Application Note

Multiple Time Bases on a Single MSP430™ Timer Module

Katie Pier *MSP430 Applications*

ABSTRACT

The timer modules on MSP430™ ultra-low power microcontrollers often base several different outputs off of a single time base – a single timer period. This is especially true for the typical implementation of pulse width modulation (PWM) signals on the MSP430 devices, where one capture compare register (TxCCR0) sets the period, and the rest (TxCCRx) simply set different duty cycles. However, in some applications, multiple time bases are needed to generate multiple output frequencies. Normally this is done using multiple timer modules, but this might require upgrading to a part with many more extra features that may not be needed for the application. However, in exchange for a small amount of software overhead, multiple time bases can be implemented on a single MSP430 timer module, allowing for more features to be implemented on a simpler MSP430 device. This potentially can reduce cost and board space by allowing a smaller MSP430 device to be used and, on large MSP430 devices, allows for new applications requiring a large number of different output frequencies to be implemented.

The source code and other files described in this application report can be downloaded from https://www.ti.com/lit/zip/slaa513.

Table of Contents

Trademarks

MSP430™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Typical Single Time Base Method

In some applications, it is necessary to generate multiple signals with unique frequencies simultaneously. In typical use of an MSP430 MCU, Up Mode or Up/Down Mode is used on the timer modules with each frequency generated corresponding to a unique timer module. Capture Compare Register 0 (TxCCR0) for each timer module is set at the beginning of the program to set the period of the timer. Additional Capture Compare Registers (TxCCRx) for each module can be set to a value at the beginning of the program to generate different duty cycles, but because the timer only counts up to the value in TxCCR0, everything on the timer module is done in reference to this same timer period. In this method, everything is set at the beginning of timer operation, and the settings are left constant from then on – everything is done in hardware, and there are no values that need to be reloaded in the ISR. Figure 1-1 shows the relationship of TxCCR0 value and period value t0 in Up Mode.

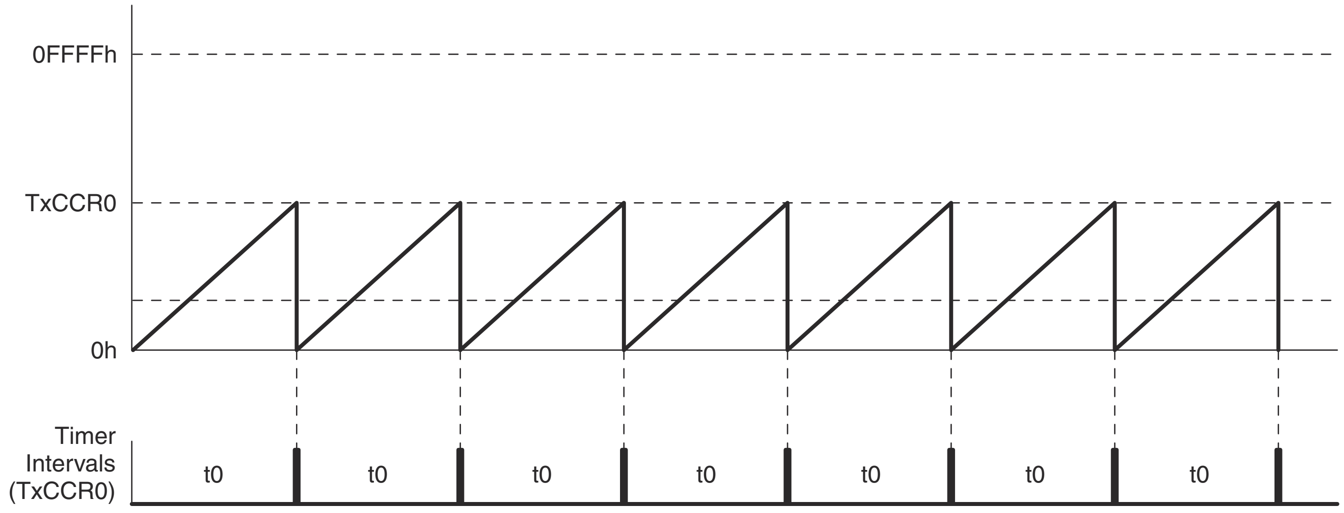


Figure 1-1. Up Mode Time Intervals

When using the single time base method, the number of frequencies that can be simultaneously produced on a particular MSP430 device is dependent on the number of timer modules on the MSP430 device, and the number of possible simultaneous duty cycles is dependent on the number of TxCCRx registers on each module minus 1.

