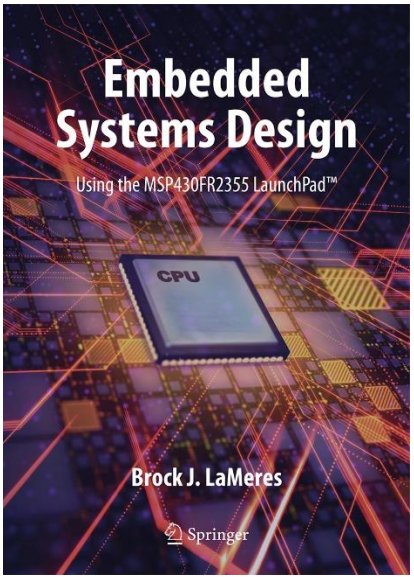
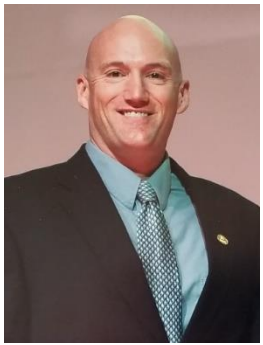
CHAPTER 12: INTRODUCTION TO TIMERS

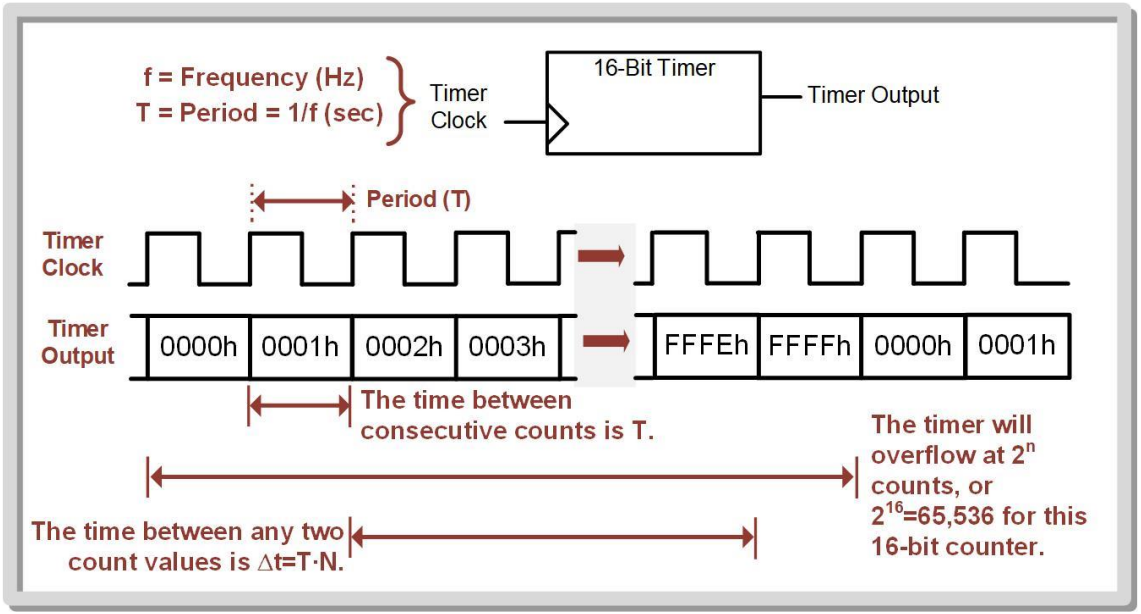
12.1 TIMER OVERVIEW





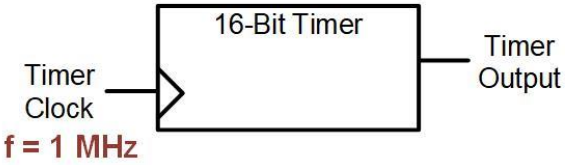
12.1 TIMER OVERVIEW

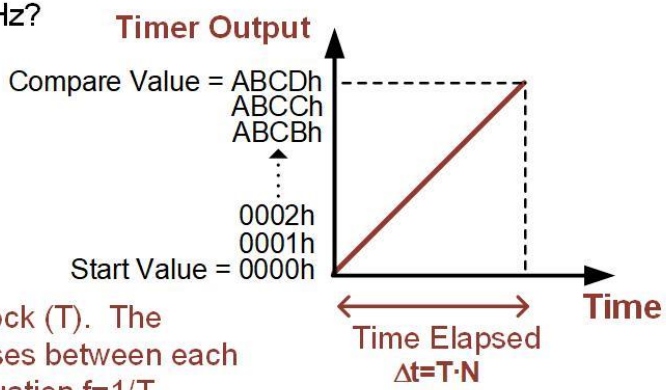
- **Timer** – a binary counter that is clocked from a free-running clock with a known frequency.
- The time elapsed can be found by multiplying the period of the clock (**T = 1/f**) by the number of counts that have occurred (**N**). **Time elapsed (Δt) = T * N**



EXAMPLE: CALCULATING THE TIME ELAPSED BETWEEN 0000h AND ABCDh

Calculate how much time elapses between when a 16-bit timer is cleared and when it reaches the value ABCDh if the clock frequency is 1 MHz?





First, we need to calculate the period of the clock (T). The period represents the amount of time that passes between each count value. The period is found using the equation f=1/T.

$$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = \frac{1}{1\ \text{MHz}} = 1e^{-6} = 1\ \mu s$$

EXAMPLE: CALCULATING THE TIME ELAPSED BETWEEN 0000h AND ABCDh

Next, we need to determine how many counts have occurred (N). In this example, the timer starts at 0000h, so we simply need to convert ABCDh to decimal and that will represent the number of counts that have occurred.

$$\text{ABCDh} = 43{,}981 \text{ decimal}$$

Finally, we multiply the period of the clock by the number of counts that have occurred to find the total amount of time that has elapsed.

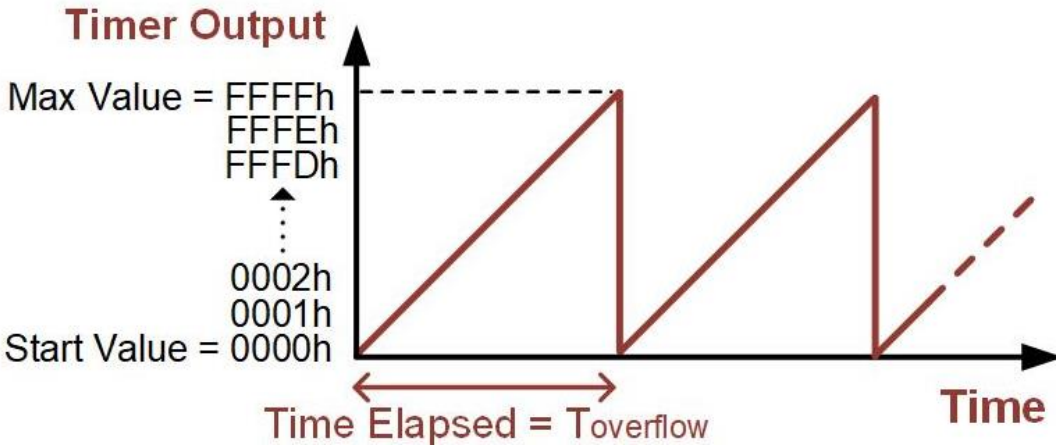
$$\begin{aligned}\Delta t &= T \cdot N \\ &= (1\ \mu s) \cdot (43{,}981) \\ &= 43.981\ ms\end{aligned}$$

12.1 TIMER OVERVIEW

- **Timer Overflow Period ($T_{overflow}$)** – the amount of time between when a timer starts at 0 and when it reaches its maximum value and rolls over back to 0.

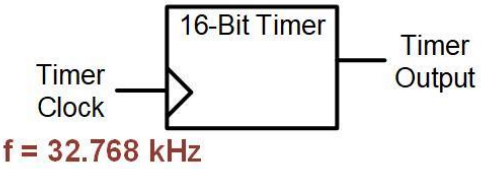
- **$\Delta t = T * N$**

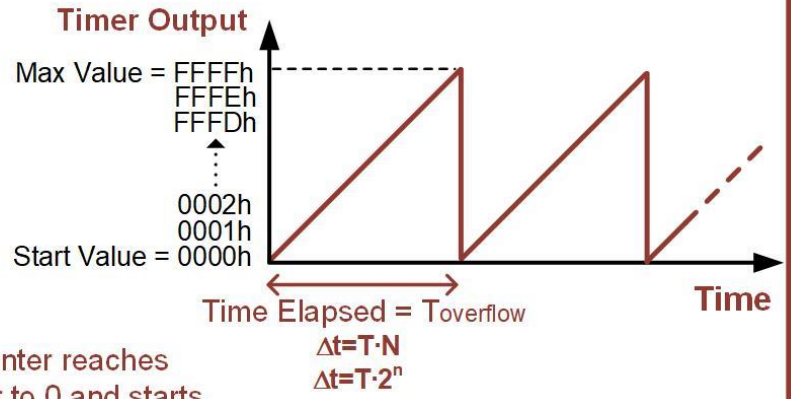
- **$N = 2^n$**



EXAMPLE: CALCULATING THE TIMER OVERFLOW PERIOD

Calculate the timer overflow period of a 16-bit timer if the clock frequency is 32.768 kHz?





