Application Note

单个 MSP430™ 计时器模块的多时基

Katie Pier *MSP430 Applications*

摘要

超低功耗微控制器 MSP430™ 上的计时器模块通常将多个不同的输出基于单个时基（单个计时器周期）。对于 MSP430 器件上脉宽调制（PWM）信号的典型实现尤其如此，其中一个捕获比较寄存器（TxCCR0）设置周期，而其余寄存器（TxCCRx）只设置不同的占空比。不过，一些应用中需要多个时基来生成多个输出频率。通常，这通过使用多个计时器模块来实现，但这可能需要升级到具有更多额外特性的器件，而应用可能并不需要那些额外特性。不过，为了获得较小的软件开销，可以在单个 MSP430 计时器模块上实现多个时基，从而允许在更为简单的 MSP430 器件上实现更多特性。通过允许使用更小的 MSP430 器件，这或许能够降低成本和缩小电路板空间，而在较大的 MSP430 器件上，这样便可以支持要求实现大量不同输出频率的新应用。

本应用报告中所述的源代码及其他文件可以从以下位置下载：https://www.ti.com/cn/lit/zip/slaa513。

内容

商标

1 典型单时基方法

在一些应用中，必须同时生成多个具有独特频率的信号。在 MSP430 MCU 的典型用法中，计时器模块上使用向上计数模式或向上/向下计数模式，并且生成的每个频率都对应于一个唯一的计时器模块。每个计时器模块的捕获比较寄存器 0（TxCCR0）均在程序开头设置，以设定计时器的周期。每个模块的额外捕获比较寄存器（TxCCRx）都可以在程序开头设置为某个值，以生成不同的占空比，但由于计时器只能向上计数到 TxCCR0 中的值，因此计时器模块上的一切都是相对于这个相同计时器周期来完成的。在这种方法中，一切都在计时器工作开始时设置，并且设置之后会一直保持不变 – 一切都在硬件内完成，无需在 ISR 中重新加载任何值。图 1-1 显示了向上计数模式下 TxCCR0 值与周期值 t0 之间的关系。

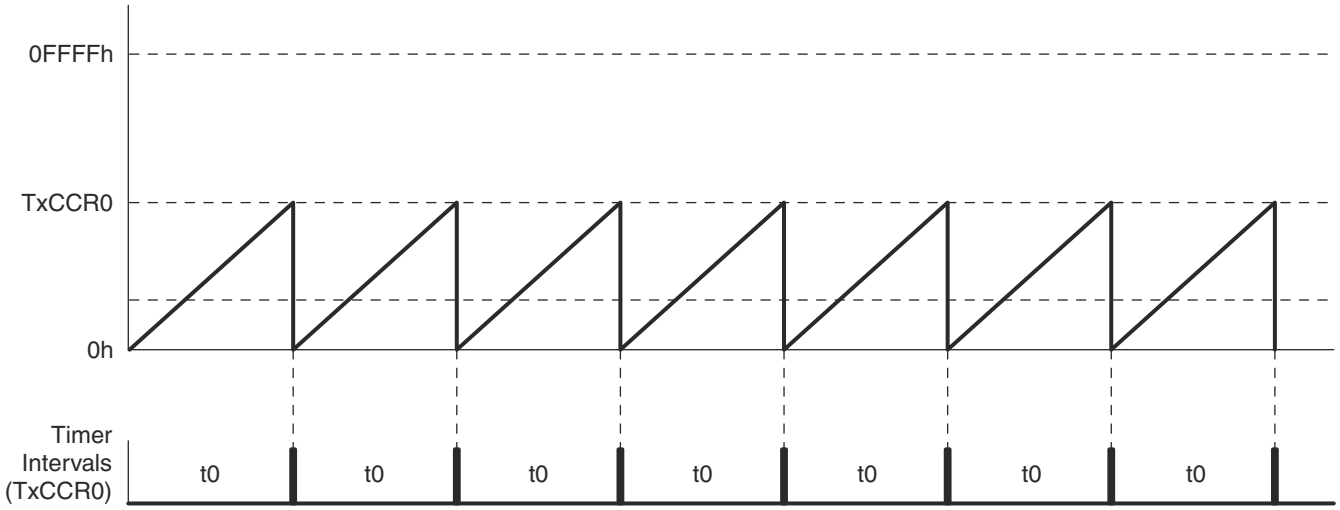


图 1-1. 向上计数模式时间间隔

使用单时基方法时，可在特定 MSP430 器件上同时生成的频率数量取决于器件上的计时器模块数量，而可同时生成的占空比数量则取决于每个模块上 TxCCRx 寄存器数量减去 1。