For example, an MSP430F5529 device features TA0 with five capture/compare (CC) registers, TA1 with three CC registers, TA2 with three CC registers, and TB0 with seven CC registers. Therefore, using the single time base method on an MSP430F5529:

Number of Frequencies = 3 Timer_A modules + 1 Timer_B module = 4 Independent Frequencies

Number of Duty Cycles TA0 = 5 CC – 1 = 4 Duty Cycles at TA0 frequency

Number of Duty Cycles TA1 = 3 CC – 1 = 2 Duty Cycles at TA1 frequency

Number of Duty Cycles TA2 = 3 CC – 1 = 2 Duty Cycles at TA2 frequency

Number of Duty Cycles TB0 = 7 CC – 1 = 6 Duty Cycles at TB0 frequency

2 Multiple Time Base Method

It is possible to implement multiple timer periods on the same timer module by using Continuous Mode. In Continuous Mode, the timer always count up to 0xFFFF and then rolls over, rather than resetting at TxCCR0 each time. This means that the period is no longer set by placing a constant value in TxCCR0; rather, it is the difference between the previous and next value of TxCCRx that sets the period *for each capture compare register*. For example, this means that in the code, instead of setting TACCR0 = 0xFF at the beginning of the program, we would need to dynamically change TxCCR0 by 0xFF every time the counter reaches TxCCR0 (“TACCR0 += 0xFF”). Because each TxCCRx register is now completely independent from TxCCR0, a different frequency can be generated for each TxCCRx register on each timer module, drastically increasing the number of frequencies that can be produced. Figure 2-1 shows the relationship of the TxCCR0 value and two independent periods t0 and t1 in Continuous Mode. t0 corresponds to a count that is added to TxCCR0 at each interrupt, and t1 corresponds to a count that is added to TxCCR1 at each interrupt, creating the two different periods.