Timer overflow refers to when the counter reaches its maximum value and then rolls over to 0 and starts counting again. The time between rollovers is called the timer overflow period ($T_{overflow}$). The time elapsed on a counter is always $\Delta t = T \cdot N$, but when calculating overflow, $N = 2^n$. Let's start with finding the period of the clock.

$$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = \frac{1}{32.768\ kHz} = 30.518e^{-6} = 30.518\ \mu s$$

EXAMPLE: CALCULATING THE TIMER OVERFLOW PERIOD

Next, we find the number of counts that will occur between when the timer starts at zero and when it reaches its reaches its maximum value and rolls-over (N). This is found using the expression for the number of unique codes in a binary number (i.e., 2^n where "n" is the width of the binary counter).

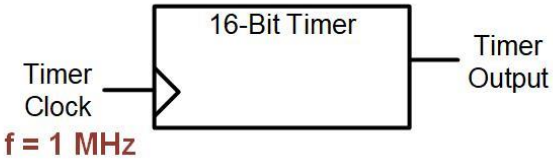
$$N = 2^n = 2^{16} = 65{,}536$$

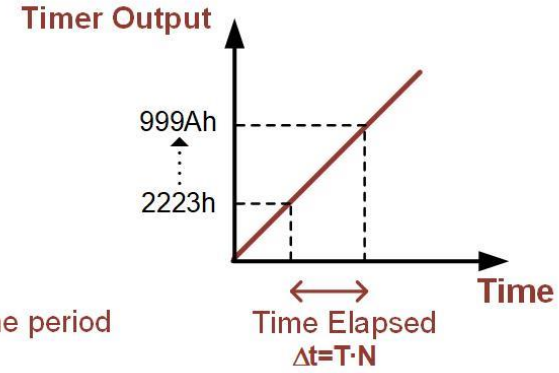
Finally, we multiply the period of the clock by the number of counts in the timer to find the timer overflow period.

$$\begin{aligned} T_{overflow} &= T \cdot N \\ &= T \cdot 2^n \\ &= (30.518\ \mu s) \cdot (65{,}536) \\ &= 2\ s \end{aligned}$$

EXAMPLE: CALCULATING THE TIME ELAPSED BETWEEN 2223h AND 999Ah

Calculate how much time elapses between the values 2223h and 999Ah on a 16-bit timer if the clock frequency is 1 MHz?





First, we need to calculate the period of the clock. The period is found using the equation f=1/T.

$$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = \frac{1}{1\ \text{MHz}} = 1e^{-6} = 1\ \mu s$$

EXAMPLE: CALCULATING THE TIME ELAPSED BETWEEN 2223h AND 999Ah

Next, we need to determine how many counts have occurred between 2223h and 999Ah (N). We can find this by simply subtracting the first value from the second value. We will need to convert the numbers to decimal at some point. This conversion can be done before or after the subtraction. Let's do the conversion before the subtraction.

2223h = 8,739 decimal **999Ah = 39,322 decimal**

Next, we subtract the first value from the second value to find the number of counts that have occurred (N).

$$N = 39{,}322 - 8{,}739$$
$$= 30{,}583$$

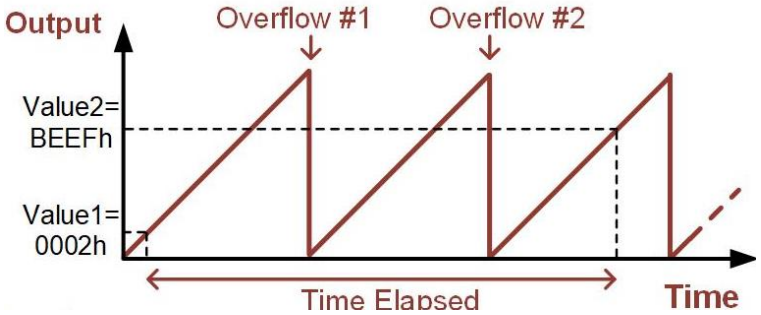
Finally, we multiply the period of the clock by the number of counts that have occurred to find the total amount of time that elapsed.

$$\Delta t = T \cdot N$$
$$= (1\ \mu s) \cdot (30{,}583)$$
$$= 30.583\ ms$$

12.1 TIMER OVERVIEW

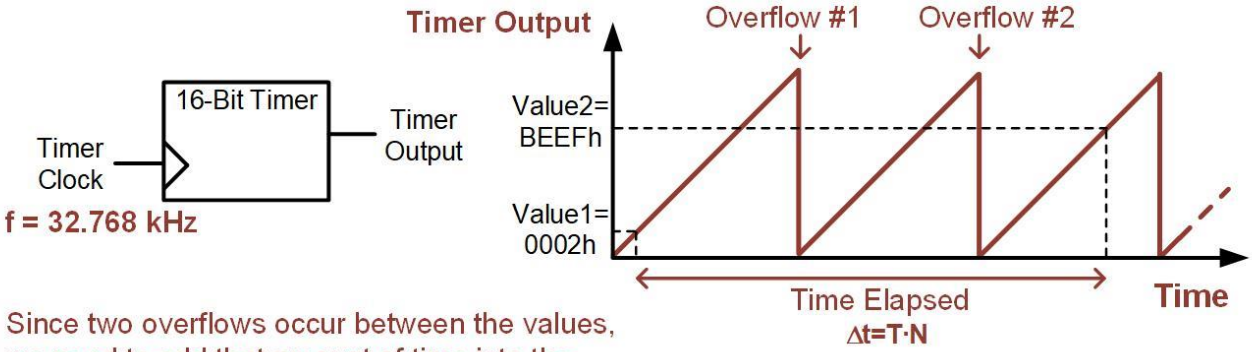
- Often when tracking the time elapsed between two external events with a timer, there can be multiple timer overflows between the events.

- This means that we must account for the additional time beyond simply subtracting two specific timer values.



EXAMPLE: CALCULATING THE TIME BETWEEN VALUES WITH OVERFLOWS

Calculate how much time elapses between the values 0002h and BEEFh on a 16-bit timer if two overflows occur between the values and the clock frequency is 32.768 kHz?



Since two overflows occur between the values, we need to add that amount of time into the calculation of how much time has elapsed. We can assume that we have put code into our program to count the number of overflows and it is provided to us as an integer. We simply add the number of counts in each overflow to the second timer value and then subtract the first timer value. The total amount of time elapsed then becomes:

$$\Delta t = T \cdot [((\#\ of\ Overflows) \cdot 2^n + Value2) - Value1]$$

EXAMPLE: CALCULATING THE TIME BETWEEN VALUES WITH OVERFLOWS

Let's first calculate the period of the clock.

$$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = \frac{1}{32.768\ \text{kHz}} = 30.518e^{-6} = 30.518\ \mu s$$

Next, let's convert our two timer values into decimal so that the calculations are simpler.

0002h = 2 decimal

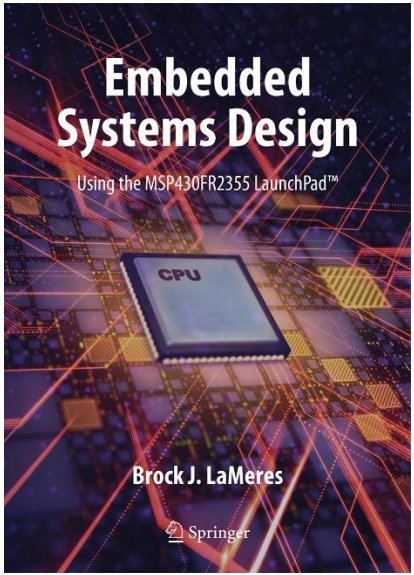
BEEFh = 48,879 decimal

Finally, we add all of the counts together and multiply by the period of the clock to find the total amount of time elapsed.

$$\Delta t = T \cdot N$$
$$= T \cdot [((\#\ \text{of Overflows}) \cdot 2^n + \text{Value2}) - \text{Value1}]$$
$$= (30.518\ \text{us}) \cdot [((2) \cdot 2^{16} + 48{,}879) - 2]$$
$$= 5.491\ \text{s}$$