例如，MSP430F5529 器件包括具有五个捕获/比较（CC）寄存器的 TA0、具有三个 CC 寄存器的 TA1、具有三个 CC 寄存器的 TA2，以及具有七个 CC 寄存器的 TB0。因此，在 MSP430F5529 上使用单时基方法时：

频率数量 = 3 个 Timer_A 模块 + 1 个 Timer_B 模块 = 4 个独立频率

占空比数量 TA0 = 5 CC – 1 = 4 个占空比（TA0 频率下）

占空比数量 TA1 = 3 CC – 1 = 2 个占空比（TA1 频率下）

占空比数量 TA2 = 3 CC – 1 = 2 个占空比（TA2 频率下）

占空比数量 TB0 = 7 CC – 1 = 6 个占空比（TB0 频率下）

2 多时基方法

可以通过使用连续模式在同一计时器模块上实现多个计时器周期。在连续模式下，计时器始终会向上计数到 0xFFFF，然后回滚，而不是每次在 TxCCR0 处复位。这意味着，周期不再是通过将常数值置于 TxCCR0 中来进行设置，而是由 TxCCRx 上一个值和下一个值之间的差值来为*每个捕获比较寄存器*设置周期。例如，这意味着在代码中，我们并不是在程序开头设置 TACCR0 = 0xFF，而是需要在每次计数器达到 TxCCR0 时动态更改 TxCCR0（“TACCR0 += 0xFF”）。由于每个 TxCCRx 寄存器现在都完全独立于 TxCCR0，因此可以针对每个计时器模块上的每个 TxCCRx 寄存器生成一个不同的频率，从而显著增加可生成的频率数量。图 2-1 显示了连续模式下 TxCCR0 值与两个独立周期 t0 和 t1 之间的关系。t0 对应于每次中断时增加到 TxCCR0 的计数，而 t1 对应于每次中断时增加到 TxCCR1 的计数，这会生成两个不同的周期。