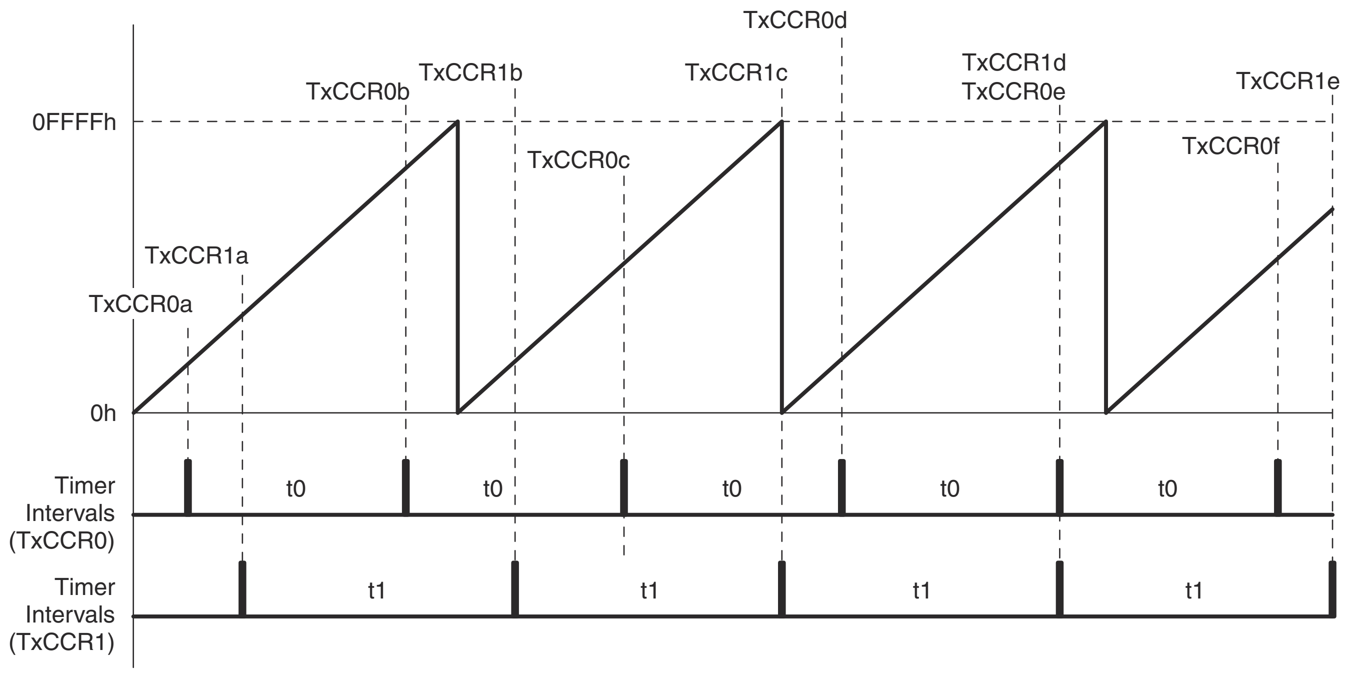


Figure 2-1. Continuous Mode Time Intervals

When using the multiple time base method, the number of frequencies and duty cycles that can be simultaneously produced on a particular MSP430 device is dependent on the total number of all TxCCRx registers on the device.

Continuing with the previous example using the MSP430F5529 device, the available timers are TA0 with five capture/compare (CC) registers, TA1 with three CC registers, TA2 with three CC registers, and TB0 with seven CC registers. Therefore, using the multiple time base method on an MSP430F5529:

Number of Frequencies and Duty Cycles = 5 CC + 3 CC + 3 CC + 7 CC = 18 frequencies and duty cycles

With the single time base method four frequencies with 14 varying duty cycles was the only option available. With the multiple time base method, any combination of the available 18 frequencies and duty cycles is available.

3 Implementing the Multiple Time Base Method in a Custom Application

The following sections lay out step by step how to implement multiple time bases on a single MSP430 timer module. Example code can be found in the zip file at this link: https://www.ti.com/lit/zip/slaa513.

3.1 Timer Clock Source Selection

Depending on the time bases that are required in the application, an appropriate timer clock source must be selected. The selection depends on several factors – the frequency required, the number of signals being implemented, and the desired resolution.

The clock source must be of a higher frequency than the desired output frequencies. The more signals being implemented on a single timer, the larger the ratio of clock source frequency to desired output frequency needs to be. For further guidance on the minimum clock source frequency needed to generate multiple frequencies, see the data in Section 5.

When possible, it is best to choose a clock whose frequency is a multiple of the desired time base. For example, to generate a 1-kHz time base from a 32.768-kHz crystal on ACLK, the period would be 32.768 kHz / 1 kHz = 32.768. However, the count can only be a whole number, so the period would actually be 33. This means that the generated time base actually has a frequency of 32.768 kHz / 33 ≈ 0.993 kHz, introducing a small amount of error. If the clock source were a 1-MHz DCO, on the other hand, the period would be 1 MHz / 1 kHz = 1000 counts. This does not introduce any additional error, because the clock source frequency is a multiple of the desired time base. Additional error can be introduced, however, from using less accurate clock sources. In general, the amount of error introduced from rounding is relatively low, and the rounding error decreases as the frequency of the source clock increases. Users should weigh the impact of using a higher frequency clock against the power consumption.

If a PWM with a variable duty cycle is going to be produced (as in a motor control or PWM DAC application), then the resolution of the timer needs to be considered. The resolution is determined by the number of clock ticks that make up one period. For example, to generate a 1-kHz time base with the timer sourced from a 32.768-kHz crystal on ACLK, the period would be 32.768 kHz / 1 kHz = 33 counts. This means that there are 33 different possible settings for the PWM duty cycle, so the duty cycle can vary in steps of 1 / 33 = 3.03% duty cycle. For finer granularity, the timer can instead be sourced from the DCO running at 1 MHz. Now the period to generate a 1-kHz frequency is 1 MHz / 1 kHz = 1000 counts. This means that there are 1000 different possible settings for the PWM duty cycle, so the duty cycle can vary in steps of 1 / 1000 = 0.1% duty cycle. Users should weigh the impact of using a higher frequency clock on their power consumption versus their application's PWM resolution requirements.