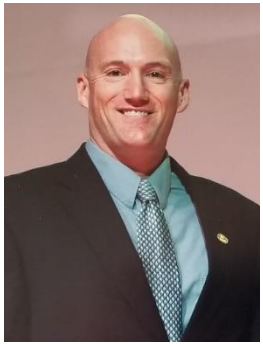
CHAPTER 12: INTRODUCTION TO TIMERS

12.1 TIMER OVERVIEW



Subscribe

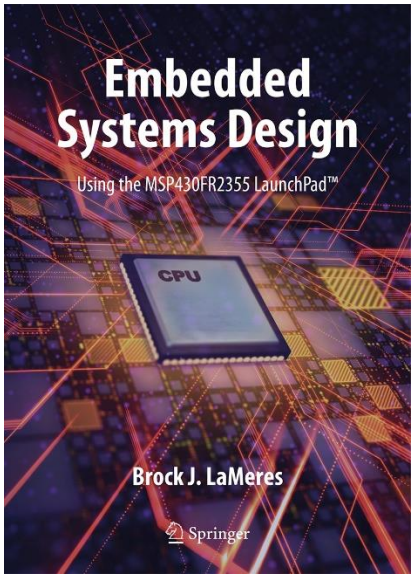
www.youtube.com/c/DigitalLogicProgramming_LaMeres

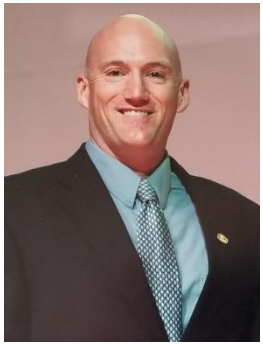


Embedded Systems Design

Chapter 12: Introduction to Timers

12.2 Timer Overflows on the MSP430FR2355 – System Overview

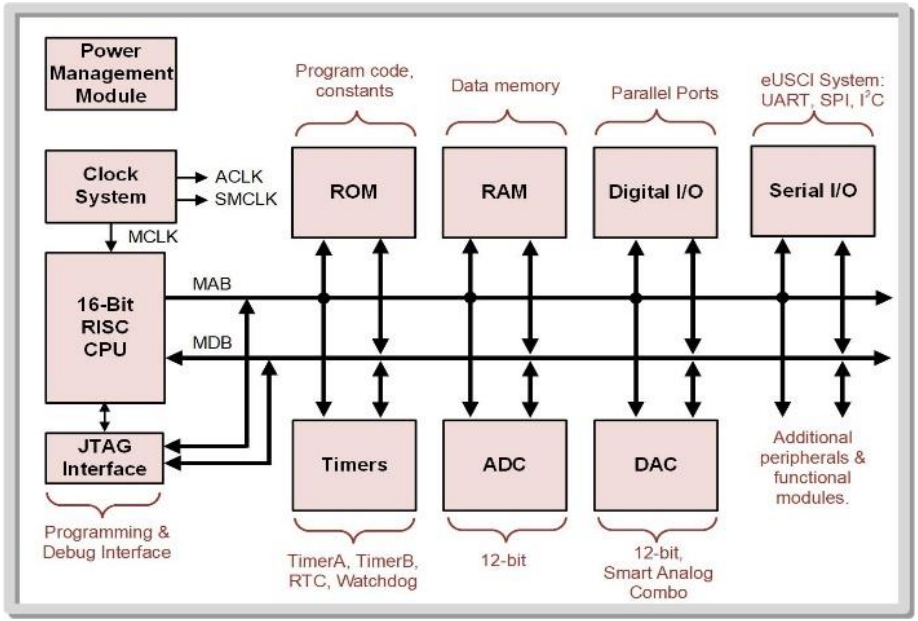




Brock J. LaMeres, Ph.D.

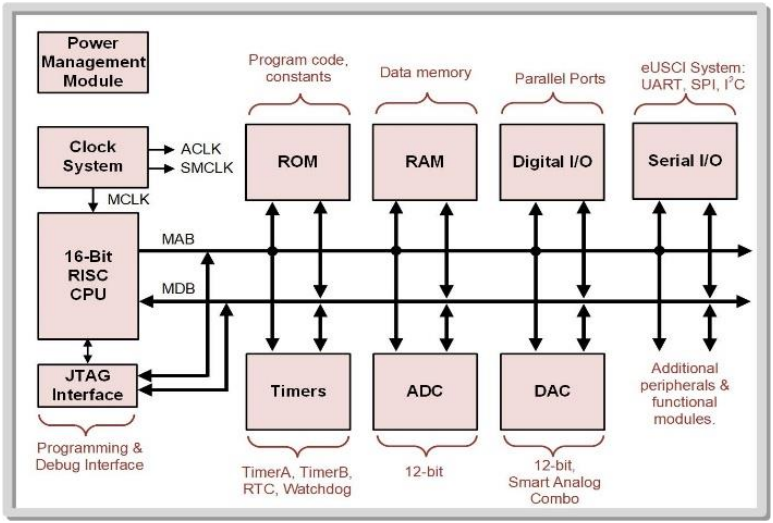
12.2 TIMERS ON THE MSP430

- MSP430 architecture contains the following timer sub-systems: **Timer_A, Timer_B, real-time clock counter (RTC),** and **Watchdog counter**.



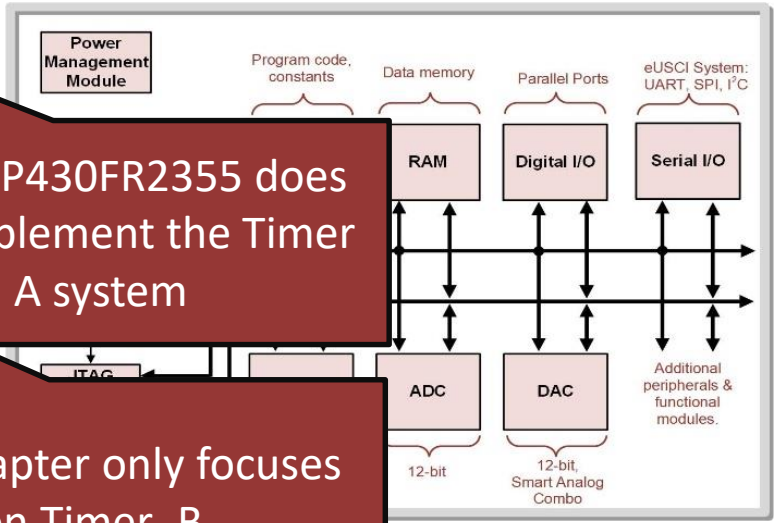
12.2 TIMERS ON THE MSP430

- **Timer_A** and **Timer_B:**
  - have multiple, independent binary counters that provide separate timing capability.
  - Each timer can generate interrupts when its value either matches a value placed into a compare register, or when it overflows.



12.2 TIMERS ON THE MSP430

- **Timer_A and Timer_B:** have multiple, independent binary counters that p separate timing capability.
  - Each timer can generate interrupts when its value matches a valu into a compare or when it over

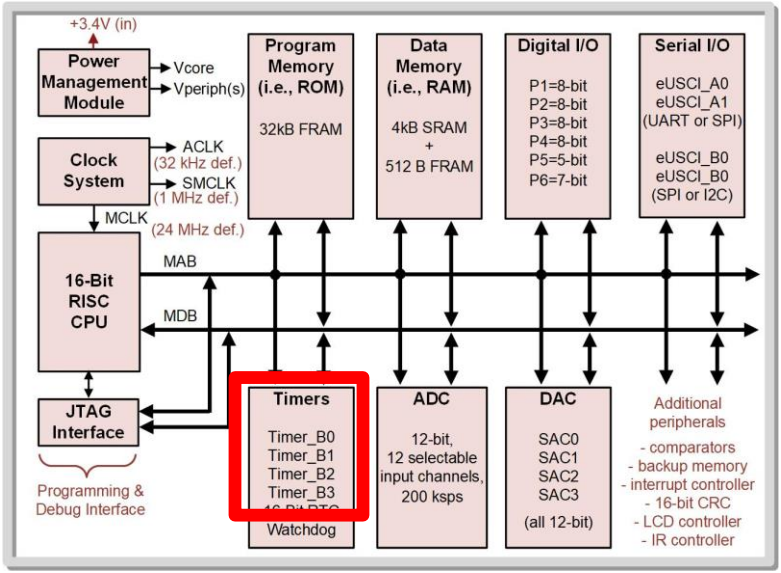


The MSP430FR2355 does NOT implement the Timer A system

This chapter only focuses on Timer_B.

12.2 TIMERS ON THE MSP430FR2355

- The MSP430FR2355 Timer_B system provides four independent timers – **TB0, TB1, TB2,** and **TB3**.

