计数器溢出引起，也可来自具有捕获或比较功能的捕获/比较寄存器。每个捕获/比较模块可独立编程，由捕获或比较外部信号以产生中断，外部信号可以是上升沿，也可下降沿，也可都是。

在不同的 MSP430 器件中，Timer_A 模块中的捕获/比较器的数量不一样，比如在 MSP430F435 中 Timer_A 模块含有 3 个捕获/比较器 (简称 CCR)，因此也经常称 Timer_A3，表示该模块含有 3 个 CCR。还是先看看 Timer_A 的结构原理图。见图 3。图中，可以将 Timer_A 分解成几个部分：计数器部分、捕获/比较寄存器、输出单元。其中计数器部分完成时钟源的选择、分频，模式控制，计数等功能。捕获/比较器用于捕获事件发生的时间或产生时间间隔。输出模块用于产生用户需要的输出信号。

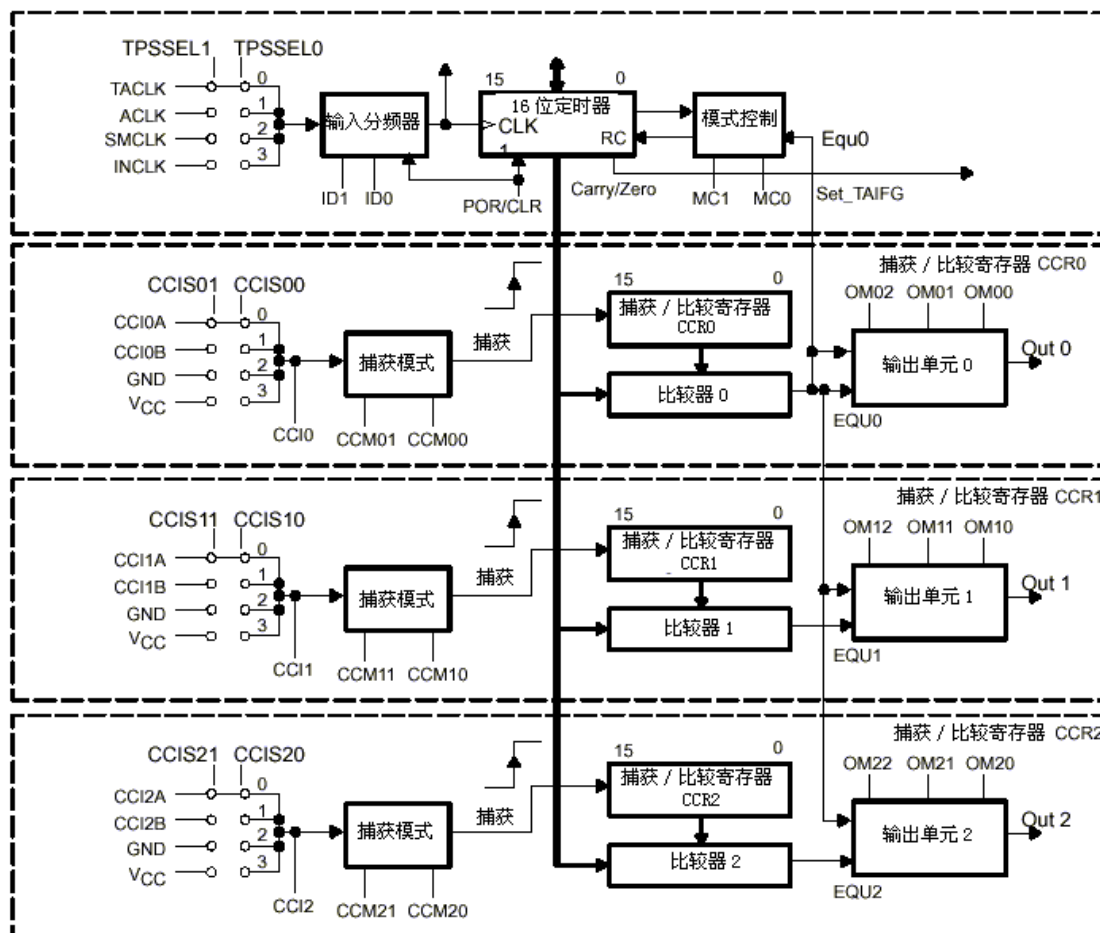


图 3 定时器 A 的结构原理

定时器 A 的寄存器在 IAR 的调试环境中见图 4 所示。操作这些寄存器就可以实现 TA 的所有功能。

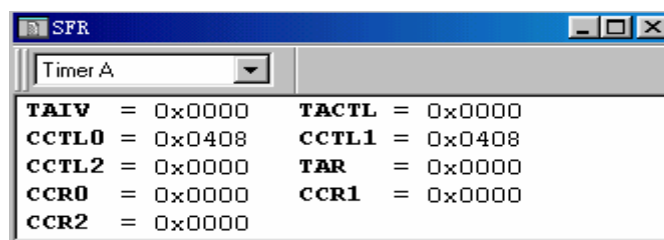


图 4 TA 的寄存器

其中 TACTL 为最主要的控制寄存器，它决定 TA 的输入时钟信号、TA 的工作模式、TA 的开启与停止、中断的申请等工作。TACTL 寄存器为 16 位寄存器，必须使用字指令对其访问。该寄存器在 POR 信号后全部复位，但在 PUC 信号后不受影响。下面是该寄存器中各位的含义。

15~10	9	8	7	6	5	4	3	2	1	0
未用	SSEL1	SSEL0	ID1	ID0	MC1	MC0	未用	CLR	TAIE	TAIFG

SSEL1、SSEL0 选择输入分频器的输入时钟源。

SSEL1	SSEL0	输入信号	输入信号说明