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

3.2 Period and Frequency Calculation

For each time-base required in the application, you need to determine the number of timer counts (n_{period}) to make up the desired period. This can be found by taking the frequency of the timer clock (f_{timer}) based on the clock source frequency (f_{source}) and IDx divider bits, and dividing by the desired time base frequency ($f_{desired}$) (see Equation 2).

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

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

3.3 Duty Cycle Calculation

To generate a PWM on the timer module, a second count is used to set the duty cycle. The duty cycle generated is the ratio of the number of timer counts for the high time (n_{high}) to the number of timer counts for one period (n_{period}). For the multiple time base method, the timer counts for the low (n_{low}) and high (n_{high}) time are the offsets that are added into the TxCCRx register in the ISR (see Equation 3).

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

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

Figure 3-1 shows how the TxCCRx values and n_{high} and n_{low} are used when implementing two independent PWM frequencies and duty cycles on a single MSP430 timer module. Compare this to Figure 2-1, where constant values were added to TxCCR0 and TxCCR1 to produce a 50% duty cycle.

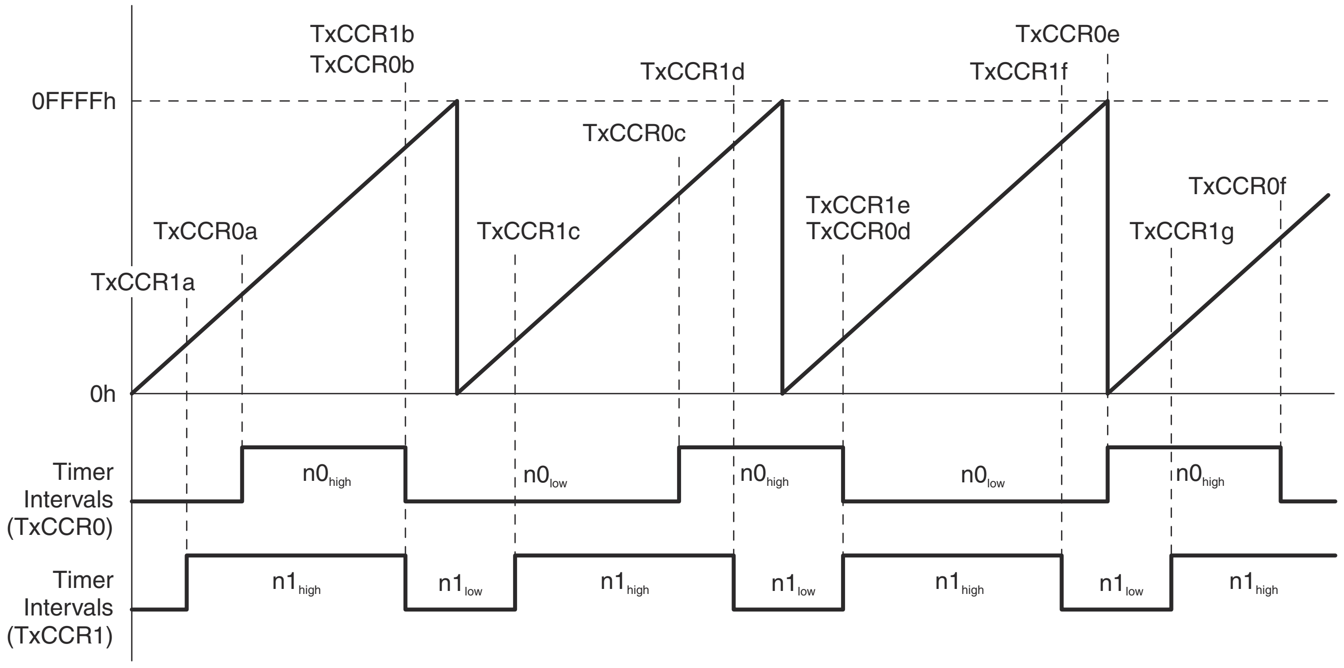


Figure 3-1. Continuous Mode PWM Generation

4 Example Code

The zip file at https://www.ti.com/lit/zip/slaa513 contains example code for implementing multiple time bases on a Timer_A module on an MSP430G2452 device and a Timer_B module on an MSP430F5529 device. Variations of the same code were used in testing to produce the data found in Section 5.