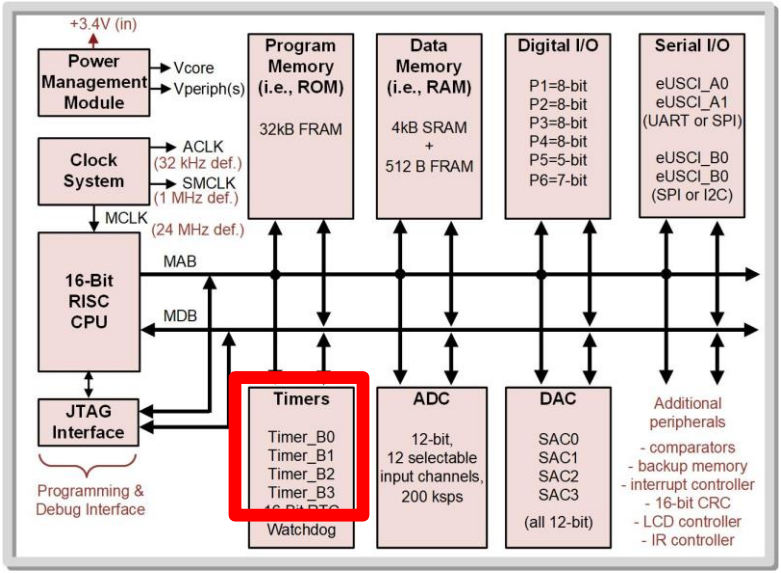


12.2 TIMERS ON THE MSP430FR2355

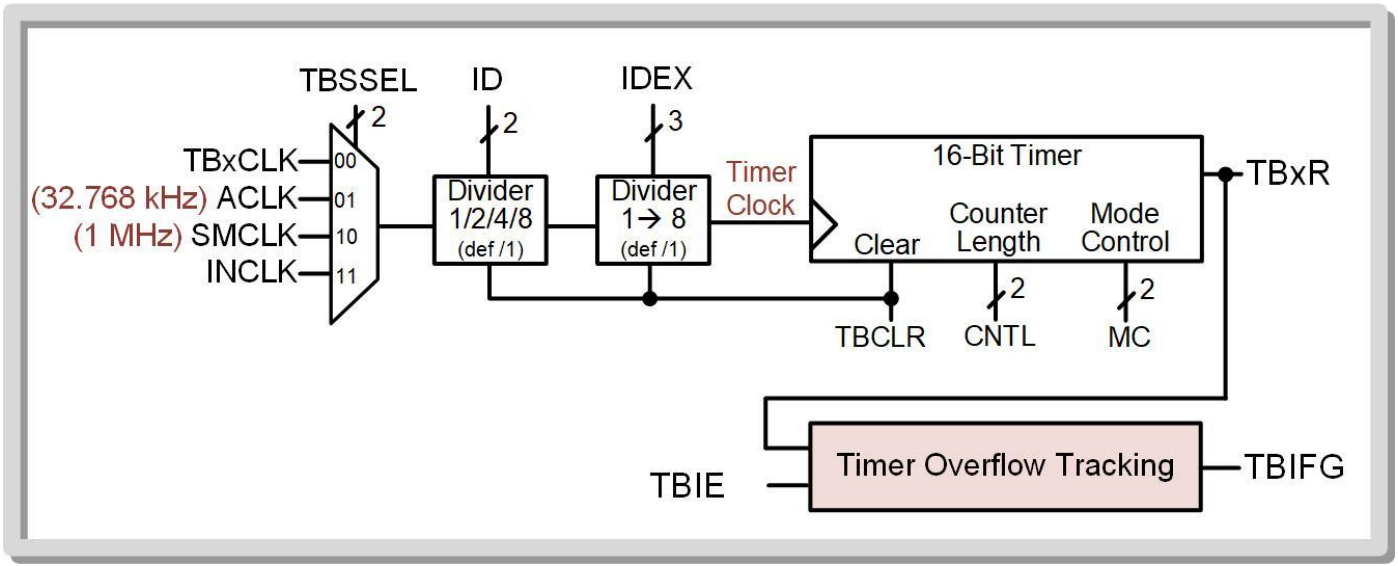
- The MSP430FR2355 Timer_B system provides four independent timers – **TB0, TB1, TB2,** and **TB3**.

- Timers TB0, TB1, and TB2 each have three capture/compare registers associated with them – Timer0_**B3**, Timer1_**B3**, and Timer2_**B3**.

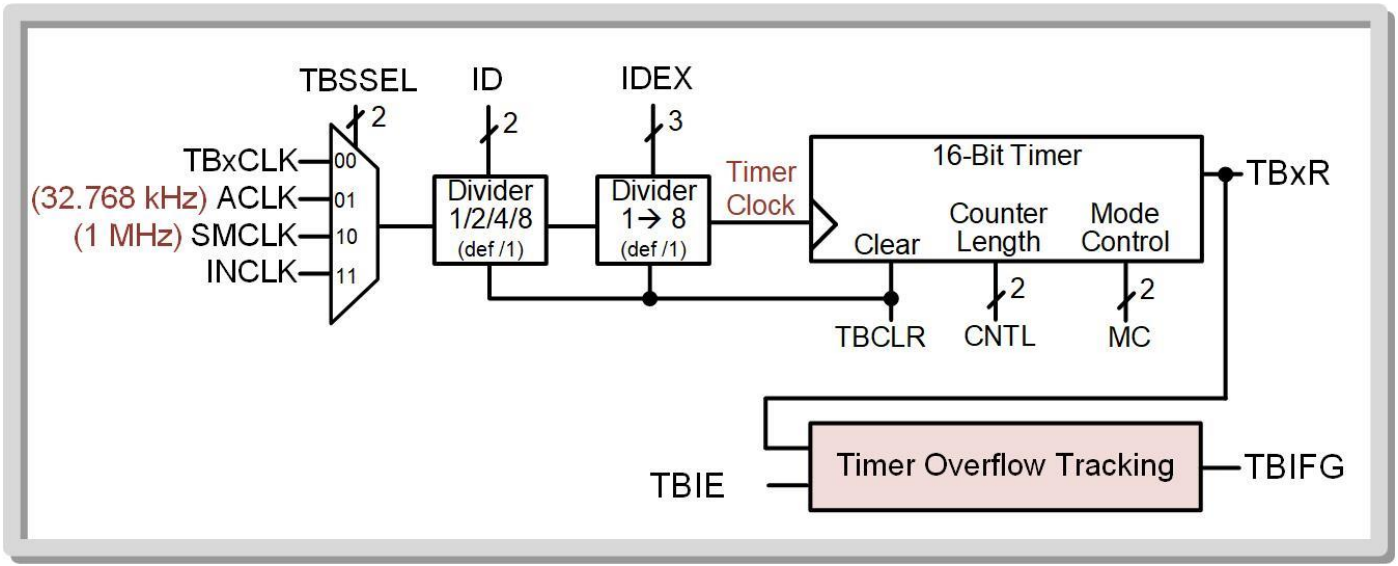
- Timer TB3 has seven capture/compare registers so it is often referred to as Timer3_**B7**.