0	0	TACLK	使用外部引脚信号作为输入（见手册）
0	1	ACLK	辅助时钟
1	0	MCLK	系统主时钟

	1	1	INCLK	外部输入时钟（见芯片手册）
ID1、ID0	选择输入分频器的分频系数。			
	00:	直通，不分频		
	01:	1/2 分频		
	10:	1/4 分频		
	10:	1/8 分频		
MC1、MC0	选择定时器模式			
	00:	停止模式，用于定时器暂停		
	01:	增计数到 CCR0 模式，该模式下，计数器计数到 CCR0，再清零计数		
	10:	连续增计数模式，计数器增计数到“FFFFH”，再清零计数		
	11:	增/减模式，增计数到 CCR0，再减计数到 0		
CLR	定时器清除位，计数器内容清零			
TAIE	中断允许位，该位允许定时器溢出中断			
TAIFG	定时器溢出标志位，在不同的定时器模式下，该位置位条件不一样。			
	增计数模式：	当定时器由 CCR0 计数到“0”时 TAIFG 置位		
	连续模式：	当定时器由 0FFFFH 计数到“0”时 TAIFG 置位		
	增/减模式：	当定时器由“1”减计数到“0”时 TAIFG 置位		

由此可见 TACTL 寄存器几乎控制了 Timer_A 的第一部分——计数器部分。下面我们对照原理图理解。

工作模式控制如图 5。由 MC1、MC0 两位选择。

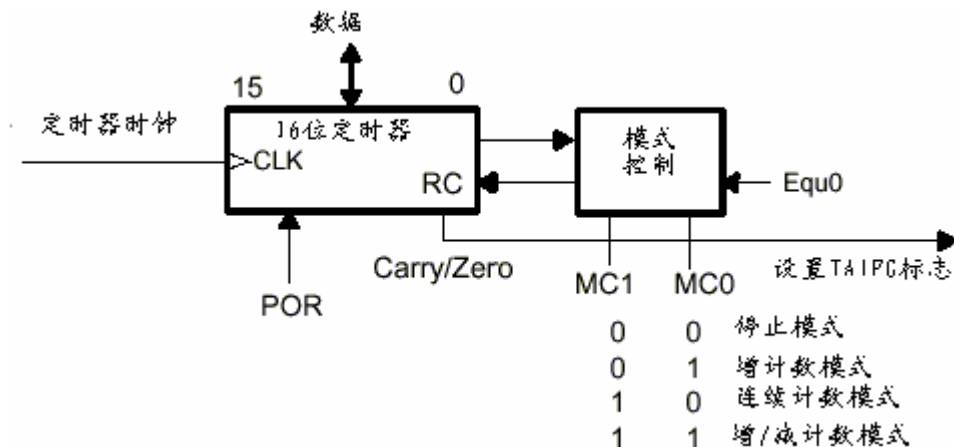


图 5 模式控制原理图

计数器输入时钟源的选择与分频控制如图 6。由 SSEL0、SSEL1 两位选择时钟源，然后再由 ID0、ID1 选择分频系数将输入信号分频，分频后的信号才用于计数器计数。在 MSP430F4XX 系列器件中，INCLK 信号经过反向驱动之后再送入，这一点与其他器件有点差别。