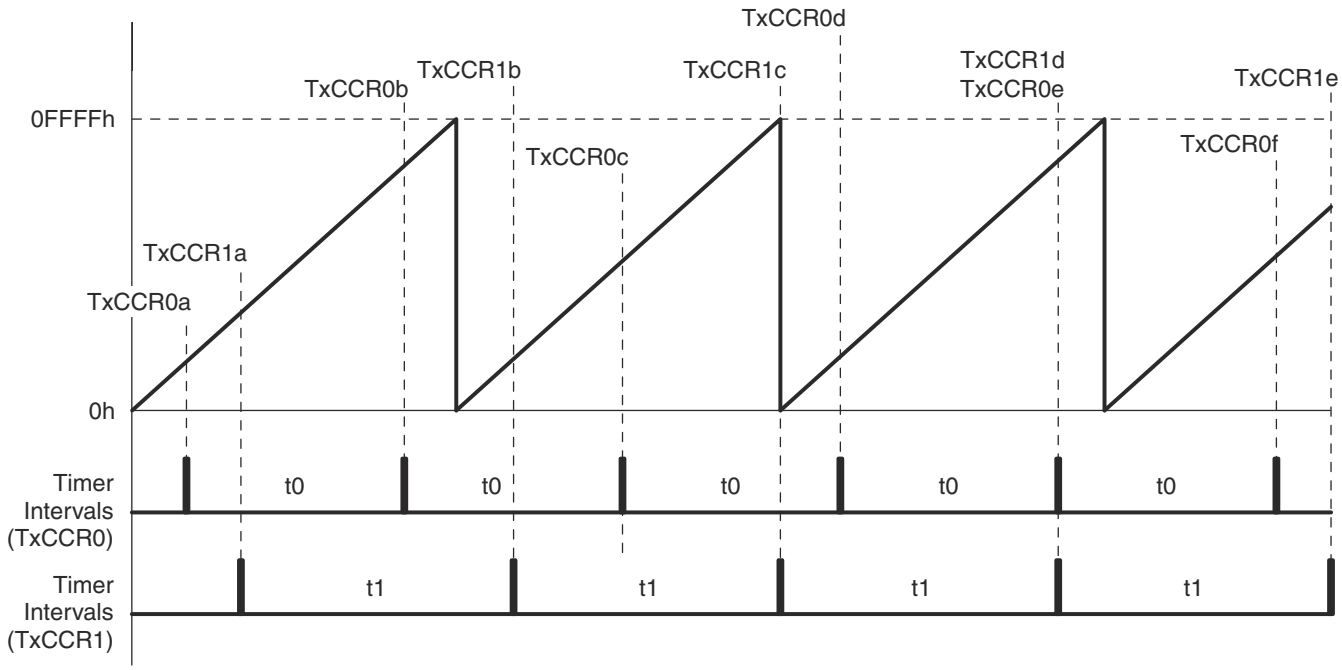


图 2-1. 连续模式时间间隔

使用多时基方法时，可在特定 MSP430 器件上同时生成的频率和占空比数量取决于器件上的 TxCCRx 寄存器总数。

继续以前面采用 MSP430F5529 器件的示例来说，可用计时器包括具有五个捕获/比较（CC）寄存器的 TA0、具有三个 CC 寄存器的 TA1、具有三个 CC 寄存器的 TA2，以及具有七个 CC 寄存器的 TB0。因此，在 MSP430F5529 上使用多时基方法时：

频率和占空比数量 = 5 CC + 3 CC + 3 CC + 7 CC = 18 个频率和占空比

使用单时基方法时，具有 14 个不同占空比的四个频率是唯一可用的选项。使用多时基方法时，则可以使用 18 个可用频率和占空比的任何组合。

3 在定制应用中实现多时基方法

以下各部分逐步介绍了如何在单个 MSP430 计时器模块上实现多个时基。示例代码可以在以下链接的 zip 文件中找到：https://www.ti.com/cn/lit/zip/slaa513.

3.1 计时器时钟源选择

根据应用中所需的时基数量，必须选择合适的计时器时钟源。具体选择取决于多个因素：所需频率、要实现的信号数量以及所需的分辨率。

时钟源的频率必须高于所需的输出频率。要在单个计时器上实现的信号越多，时钟源频率与所需输出频率之比就要越大。如需生成多个频率所需的最小时钟源频率方面的进一步指导，请参阅节 5 中的数据。

如果可能，最好选择频率为所需时基整数倍的时钟。例如，要从 ACLK 上的 32.768kHz 晶体生成一个 1kHz 时基，周期将为 32.768kHz / 1kHz = 32.768。不过，计数只能为整数，因此周期实际上将为 33。这意味着，生成的时基实际上频率为 32.768kHz / 33 ≈ 0.993kHz，引入了少量的误差。而如果时钟源为 1MHz DCO，则周期将为 1MHz / 1kHz = 1000 个。这不会引入任何额外的误差，因为时钟源频率是所需时基的整数倍。不过，使用不太准确的时钟源会引入额外的误差。通常，由舍入而导致的误差相对较小，并且随着时钟源的频率增加，舍入误差会减小。用户应衡量使用较高频率的时钟对功耗的影响。

如果要生成具有可变占空比的 PWM（例如在电机控制或 PWM DAC 应用中），那么需要考虑计时器的分辨率。分辨率由构成一个周期的时钟周期数量决定。例如，当计时器由 ACLK 上的 32.768kHz 晶体提供时钟信号时，要从该计时器生成一个 1kHz 时基，周期将为 32.768kHz / 1kHz = 33 个。这意味着，PWM 占空比存在 33 种不同的可能设置，因此占空比能够以 1 / 33 = 3.03% 占空比为步长而变化。如需获得更高的粒度，可以改为采用以 1MHz 运行的 DCO 为计时器提供时钟。现在，生成一个 1kHz 频率的周期为 1MHz / 1kHz = 1000 个。这意味着，PWM 占空比存在 1000 种不同的可能设置，因此占空比能够以 1 / 1000 = 0.1% 占空比为步长而变化。用户应衡量使用较高频率的时钟对功耗的影响，以及其应用的 PWM 分辨率要求。

$$\text{Duty cycle step size} = \frac{1}{n_{period}} \times 100\,\% \tag{1}$$

3.2 周期和频率计算

对于应用中需要的每个时基，用户需要确定用于构成所需周期的计时器计数（n_{period}）。这可以通过将基于时钟源频率（f_{source}）和 IDx 分压器位的计时器时钟频率（f_{timer}）除以所需的时基频率（$f_{desired}$）得出（请参阅方程式 2）。

$$f_{timer} = \frac{f_{source}}{IDx}$$

$$n_{period} = \frac{f_{timer}}{f_{desired}} \quad \text{(round to a whole number)} \tag{2}$$