12.2 TIMER OVERFLOWS ON THE MSP430FR2355

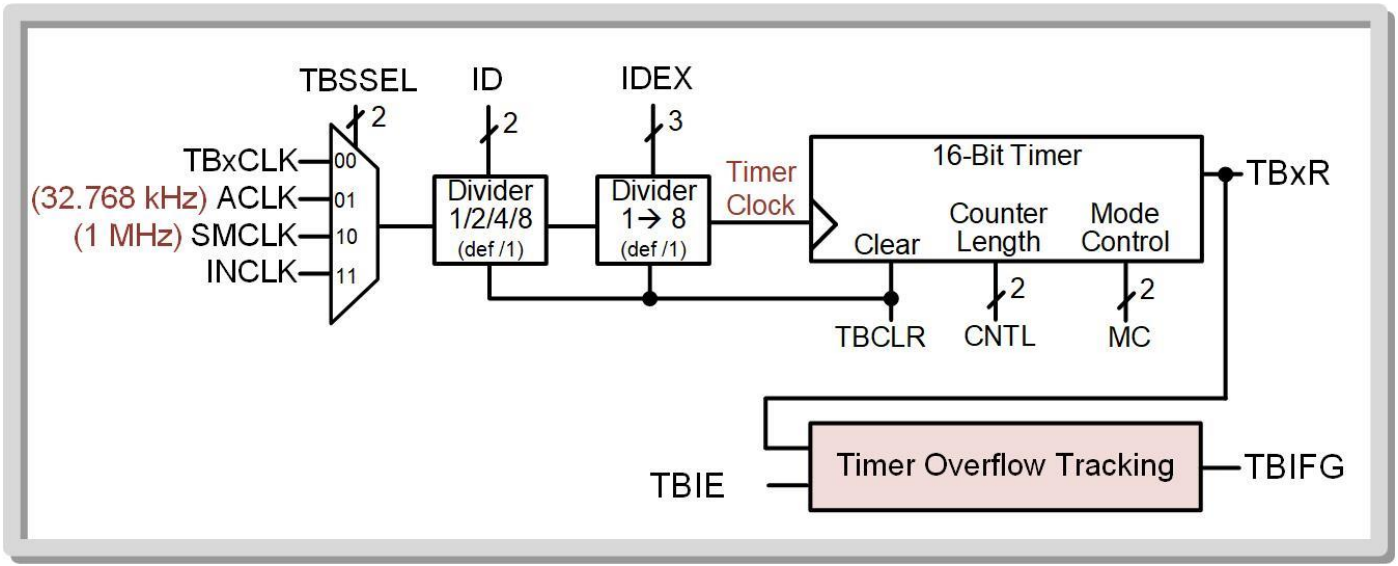


12.2 TIMER OVERFLOWS ON THE MSP430FR2355



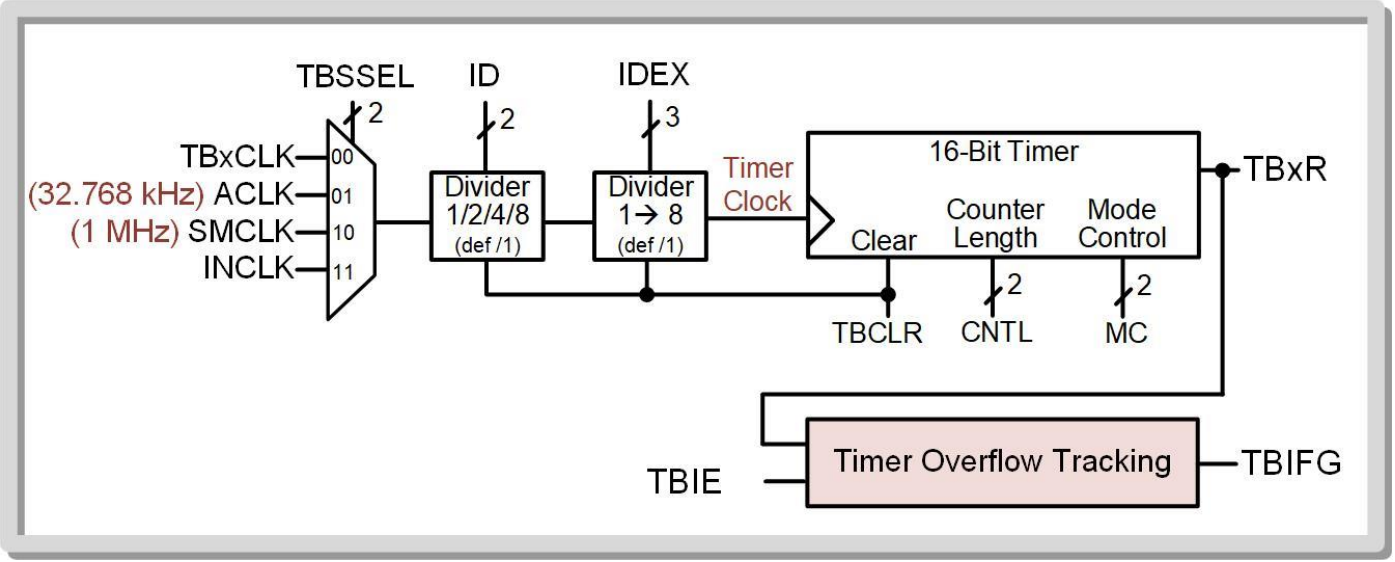
- The Timer_B system has a 16_bit binary counter which has a variety of settings that are controlled by the user.

12.2 TIMER OVERFLOWS ON THE MSP430FR2355



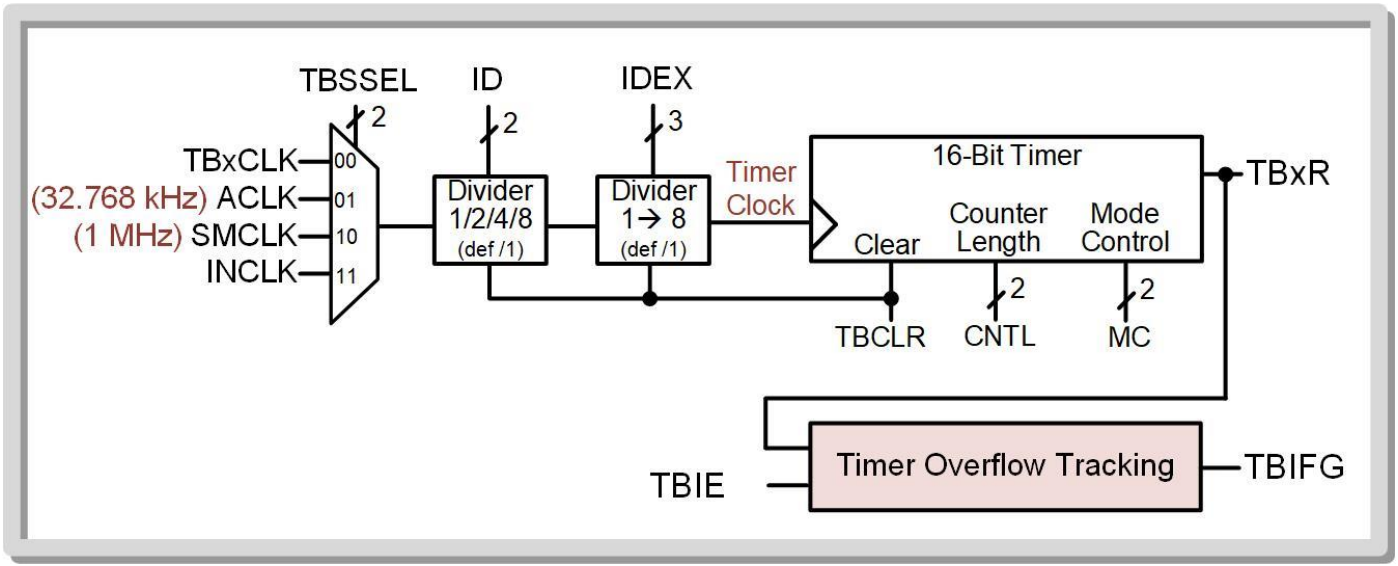
- Each timer has selectable clock inputs and the ability to divide down the clock to get slower counting frequencies.
- Types of clock sources: **ACLK (32.768 kHz)** and **SMCLK (1 MHz)**.

12.2 TIMER OVERFLOWS ON THE MSP430FR2355



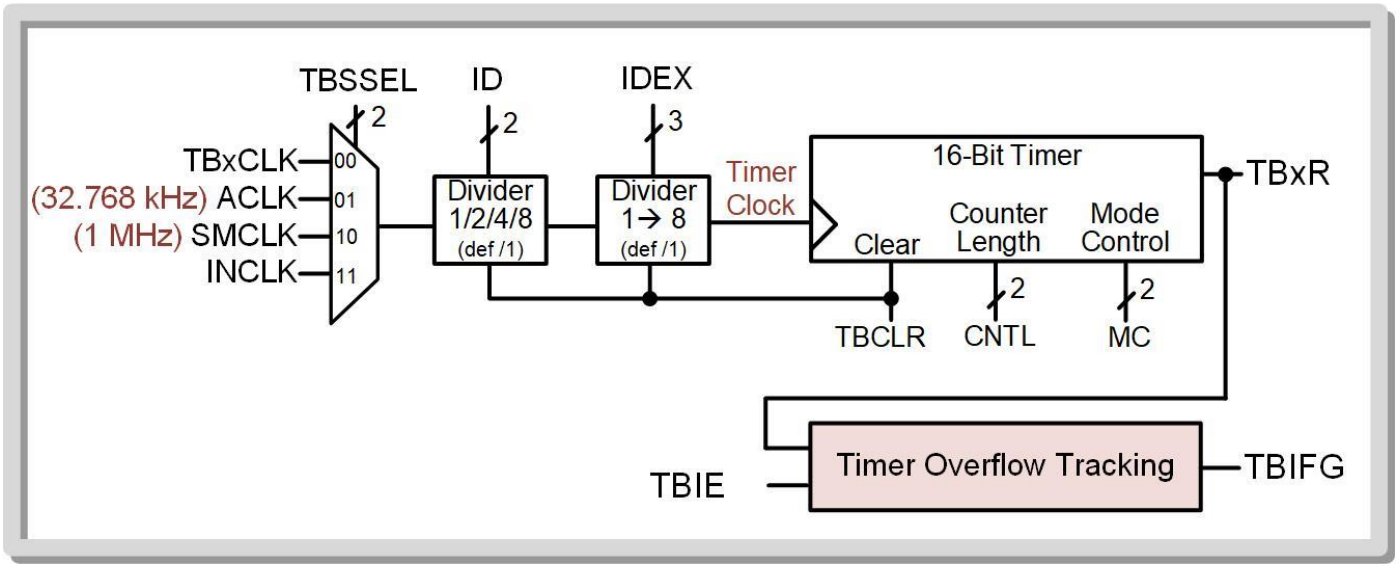
- A first stage clock divider (ID) can divide the clock by 1, 2, 4, and 8.
- A second stage clock divider (IDEX) can further divide the clock by 1, 2, 3, 4, 5, 6, 7, and 8.
- This gives 32 different divider settings (min=1, max=64).