4.1 Method

In addition to setting up clocks, pins, and other standard MSP430 initialization, there are two main pieces of code that need to be implemented for the multiple time base method – the timer initialization and the timer ISRs. The timer initialization is essentially the same no matter what frequencies you are trying to implement. The following code shows the timer initialization from the file multi_freq_g2452_example.c:

```
  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
```

The only things that change in this initialization process, depending on the particular application, are the number of TxCCTLx registers used (which should be set according to the desired number of output frequencies) and the TxSSELx bits in the TxCTL register (which should be set according to the desired timer clock source).

If multiple duty cycles are desired in addition to multiple frequencies, then one more setting is needed for the TxCCTLx registers: the CCISx bits must be set. This is because the ISR for multiple frequencies and duty cycles (see Section 4.1.2) uses the CCI bit to read back the signal on the pin to determine if the output is currently in the high or low phase. The CCISx bits select which signal or pin is connected to the internal CCI signal for that TxCCTLx register – either the pin or signal connected to CCIxA or CCIxB. For this implementation, the user must select the CCISx setting that corresponds to the GPIO pin where they are outputting the PWM. Information about the timer signal connections for each timer module on the device is provided in the device-specific datasheet. The following code shows the timer initialization from the file multi_freq_pwm_g2452_example.c:

```
  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 ISR for Multiple Frequencies

For implementing multiple frequencies on a single timer module, all that is needed in the timer ISRs is to determine which TxCCRx count triggered the interrupt and to add the timer counts for one half of the corresponding period to the current TxCCRx value. One half of the period is added because the output is toggling each time the count reaches the TxCCRx value, so it takes two interrupts before one full period has passed. The following code shows the timer ISRs from the file multi_freq_g2452_example.c:

```
// 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;
  }
}
```

This example shows the ISRs for generating 5-kHz, 2.5-kHz, and 1-kHz signals from a 1-MHz timer clock source (see Equation 4).

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

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

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

4.1.2 ISR for Multiple Frequencies and Duty Cycles (PWM)

For implementing multiple duty cycles as well as frequencies on a single timer module, the timer ISRs become slightly more complex. After determining which TxCCRx count triggered the interrupt, we must also determine whether the output is entering the high or low portion of the signal output so that we know whether to add the count corresponding to the high or low time. This can be done by checking the CCI bit in the corresponding TxCCTL0 register. If the CCI bit is 1, then we know that the output just toggled to high, the signal is at the beginning of the high cycle, so we must add the count corresponding to the high time. Conversely, if the CCI bit is 0, the output just toggled to low, and we must add the count corresponding to the low time. The sum of the counts corresponding to the high and low times determines the period of the signal, and the ratio of the high and low times to the period value determines the duty cycle. The following code shows the timer ISRs from the file multi_freq_g2452_pwm_example.c:

```
// 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;
  }
}
```

In this case, the code generates the same frequencies of 5-kHz, 2.5-kHz, and 1-kHz signals from a 1-MHz timer clock source. However, this time the signals have 25%, 12.5%, and 60% duty cycles respectively (see Equation 5).

$$
\begin{aligned}
f_0 &= \frac{1\text{ MHz}}{(50+150)} = 5\text{ kHz} & \quad \text{DutyCycle}_0 &= \frac{50}{(50+150)} \times 100\% = 25\% \\
f_1 &= \frac{1\text{ MHz}}{(50+350)} = 2.5\text{ kHz} & \quad \text{DutyCycle}_1 &= \frac{50}{(50+350)} \times 100\% = 12.5\% \\
f_2 &= \frac{1\text{ MHz}}{(600+400)} = 1\text{ kHz} & \quad \text{DutyCycle}_2 &= \frac{600}{(600+400)} \times 100\% = 60\%
\end{aligned} \tag{5}
$$

4.2 Included Code Examples

- multi_freq_g2452_example.c – MSP430G2452 example generating three independent frequencies on a Timer_A module.
- multi_freq_pwm_g2452_example.c – MSP430G2452 example generating three independent PWMs with different frequencies and duty cycles on a Timer_A module.
- multi_freq_f5529_example.c – MSP430F5529 example generating seven independent frequencies on a Timer_B module. Makes use of the 5xx/6xx Core Libraries for clock configuration. Outputs are port mapped to Port 4 so signals are accessible on an MSP-EXP430F5529 Experimenter Board.
- multi_freq_pwm_f5529_example.c – MSP430F5529 example generating seven independent PWMs with different frequencies and duty cycles on a Timer_B module. Makes use of the 5xx/6xx Core Libraries for clock configuration. Outputs are port mapped to Port 4 so signals are accessible on an MSP-EXP430F5529 Experimenter Board.

5 Limitations of the Multiple Time Base Method

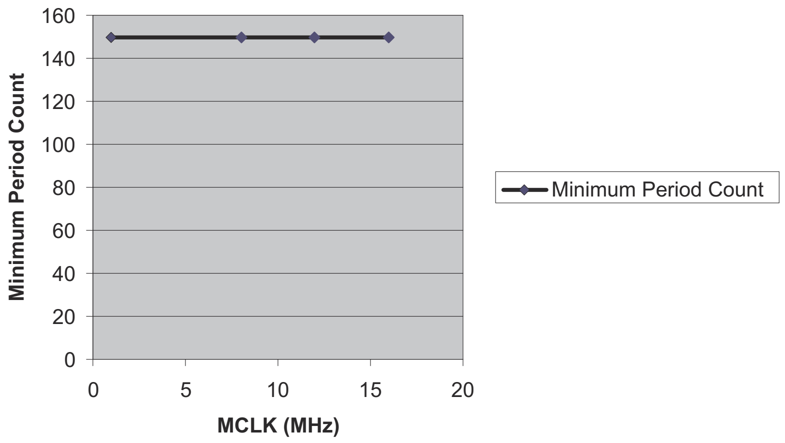
5.1 ISR Overhead

In the single time base method of producing PWMs, the TxCCRx registers do not need to be reloaded and the output is produced automatically without ever having to enter an ISR. This means that after the initial timer configuration there is no software overhead, because everything is handled in hardware. For the multiple time base method, an ISR is entered at the end of each high and low period, adding some software overhead.

While the ISRs in the code examples are kept to the smallest length possible, the required operations which add to the software overhead include:

- Wake-up time if coming from a low-power mode
- ISR entry time
- Decision logic for which TxCCRx triggered the interrupt
- Decision logic for what part of the cycle (high/low) just finished (when implementing PWMs only)
- Reloading the TxCCRx register by adding the appropriate period value