3.3 占空比计算

要在计时器模块上生成 PWM，应使用另一个计数来设置占空比。生成的占空比为高电平时间的计时器计数（n_{high}）与一个周期的计时器计数（n_{period}）之比。对于多时基方法，低电平时间的计时器计数（n_{low}）和高电平时间的计时器计数（n_{high}）是添加到 ISR 中 TxCCRx 寄存器的偏置（请参阅方程式 3）。

$$n_{high} = \frac{(\text{Duty cycle \%})}{100\,\%} \times n_{period} \quad \text{(round to a whole number)}$$

$$n_{low} = n_{period} - n_{high} \tag{3}$$

图 3-1 显示了在单个 MSP430 计时器模块中实现两个单独的 PWM 频率和占空比时如何使用 TxCCRx 值以及 n_{high} 和 n_{low}。将此图与图 2-1 进行比较可以看到，为 TxCCR0 和 TxCCR1 增加了一个常数值，以生成 50% 占空比。

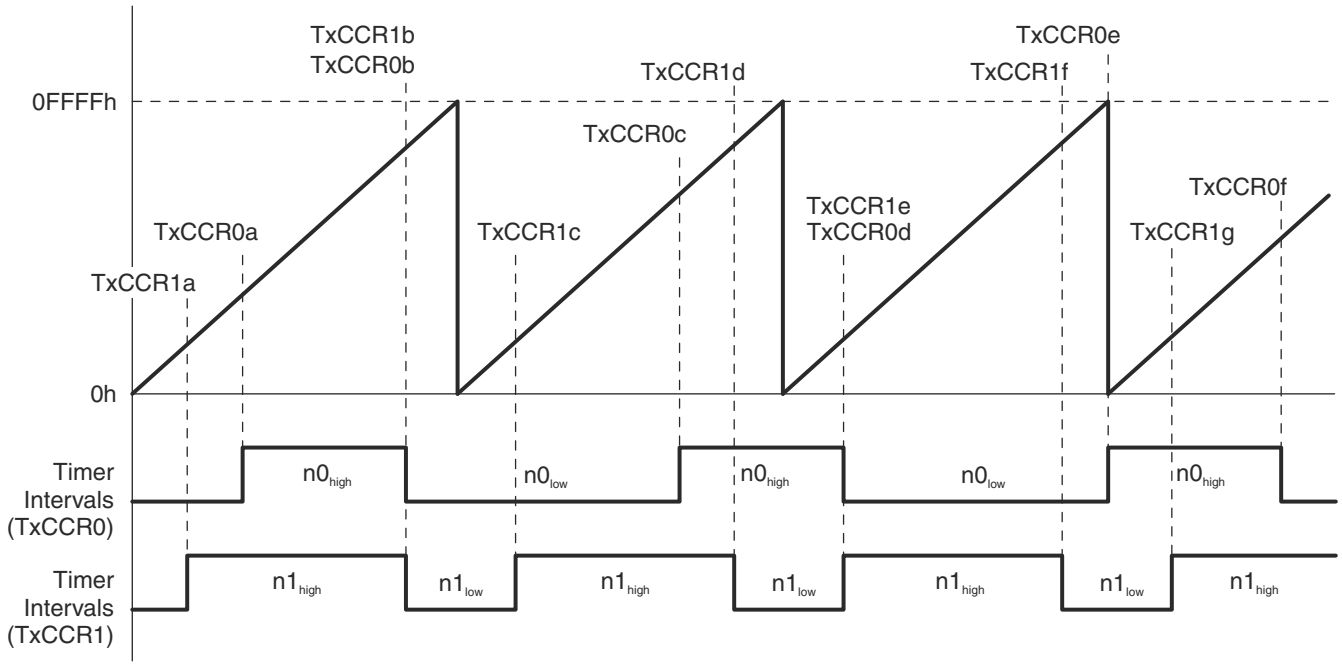


图 3-1. 连续模式 PWM 生成

4 示例代码

https://www.ti.com.cn/lit/zip/slaa513 上提供的 zip 文件包含用于在 MSP430G2452 器件的 Timer_A 模块上以及 MSP430F5529 器件的 Timer_B 模块上实现多个时基的示例代码。测试时使用了相同代码的不同变体来生成数据（请参阅节 5）。

4.1 方法

除了设置时钟、引脚和其他标准 MSP430 初始化外，还需要针对多时基方法实现两个主要代码段：计时器初始化和计时器 ISR。不管用户尝试实现的频率如何，计时器初始化都基本相同。以下代码显示了 multi_freq_g2452_example.c 文件中的计时器初始化：

```
  TACCTL0 = OUTMOD_4 + CCIE;                // CCR0 toggle, interrupt enabled
  TACCTL1 = OUTMOD_4 + CCIE;                // CCR1 toggle, interrupt enabled
  TACCTL2 = OUTMOD_4 + CCIE;                // CCR2 toggle, interrupt enabled
  TACTL = TASSEL_2 +  MC_2 + TAIE;          // SMCLK, Contmode, int enabled
```