12.2 TIMER OVERFLOWS ON THE MSP430FR2355



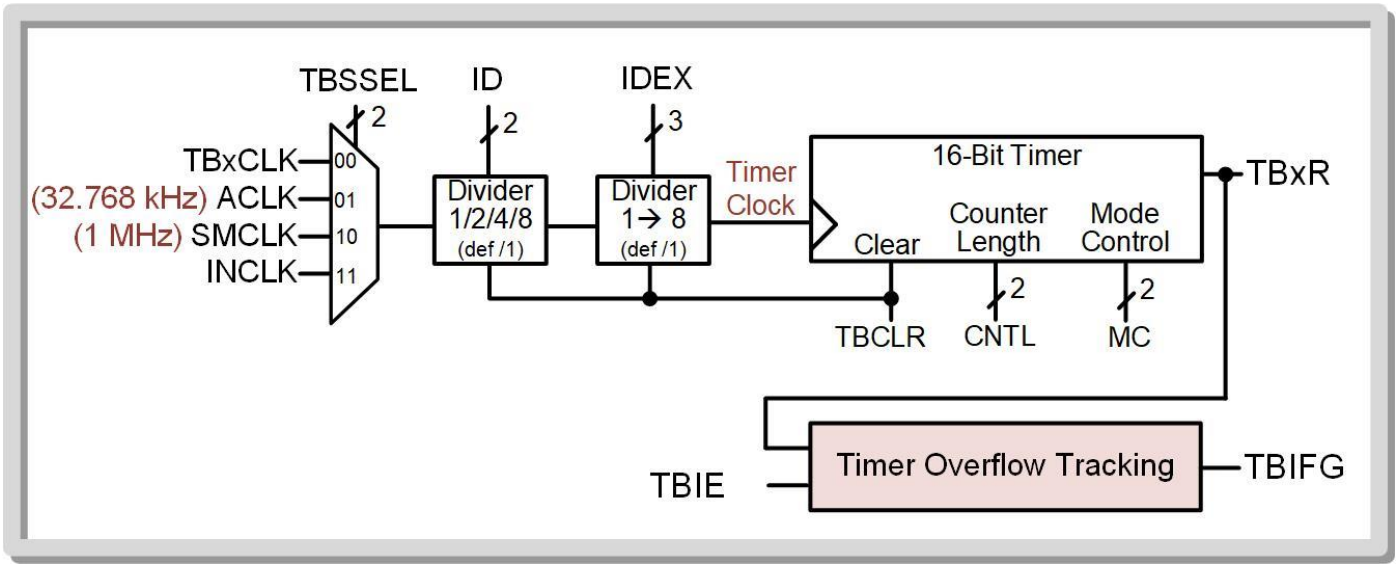
- The counter length can be changed using CNTL (8, 10, 12, 16 bits)

12.2 TIMER OVERFLOWS ON THE MSP430FR2355



- The counting mode can be changed using MC (Up, Continuous, Up/Down).
  - **Continuous counting mode** - counts up to its maximum value and then rolls-over to 0.
  - In this mode, a roll-over (2^n) can generate an interrupt.

12.2 TIMER OVERFLOWS ON THE MSP430FR2355



- Recommended Steps to Configure the Timer
  - Write a 1 to the TBCLR bit (TBCLR = 1) to clear TBxR, the clock divider states, and the counter direction.
  - Apply desired configurations to TBxCTL.

12.2 TIMER_B CONTROL REGISTER (TBxCTL)

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Rsv	TBCLGRP		CNTL		Rsv	TBSSEL		ID		MC		Rsv	TBCLR	TBIE	TBIFG
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15	Reserved	-
14:13	TBCLGRP	Timer_B Compare Latch Groupings (see device specific datasheet for info)
12:11	CNTL	Counter Length 00=16-bit 01=12-bit 10=10-bit 11=8-bit
10	Reserved	-
9:8	TBSSEL	Timer_B Clock Source Select 00=TBxCLK (external pin) 01=ACLK (32.768 kHz) 10=SMCLK (1 MHz) 11=INCLK (external pin)
7:6	ID	Input Clock Divider 00=Divide by 1 01=Divide by 2 10=Divide by 4 11=Divide by 8
5:4	MC	Mode Control 00=Stop Mode. Timer is Halted. 01=Up Mode: (Used for Timer Compares. Timer Counts up to TBxCL0) 10=Continuous Mode: Timer counts full range as set by CNTL and overflows. 11=Up/Down Mode: Timer counts up to TBxCL0 and down to 0000h.
3	Reserved	-
2	TBCLR	Timer_B Clear. Setting this bit clears the timer and the clock dividers.
1	TBIE	Timer_B Interrupt Enable (0=IRQ Disabled; 1=IRQ Enabled)
0	TBIFG	Timer_B Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending)

12.2 TIMER_B EXPANSION REGISTER 0 (TBxEX0) DETAILS

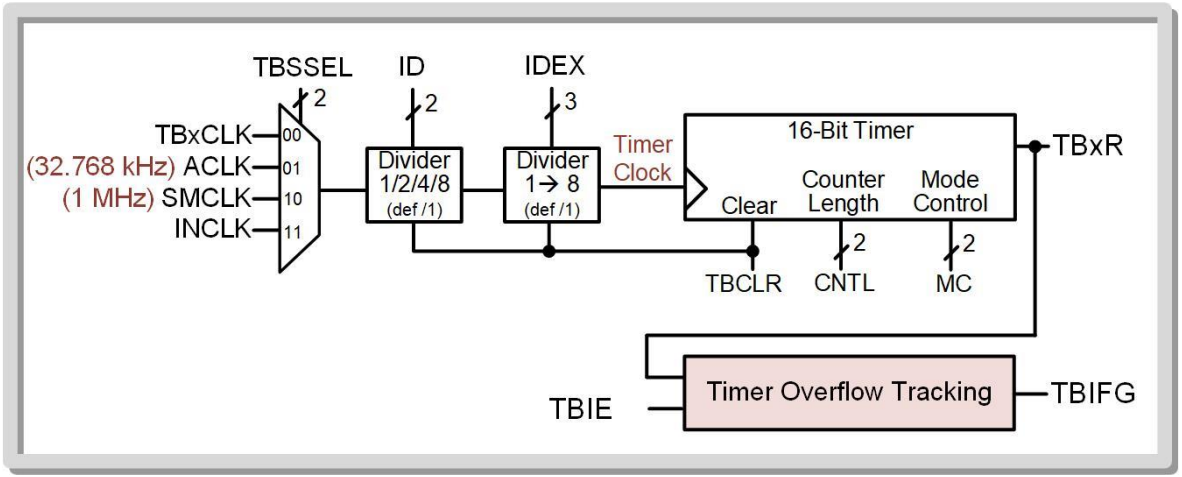
Timer_B x Expansion Register 0 (TBxEX0)

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Rsv													IDEX		
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:3	Reserved	-
2:0	IDEX	Input Clock Divider Expansion 000=Divide by 1 001=Divide by 2 010=Divide by 3 011=Divide by 4 100=Divide by 5 101=Divide by 6 110=Divide by 7 111=Divide by 8

12.2 TIMER OVERFLOWS ON THE MSP430FR2355

- Using the overflow interrupt
  - Flag = TBIFG bit in TBxCTL register.
  - Local Enable = TBIE bit in TBxCTL register.
  - Global Enable = GIE bit in SR.