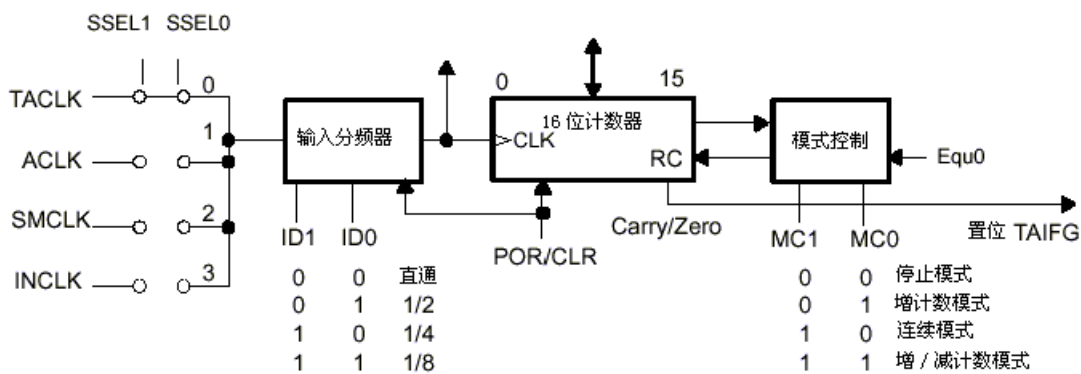


图 6 计数器输入时钟源的选择与分频控制

Timer_A 定时器共有 4 种工作模式。由控制寄存器 TACTL 中 MC0、MC1 两位决定（详见 TACTL 寄存器介绍）。

停止模式

当 MC1=MC0=“0”时，定时器暂停。暂停时定时器的值（TAR 的内容）不受影响。当定时器在暂停后重新计数时，计数器将从暂停时的值开始以暂停前的计数方向计数。如果不能这样，则可通过 TACTL 中 CLR 控制位来清除定时器的方向记忆特性。

增计数模式

当 MC0=“1”，MC1=“0”时定时器工作在增计数模式。该模式用于定时周期小于 65536 的连续计数方式。捕获/比较寄存器 CCR0 的数据定义定时器的计数周期。

增计数模式的计数器活动规则：当计数器 TAR 增计数到 CCR0 的值或当计数值与 CCR0 相等（或定时器值大于 CCR0 的值）时，定时器复位并从“0”开始重新计数。图 7 说明了增计数模式的计数过程。当定时器的值等于 CCR0 的值时，设置标志位 CCIFG 为“1”，而当定时器从 CCR0 计数到“0”时，设置标志位 TAIFG 为“1”。

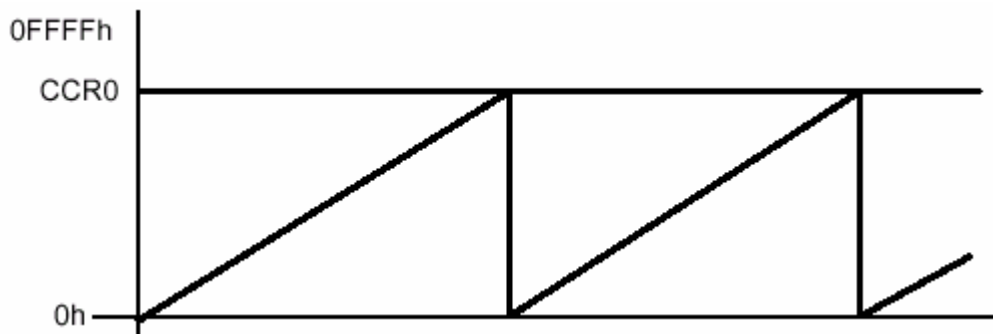


图 7 增计数模式的定时器 TAR

连续计数模式

在需要 65536 个时钟周期的定时应用场合常用连续模式。其典型的应用是产生多个独立的时序信号。在这种计数模式中，CCR0 的工作方式与其他比较寄存器的工作方式相同。利用捕获/比较寄存器与各输出单元的输出模式可以捕获各种外部事件发生的定时器数据（事件发生的时间）或者产生不同类型的输出信号。