根据具体的应用，此初始化过程中唯一要做出的更改是使用的 TxCCTLx 寄存器数量（应根据所需的输出频率数量来设置）和 TxCTL 寄存器中的 TxSSELx 位（应根据所需的计时器时钟源来设置）。

如果除了多个频率外还需要多个占空比，那么还需要对 TxCCTLx 寄存器进行一项设置：必须设置 CCISx 位。这是因为用于生成多个频率和占空比的 ISR（请参阅节 4.1.2）使用 CCI 位来读回引脚上的信号，从而确定输出当前处于高电平还是低电平阶段。CCISx 位选择哪个信号或引脚会连接到该 TxCCTLx 寄存器对应的内部 CCI 信号 – 连接到 CCIxA 或 CCIxB 的任一引脚或信号。对于此实现，用户必须选择用于输出 PWM 的 GPIO 引脚所对应的 CCISx 设置。有关器件上每个计时器模块的计时器信号连接信息，请参阅具体器件的数据表。以下代码显示了 multi_freq_pwm_g2452_example.c 文件中的计时器初始化：

```
  TACCTL0 = CCIS_0 + OUTMOD_4 + CCIE;       // CCR0 toggle, interrupt enabled, CCI0A input
  TACCTL1 = CCIS_0 + OUTMOD_4 + CCIE;       // CCR1 toggle, interrupt enabled, CCI1A input
  TACCTL2 = CCIS_0 + OUTMOD_4 + CCIE;       // CCR2 toggle, interrupt enabled, CCI2A input
  TACTL = TASSEL_2 +  MC_2 + TAIE;          // SMCLK, Contmode, int enabled
```

4.1.1 用于生成多个频率的 ISR

要在单个计时器模块上实现多个频率，计时器 ISR 中只需确定触发中断的 TxCCRx 计数并将相应周期一半的计时器计数加到当前 TxCCRx 值上。之所以加上该周期的一半，是因为每次计数达到 TxCCRx 值，输出都会进行切换，因此需要经过两次中断，一个完整周期才会结束。以下代码显示了 multi_freq_g2452_example.c 文件中的计时器 ISR：

```
// Timer_A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0 (void)
{
  TACCR0 += 100;                            // reload period
}
// Timer_A1 Interrupt Vector (TA0IV) handler
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A1(void)
{
  switch( TA0IV )
  {
  case  2:  TACCR1 += 200;                  // reload period
            break;
  case  4:  TACCR2 += 500;                  // reload period
            break;
  case 10: P1OUT ^= 0x01;                   // Timer overflow
            break;
  default: break;
  }
}
```

此示例显示了从 1MHz 计时器时钟源生成 5kHz、2.5kHz 和 1kHz 信号的 ISR（请参阅方程式 4）。

$$f_0 = \frac{1MHz}{(2 \times 100)} = 5\,kHz$$

$$f_1 = \frac{1MHz}{(2 \times 200)} = 2.5\,kHz$$

$$f_2 = \frac{1MHz}{(2 \times 500)} = 1kHz \quad (4)$$

4.1.2 用于生成多个频率和占空比（PWM）的 ISR

要在单个计时器模块上实现多个占空比以及多个频率，计时器 ISR 会变得稍微复杂一些。在确定触发中断的 TxCCRx 计数后，我们还必须确定输出是进入信号输出的高电平还是低电平部分，以便判断是否要增加高电平时间或低电平时间对应的计数。这可以通过检查相应 TxCCTL0 寄存器中的 CCI 位来实现。如果 CCI 位为 1，那么我们便知道输出刚刚切换至高电平，信号位于高电平周期的开始处，因此我们必须增加高电平时间对应的计数。相反，如果 CCI 位为 0，则表示输出刚刚切换至低电平，因此我们必须增加低电平时间对应的计数。高电平时间和低电平时间所对应计数之和决定信号的周期，而高电平时间和低电平时间与周期值之比则决定占空比。以下代码显示了 multi_freq_g2452_pwm_example.c 文件中的计时器 ISR：

```
// Timer_A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0 (void)
{
    if(TACCTL0 & CCI)                          // If output currently high
    {
        TACCR0 += 50;                          // 25% high
    }
    else
    {
        TACCR0 += 150;                         // 75% low
    }
}
// Timer_A1 Interrupt Vector (TA0IV) handler
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A1(void)
{
  switch( TA0IV )
  {
  case  2:  if(TACCTL1 & CCI)                  // If output currently high
            {
                TACCR1 += 50;                  // 12.5% high
            }
            else
            {
                TACCR1 += 350;                 // 87.5% low
            }
           break;
  case  4:  if(TACCTL2 & CCI)                  // If output currently high
            {
                TACCR2 += 600;                 // 60% high
            }
            else
            {
                TACCR2 += 400;                 // 40% low
            }
           break;
  case 10: P1OUT ^= 0x01;                    // Timer overflow
           break;
  default: break;
  }
}
```