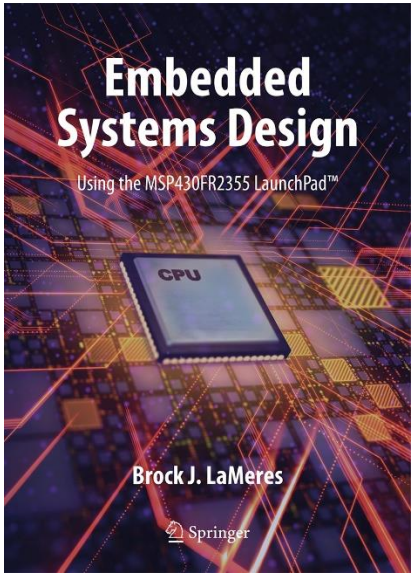
12.2 TIMER OVERFLOWS ON THE MSP430FR2355

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDRESS	CCS SECTION
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	.reset
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	.int45
User NMI	NMIIFG, OFIFG	Non-Maskable	FFFAh	.int44
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	.int43
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	.int42
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	.int41
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIFG2, TB1IFG (TB1IV)	Maskable	FFF2h	.int40
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	.int39
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIFG2, TB2IFG (TB2IV)	Maskable	FFEEh	.int38
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	.int37
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	.int36
RTC counter	RTCIFG	Maskable	FFE8h	.int35
Watchdog interval	WDTIFG	Maskable	FFE6h	.int34

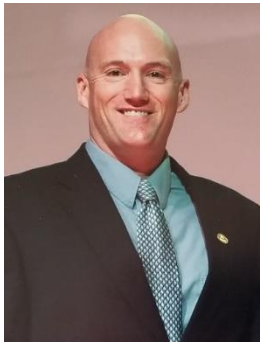
Embedded Systems Design

Chapter 12: Introduction to Timers

12.2 Timer Overflows on the MSP430FR2355 – System Overview



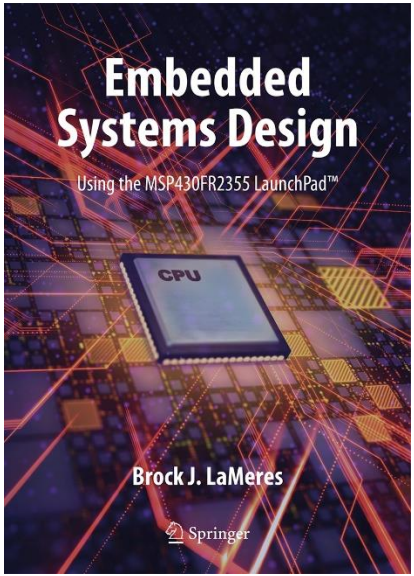
www.youtube.com/c/DigitalLogicProgramming_LaMeres

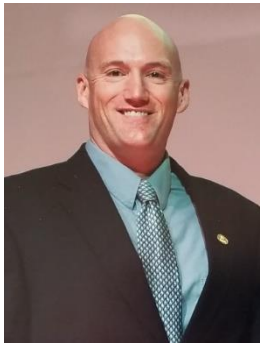


Brock J. LaMeres, Ph.D.

CHAPTER 12: INTRODUCTION TO TIMERS

12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (ACLK)

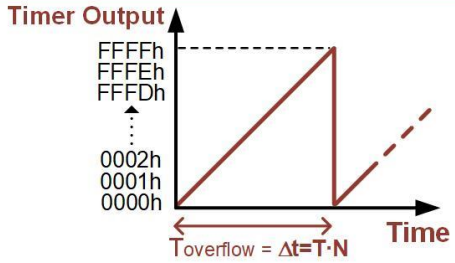




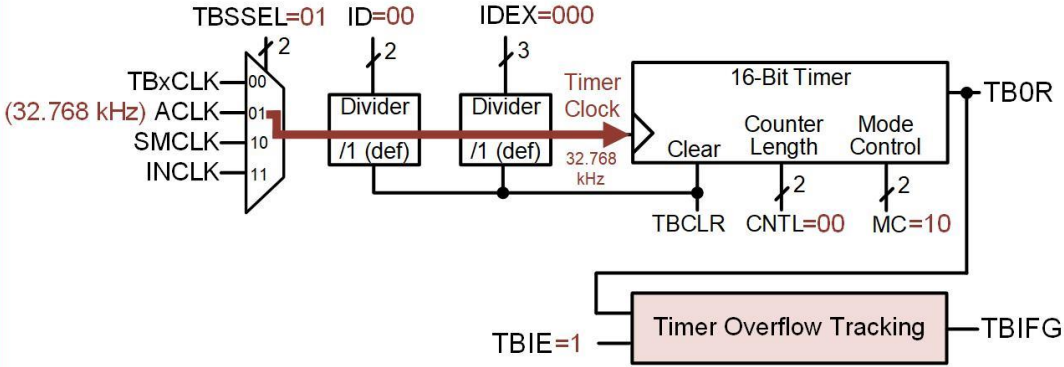
EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

Let's design a program that will toggle LED1 every time TB0 overflows when clocked with **ACLK**. First, let's calculate how long the timer overflow period is using this configuration. We will not divide down ACLK so the timer clock will be 32.768 kHz.

$$T_{overflow} = T \cdot N = (1/f) \cdot 2^n = (1/32.768k) \cdot 2^{16} = 2 \text{ seconds}$$



Next, we need to decide on the settings for TB0. We want the clock source to be ACLK with both dividers set to 1. We also want the counter length to be 16-bits and the counting mode to be continuous. We then need to enable TBIE. We can use the default values for the dividers and the counter length to make our program more compact. The following is our timer setup:



EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_ACLK_Overflow

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.”

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0

;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__ACLK, &TB0CTL
        bis.w   #MC__CONTINUOUS, &TB0CTL
        bis.w   #TBIE, &TB0CTL
        bic.w   #TBIFG, &TB0CTL
        bis.w   #GIE, SR

main:
        jmp     main

;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_Overflow:
        xor.b   #BIT0, &P1OUT
        bic.w   #TBIFG, &TB0CTL
        reti
```

- Setup LED1 to be an output: (P1DIR.0=1)
- Put LED1's initial value at zero: (P1OUT.0=0)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select ACLK as Timer Source: (TBSSEL=01)
- Choose Continuous Counting: (MC=10)
- Enable Overflow Interrupt: (TBIE=1)
- Clear Interrupt Flag: (TBIFG=0)
- Enable Maskable Interrupts: (GIE=1)

The main program doesn't do anything but loop forever.

The ISR toggles LED1 and clears the TBIFG flag. The ISR needs an address label to mark its starting address. The ISR needs to end with `reti` to return to the main program.

EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int42"
         short  ISR_TB0_Overflow
```

We initialize the vector table for Timer0_B3 using the CCS section name found in the linker file. Timer0_B3 has two vector addresses associated with it. We want the one that has "TB0IFG" listed in the interrupt flag column (.int42).

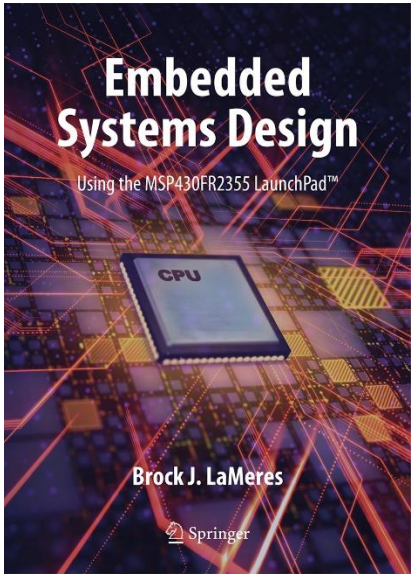
At this vector address, we want to download the starting address of the ISR (i.e., ISR_TB0_Overflow).

Step 3: Debug your program and run it. You will see LED1 toggle on and off every two seconds.

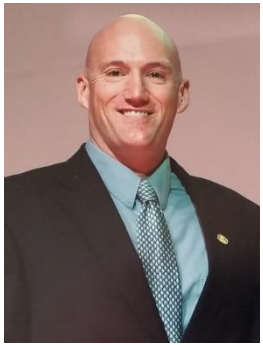
Did it work? Did you see LED1 toggle every 2 seconds? Notice that we don't need to create a delay in the main program anymore as the timer system takes care of it. This is much more efficient, and accurate.

CHAPTER 12: INTRODUCTION TO TIMERS

12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (ACLK)

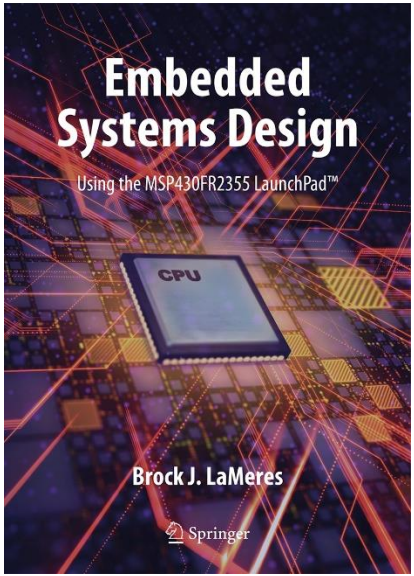


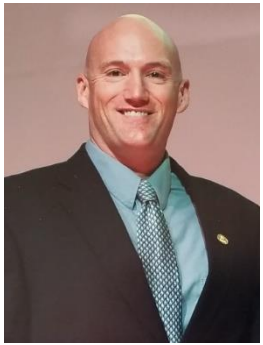
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 12: INTRODUCTION TO TIMERS