连续模式的计数器活动规则：定时器从它的当前值开始计数，当计数到 0FFFFH 后又从“0”开始重新计数，如图 8，当定时器从“0FFFFH”计数到“0”时，设置标志位 TAIFG。

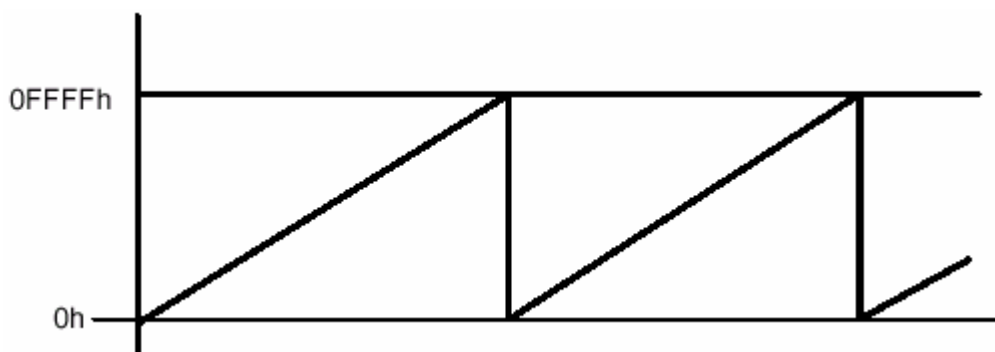


图 8 连续计数模式定时器 TAR

如果相应的中断允许，则每当一个定时间隔到都会产生中断请求。那么在连续模式下，须将下一事件发生的时间在当前的中断程序中加到 CCRx 中。在图 9 可看出这种情况：每隔 Δt 产生中断，须在定时器等于 CCR0a 时产生的中断服务程序中，将 CCR0b 加到 CCR0 寄存器中。

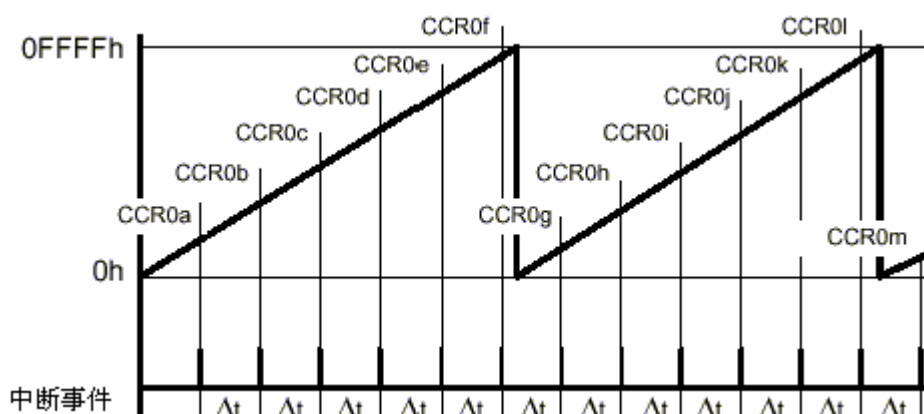


图 9 在连续模式时中断与 CCRx 的关系

增 / 减计数模式

在增减计数模式下，计数器 TAR 的值先增后减：当增计数到 CCR0 的值时，计数器停止增计数，变为减计数，当减到“0”时，设置标志位 TAIFG。由此可见这种模式的计数周期为 CCR0 值的两倍。所以常用于须得到对称波形的场合。这种模式时计数器中数值变化情况如图 10 所示。

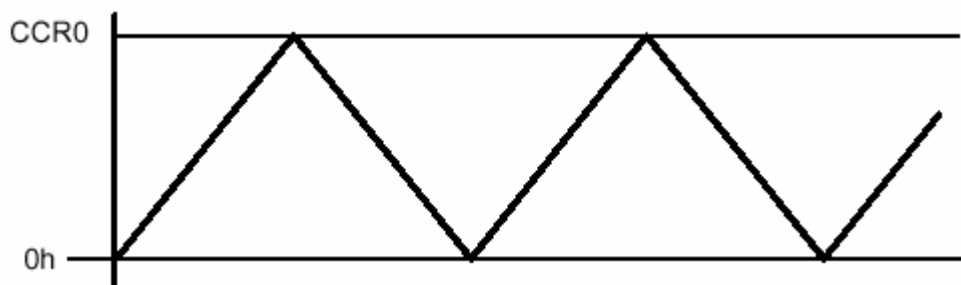


图 10 增 / 减计数模式时的计数器 TAR

增/减模式时，中断标志 CCIFG0、TAIFG 会在相等的时间间隔置位。在一个完整的周期中，每个标志位只置位一次，分别在半周期时发生。当定时器 TAR 的值从 CCR0-1 增计数到 CCR0 时，中断标志 CCIFG0 置位。当定时器从“1”减计数到“0”时，中断标志 TAIFG