本例中，代码会从 1MHz 计时器时钟源生成多个 5kHz、2.5kHz 和 1kHz 的相同频率信号。不过，这时信号的占空比分别为 25%、12.5% 和 60%（请参阅方程式 5）。

$$f_0 = \frac{1\ \text{MHz}}{(50 + 150)} = 5\ \text{kHz} \qquad \text{DutyCycle}_0 = \frac{50}{(50 + 150)} \times 100\% = 25\%$$

$$f_1 = \frac{1\ \text{MHz}}{(50 + 350)} = 2.5\ \text{kHz} \qquad \text{DutyCycle}_1 = \frac{50}{(50 + 350)} \times 100\% = 12.5\%$$

$$f_2 = \frac{1\ \text{MHz}}{(600 + 400)} = 1\ \text{kHz} \qquad \text{DutyCycle}_2 = \frac{600}{(600 + 400)} \times 100\% = 60\% \tag{5}$$

4.2 包含的代码示例

- multi_freq_g2452_example.c – 用于在 Timer_A 模块上生成三个不同频率的 MSP430G2452 示例。
- multi_freq_pwm_g2452_example.c – 用于以不同的频率和占空比在 Timer_A 模块上生成三个不同 PWM 的 MSP430G2452 示例。
- multi_freq_f5529_example.c – 用于在 Timer_B 模块上生成七个不同频率的 MSP430F5529 示例。将 5xx/6xx 核心软件库用于时钟配置。输出会映射到端口 4，以便可在 MSP-EXP430F5529 实验板上访问相关信号。
- multi_freq_pwm_f5529_example.c – 用于以不同的频率和占空比在 Timer_B 模块上生成七个不同 PWM 的 MSP430F5529 示例。将 5xx/6xx 核心软件库用于时钟配置。输出会映射到端口 4，以便可在 MSP-EXP430F5529 实验板上访问相关信号。

5 多时基方法的限制

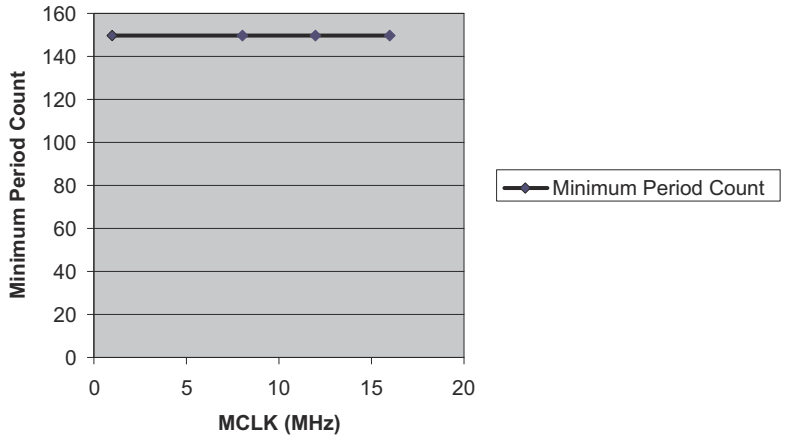
5.1 ISR 开销

在用于生成 PWM 的单时基方法中，无需重新载入 TxCCRx 寄存器，并且甚至无需进入 ISR，便会自动生成输出。这意味着，在初始计时器配置后，不存在软件开销，因为一切工作均由硬件处理。在多时基方法中，每个高电平周期和低电平周期结束时都会进入 ISR，因此会增加一些软件开销。

虽然代码示例中的 ISR 都保持长度尽可能最小，但增加软件开销的所需操作包括：

- 从低功耗模式退出时的唤醒时间
- ISR 进入时间
- 用于确定哪个 TxCCRx 触发中断的判定逻辑
- 用于确定哪个周期部分（高电平/低电平）刚刚结束的判定逻辑（仅实现 PWM 时）
- 通过添加合适的周期值来重新载入 TxCCRx 寄存器