This overhead increases for each signal generated, as this adds to the percentage of time spent in the ISR. Problems occur if the number of cycles spent in the ISR is too great compared to the cycles between ISRs (which is set by your period count value), preventing interrupts from being serviced in a timely manner. Therefore, there is a cutoff point that varies depending on the number of TACCRx registers being used due to the latency added by the ISR code. For a particular number of signals being generated this cutoff point, in terms of number of cycles when the timer clock is being sourced from MCLK, remains essentially constant across MCLK frequencies. This is because the value is an indicator of the ratio between ISR cycles and cycles between interrupts. Figure 5-1 shows the constant relationship of the minimum period count yielding reliable signals versus MCLK. This results in turn in a linear relationship between the maximum output frequency and MCLK, as seen in Figure 5-2.

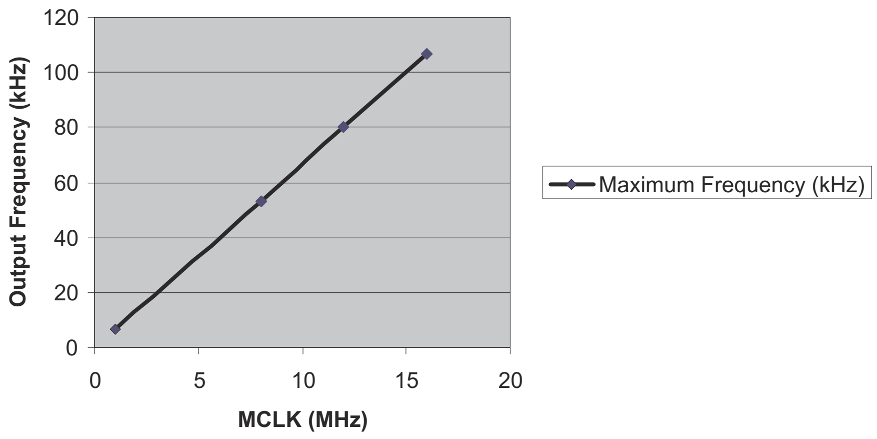


A. The data in Figure 5-1 was generated from testing on an MSP430G2452 producing three signals of the same period (“worst-case”).

B. The data points in Figure 5-1 and Figure 5-2 can also be found in Table 5-1.

C. This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.

Figure 5-1. Minimum Period Count vs MCLK Frequency



A. The data in Figure 5-2 was generated from testing on an MSP430G2452 producing three signals of the same period (“worst-case”).

B. The data points in Figure 5-1 and Figure 5-2 can also be found in Table 5-1.

C. This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.

Figure 5-2. Maximum Output Frequency vs MCLK Frequency

Table 5-1. MSP430G2452 Continuous Mode Performance(1)(2)

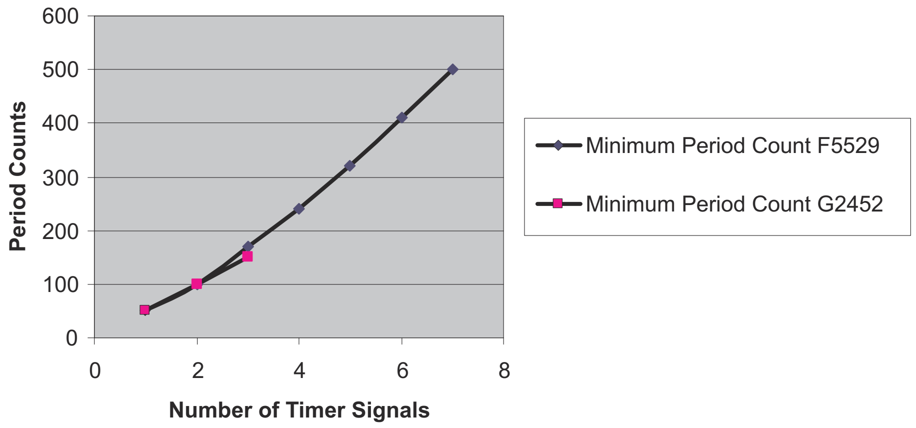
MCLK (MHz)	Minimum Period Count	Maximum Frequency (kHz)
1	150	6.67
8	150	53.33
12	150	80
16	150	106.67

(1) The data in Table 5-1 was generated from testing on an MSP430G2452 producing three signals of the same period (“worst-case”).

(2) This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.

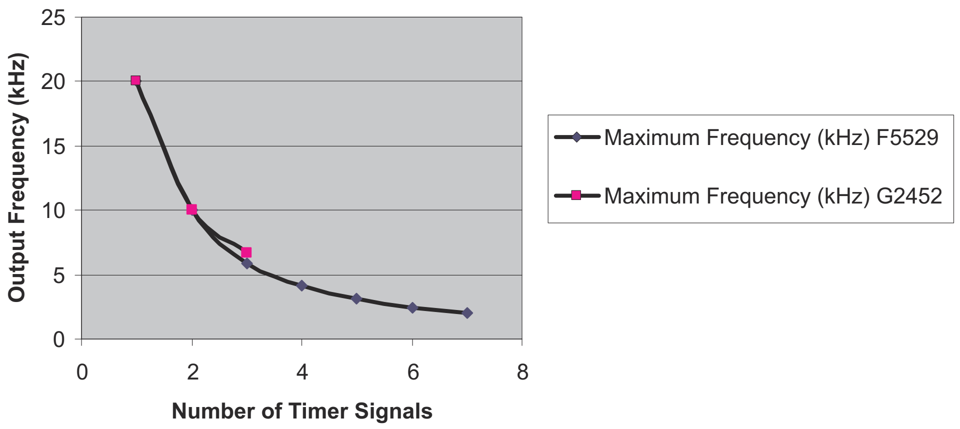
5.2 Maximum Output Frequency vs Number of Signals

The number of output signals being produced has a significant effect on the maximum frequencies that can be reliably produced using continuous mode. This is due to the increase in ISR time when an additional signal must be handled. Figure 5-3 and Figure 5-4 show the relationships between period count and the maximum output frequency produced versus the number of timer signals implemented on the Timer_B module of an MSP430F5529 and from the Timer_A module of an MSP430G2452 sourced from a 1-MHz MCLK. The data is quite similar for both timer modules, so the same values can be used as general guidelines for most MSP430 devices. As shown in Figure 5-1, the Minimum Period Count is independent of MCLK frequency, so the same data can be used as a guideline for any MCLK frequency.



A. The data in Figure 5-3 was generated from testing on an MSP430G2452 and an MSP430F5529 producing signals of the same period.
B. The data points in Figure 5-3 and Figure 5-4 can also be found in Table 5-2.
C. This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.