置位。

下面将以设计一段时间的定时为例讲述各种定时器如何初始化。

1、 BT 的初始化：（假设允许中断）

```
IE2 |= BTIE;                // Enable BT interrupt
BTCTL = BTSSEL+BTIP2+BTIP1+BTIP0;
_EINT();                     // Enable interrupts
```

2、 WDT 的初始化：（假设允许中断）

```
WDTCTL = WDT_MDLY_32;       // Set Watchdog Timer interval to ~30ms
IE1 |= WDTIE;                // Enable WDT interrupt
_EINT();                     // Enable interrupts
```

3、 TA 的初始化：（假设允许中断）

```
TACTL = TASSEL1 + TAC        // SMCLK, clear TAR
CCTL0 = CCIE;                // CCR0 interrupt enabled
CCR0 = 50000;
TACTL |= MC1;                // Start Timer_A in continuous mode
_EINT();                     // Enable interrupts
```

4、 TB 的初始化：（假设允许中断）

```
TBCTL = TBSSEL1 + TBCLR;     // SMCLK, clear TAR
TBCCTL0 = CCIE;              // CCR0 interrupt enabled
TBCCR0 = 50000;
TBCTL |= MC1;                // Start Timer_A in continuous mode
_EINT();                     // Enable interrupts
```

综合举例：使用 TA 设计时钟，并在液晶上显示。

首先，设置 LCD：

```
LCDCTL = 0XFD;
BTCTL = BTFRFQ1;             // STK LCD freq
P5SEL = 0xFC;                // Common and Rxx all selected
```

然后，设置定时器 TA。

```
TACTL = TASSEL1 + TACLRL;    // SMCLK, clear TAR
CCTL0 = CCIE;                // CCR0 interrupt enabled
CCR0 = 20000;
TACTL |= MC1;                // Start Timer_A in continuous mode
```

最后，打开中断，写中断服务程序，详细的程序清单如下：

```
#include <msp430x44x.h>
char digit[20] = {1,0,0,0,0,2,1,8};
unsigned char distab[] = { 0xaf,0x06,0x6d,0x4f,
                           0xc6,0xcb,0xeb,0x0e,
                           0xef,0xcf
                          };

void main(void)
{
    int i;
    WDTCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
```

```

FLL_CTL0 |= XCAP14PF;           // Configure load caps
LCDCTL = 0XFD;
BTCTL = BTFRFQ1;                // STK LCD freq
P5SEL = 0xFC;                   // Common and Rxx all selected
TACTL = TASSEL1 + TACLR;        // SMCLK, clear TAR
CCTL0 = CCIE;                   // CCR0 interrupt enabled
CCR0 = 20000;
TACTL |= MC1;                   // Start Timer_A in continuous mode
_EINT();                         // Enable interrupts
for (;;)
{
    _BIS_SR(CPUOFF);            // CPU off
    _NOP();                     // Required only for C-spy
}
}

interrupt[TIMERA0_VECTOR] void Timer_A (void)
{
    char i=0;
    CCR0 += 20000;               // Add Offset to CCR0
    LCDMEM[7]=0xa;  LCDMEM[8]=0x90;  LCDMEM[9]=0x12;  LCDMEM[10]=0x80;
    LCDMEM[11]=0x2;  LCDMEM[12]=0x93;  LCDMEM[13]=0x72;  LCDMEM[14]=0x5b;
    LCDMEM[15]=0x94;
    for(i=0;i<6;i++)
        LCDMEM[i+1]=distab[i];
    digit[0]++;
    if(digit[0]==50)
    {
        digit[0]=0;
        digit[1]++;
        if(digit[1]==10)
        {
            digit[1]=0;
            digit[2]++;
            if(digit[2]==6)
            {
                digit[2]=0;
                digit[3]++;
                if(digit[3]==10)
                {
                    digit[3]=0;
                    digit[4]++;
                    if(digit[4]==6)
                    {
                        digit[4]=0;

```


}
}
}
}
}
}