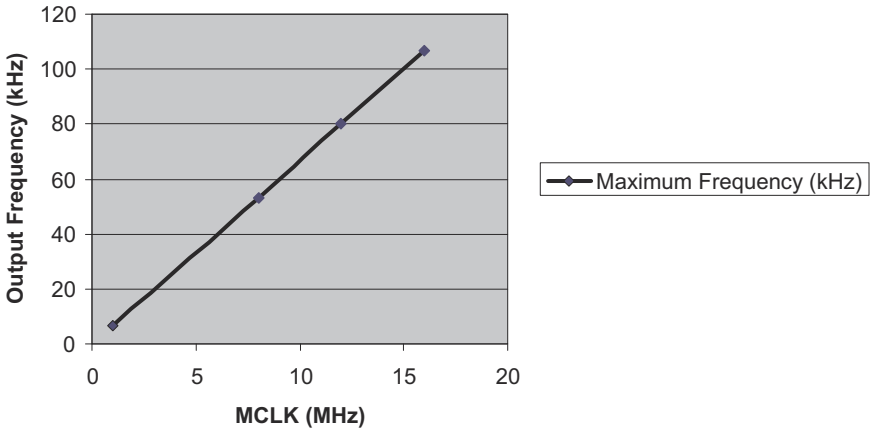
生成的每个信号都会增加此开销，因为这会增加 ISR 中所用的时间百分比。如果 ISR 中使用的周期数相比于 ISR 间的周期数（由用户的周期计数值设置）来说太大，进而导致无法及时处理中断，便会出现问题。因此，这里存在一个截止点，因 ISR 代码所增加延迟而使用的 TACCRx 寄存器数量而异。对于待生成的特定数量的信号，则当计时器时钟由 MCLK 提供时，从周期数量方面来看，此截止点会在 MCLK 频率上基本保持不变。这是因为该值表示 ISR 周期数量与中断间周期数量之比。图 5-1 显示了生成可靠信号的最小周期数量与 MCLK 之间的固定关系。这转而导致最大输出频率与 MCLK 之间存在线性关系，如图 5-2 所示。



A. 图 5-1 中的数据由 MSP430G2452 上的测试生成，该项测试用于生成三个具有相同周期的信号（“最差情况”）。
B. 图 5-1 和图 5-2 中的数据点也可以在表 5-1 中找到。

C. 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。

图 5-1. 最小周期计数与 MCLK 频率



A. 图 5-2 中的数据由 MSP430G2452 上的测试生成，该项测试用于生成三个具有相同周期的信号（“最差情况”）。

B. 图 5-1 和图 5-2 中的数据点也可以在表 5-1 中找到。

C. 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。

图 5-2. 最大输出电压与 MCLK 频率

表 5-1. MSP430G2452 连续模式性能(1)(2)

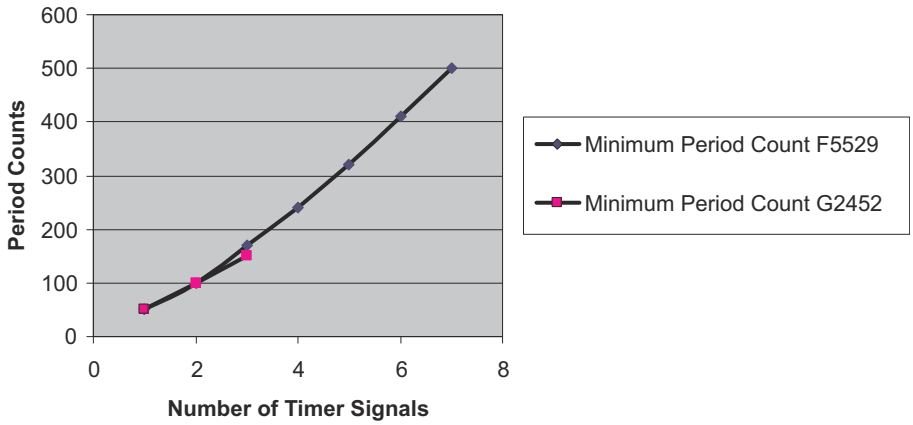
MCLK（MHz）	最小周期计数	最大频率（kHz）
1	150	6.67
8	150	53.33
12	150	80
16	150	106.67