Figure 5-3. Minimum Period Count vs Number of Timer Signals



A. The data in Figure 5-4was generated from testing on an MSP430G2452 and an MSP430F5529 producing signals of the same period.
B. The data points in Figure 5-3 and Figure 5-4 can also be found in Table 5-2.
C. This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.
D. Output frequencies are based on a 1-MHz MCLK. The maximum output frequencies scale directly with MCLK.

Figure 5-4. Maximum Output Frequency vs Number of Timer Signals

Table 5-2. Maximum Output Frequency[1][2]

Number of Timer Signals	MSP430F5529 Timer_B		MSP430G2452 Timer_A	
	Minimum Period Count	Maximum Frequency[3] (kHz)	Minimum Period Count	Maximum Frequency[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) The data in Table 5-2 was generated from testing on an MSP430G2452 and an MSP430F5529 producing signals of the same period.
(2) This data is provided only as a general guideline for the required timer source frequency to produce the desired output frequencies, and should not be regarded as a data sheet specification. Other factors like the other interrupts in the system, the construction of the ISR code, or different frequency and duty cycle combinations may affect the results in a particular application.
(3) Maximum frequency with a 1-MHz MCLK. The maximum frequencies scale directly with MCLK. For example, maximum frequency at 16-MHz MCLK = 16 x maximum frequency at 1-MHz MCLK

5.3 Power Consumption

Implementing multiple frequencies on a single timer module can result in higher current consumption than using multiple timer modules. This is because the multiple time base method requires the use of timer interrupts that periodically wake the CPU for servicing, whereas the single-time base method can run completely in hardware. This means that when using the multiple time base method, the CPU is awake for a larger percentage of the time instead of staying in low-power mode. This is related to the tables in Section 5.2, which show that the percentage of time in active mode (ISR time) increases with each additional timer signal and as timer signal frequencies approach the timer clock source frequency. This in turn increases the average power consumption.

6 References

1. *MSP430G2x52, MSP430G2x12 Mixed-Signal Microcontrollers* data sheet
2. *MSP430x2xx Family User's Guide*
3. *MSP430F551x, MSP430F552x Mixed-Signal Microcontrollers* data sheet
4. *MSP430x5xx and MSP430x6xx Family User's Guide*
5. *MSP430F5xx and MSP430F6xx Core Libraries*

7 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision A (March 2015) to Revision B (February 2022) **Page**

- Updated the numbering format for tables, figures, and cross references throughout the document.................. 1
- Corrected the link to download software in Section 3 *Implementing the Multiple Time Base Method in a Custom Application* 4