(1) 表 5-1 中的数据由 MSP430G2452 上的测试生成，该项测试用于生成三个具有相同周期的信号（“最差情况”）。

(2) 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。

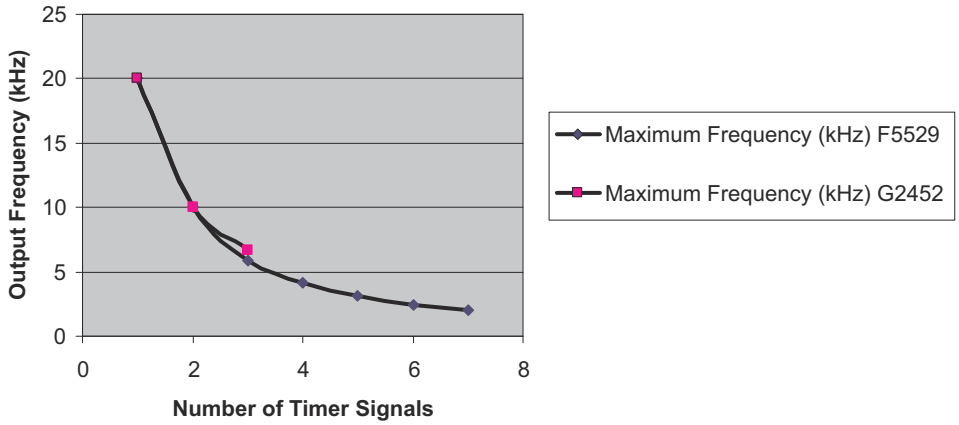
5.2 最大输出频率与信号数量

要生成的输出信号数量会显著影响可使用连续模式可靠生成的最大频率。这是因为在必须处理额外的信号时，ISR 时间会有所增加。图 5-3 和图 5-4 显示了周期计数和生成的最大输出频率与采用 1MHz MCLK 在 MSP430F5529 的 Timer_B 模块上以及 MSP430G2452 的 Timer_A 模块上实现的计时器信号数量之间的关系。两个计时器模块上的数据非常相似，因此对于大多数 MSP430 器件，可以使用相同的值作为通用指南。如图 5-1 所示，最小周期计数与 MCLK 频率无关，因此对于任何 MCLK 频率，都可以使用相同的数据作为指南。



A. 图 5-3 中的数据由 MSP430G2452 和 MSP430F5529 上的测试生成，该项测试用于生成具有相同周期的信号。
B. 图 5-3 和图 5-4 中的数据点也可以在表 5-2 中找到。
C. 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。

图 5-3. 最小周期计数与计时器信号数量



A. 图 5-4 中的数据由 MSP430G2452 和 MSP430F5529 上的测试生成，该项测试用于生成具有相同周期的信号。
B. 图 5-3 和图 5-4 中的数据点也可以在表 5-2 中找到。
C. 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。
D. 输出频率基于 1MHz MCLK。最大输出频率与 MCLK 直接成比例。

图 5-4. 最大输出频率与计时器信号数量

表 5-2. 最大输出频率(1)(2)

计时器信号数量	MSP430F5529 Timer_B		MSP430G2452 Timer_A	
	最小周期计数	最大频率(3) (kHz)	最小周期计数	最大频率(3) (kHz)
1	50	20	50	20
2	100	10	100	10
3	170	5.882	150	6.667
4	240	4.167		
5	320	3.125		
6	410	2.439		
7	500	2		

(1) 表 5-2 中的数据由 MSP430G2452 和 MSP430F5529 上的测试生成，该项测试用于生成具有相同周期的信号。

(2) 此数据仅作为通用指南，供了解生成所需输出频率时需要的计时器源频率，而不应视为数据表规范。系统中的其他中断、ISR 代码的结构或者不同的频率和占空比组合等其他因素可能会影响特定应用中的相应结果。

(3) 1MHz MCLK 时的最大频率。最大频率与 MCLK 直接成比例。例如，16MHz MCLK 时的最大频率 = 16 x 1MHz MCLK 时的最大频率。

5.3 功耗

在单个计时器模块上实现多个频率时，电流消耗可能会高于使用多个计时器模块的情况。这是因为多时基方法要求使用计时器中断，该中断会定期唤醒 CPU 来提供服务，而单时基方法可以完全在硬件中运行。这意味着，使用多时基方法时，CPU 会在更大比例的时间内处于唤醒状态，而不是处于低功耗模式。这与节 5.2 中的表格相关，那些表格显示了每增加一个计时器信号时，以及随着计时器信号频率接近计时器时钟源频率，处于工作模式的时间比例（ISR 时间）会增加。这转而会导致平均功耗增加。

6 参考文献

1. *MSP430G2x52、MSP430G2x12 混合信号微控制器 数据表*
2. *MSP430x2xx 系列用户指南*
3. *MSP430F551x、MSP430F552x 混合信号微控制器 数据表*
4. *MSP430x5xx 和 MSP430x6xx 系列用户指南*
5. *MSP430F5xx 和 MSP430F6xx 核心软件库*

7 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision A (March 2015) to Revision B (February 2022)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式	1
• 更正了节 3 *在定制应用中实现多时基方法* 中的软件下载链接	4

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265