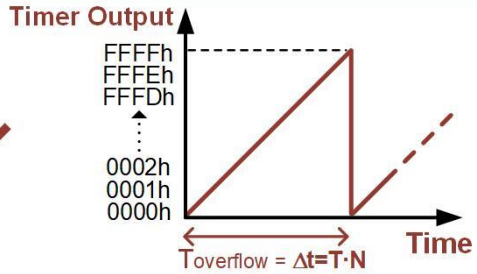
12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (ACLK, 12-BIT)



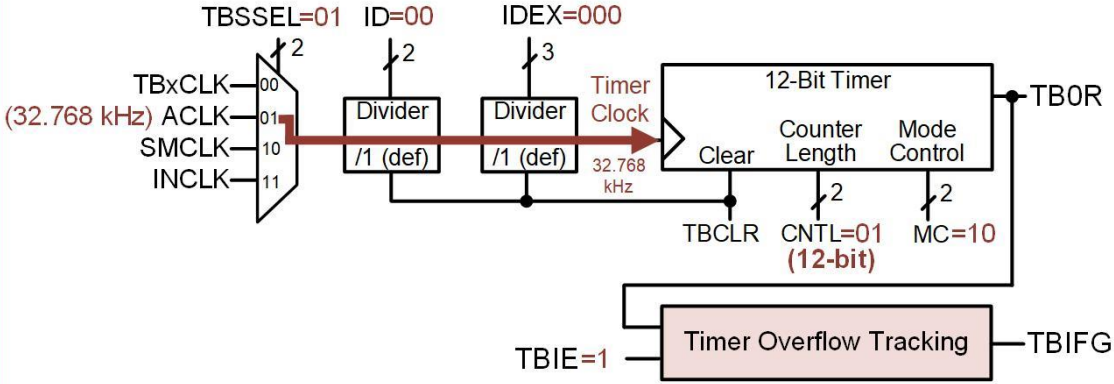


Let's design a program that will toggle LED1 every time TB0 overflows when clocked with **ACLK**, but in order to **speed up the overflow period**, we'll configure the timer as 12-bits. First, let's calculate the timer overflow period.

$$T_{overflow} = T \cdot N = (1/f) \cdot 2^n = (1/32.768k) \cdot 2^{12}$$
$$= \mathbf{125\ ms}\ (\sim 8 \text{ times per second})$$



Next, we need to decide on the settings for TB0. We want the clock source to be ACLK with both dividers set to 1. We also want the counter length to be **12-bits** and the counting mode to be continuous. We then need to enable TBIE. We can use the default values for the dividers and the counter length to make our program more compact. The following is our timer setup:



EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK AND A 12-BIT COUNTER CONFIG

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_ACLK_Overflow_12bit

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.” This is the same as the last ACLK example with one additional instruction to configure CNTL.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0

;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__ACLK, &TB0CTL
        bis.w   #MC__CONTINUOUS, &TB0CTL
        bis.w   #CNTL_1, &TB0CTL
        bis.w   #TBIE, &TB0CTL
        bic.w   #TBIFG, &TB0CTL
        bis.w   #GIE, SR

main:
        jmp     main
;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_Overflow:
        xor.b   #BIT0, &P1OUT
        bic.w   #TBIFG, &TB0CTL
        reti
```

This is the same as the last ACLK example with one additional instruction to configure CNTL.

- Setup LED1 to be an output: (P1DIR.0=1)
- Put LED1's initial value at zero: (P1OUT.0=0)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select ACLK as Timer Source: (TBSSEL=01)
- Choose Continuous Counting: (MC=10)
- **Choose 12-bit Count Length: (CNTL=01)**
- Enable Overflow Interrupt: (TBIE=1)
- Clear Interrupt Flag: (TBIFG=0)
- Enable Maskable Interrupts: (GIE=1)

The main program doesn't do anything but loop forever.

The ISR toggles LED1 and clears the TBIFG flag. The ISR needs an address label to mark its starting address. The ISR needs to end with `reti` to return to the main program.

EX: TOGGLING LED1 ON TB0 OVERFLOW USING ACLK

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int42"
         short  ISR_TB0_Overflow
```

We initialize the vector table for Timer0_B3 using the CCS section name found in the linker file. Timer0_B3 has two vector addresses associated with it. We want the one that has "TB0IFG" listed in the interrupt flag column (.int42).

At this vector address, we want to download the starting address of the ISR (i.e., ISR_TB0_Overflow).

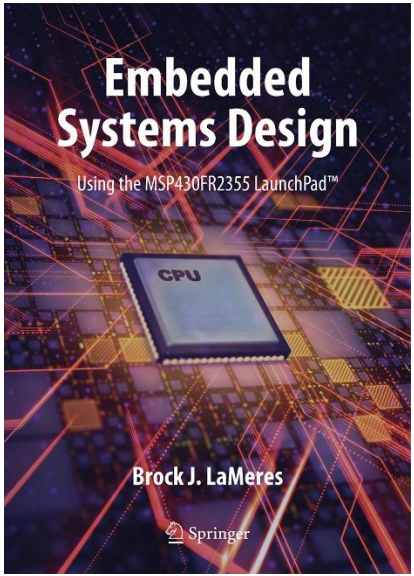
Step 3: Debug your program and run it. You will see LED1 toggle on and off every 125 ms.

Did it work? Did you see LED1 toggle every 125 ms? The ISR will trigger 8 times per second, which means LED1 will turn on 4 times per second.

Embedded Systems Design

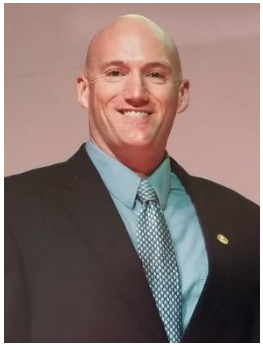
Chapter 12: Introduction to Timers

12.2 Timer Overflows on the MSP430FR2355: Example – Toggling LED1 on Timer Overflow (ACLK, 12-bit)



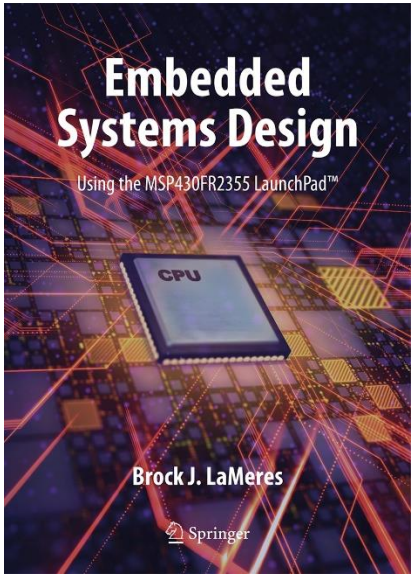
Subscribe

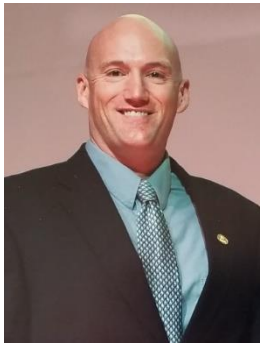
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 12: INTRODUCTION TO TIMERS

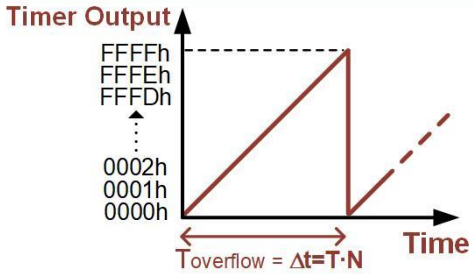
12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (SMCLK)



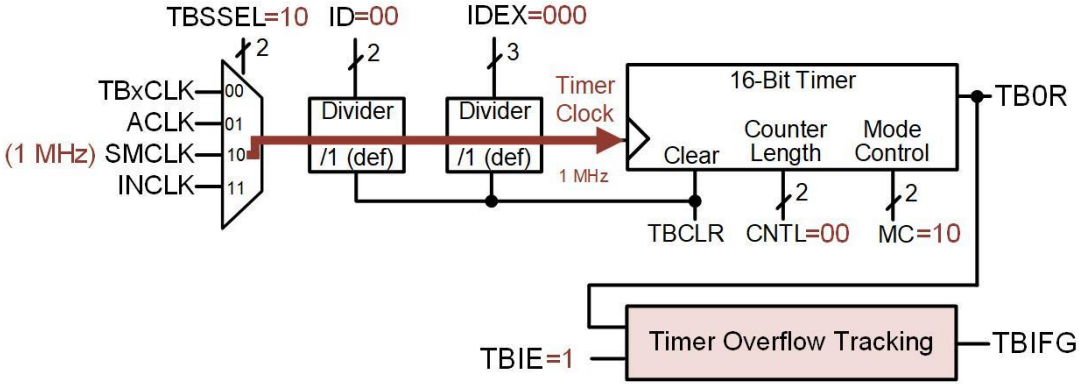


Let's design a program that will toggle LED1 every time TB0 overflows when clocked with **SMCLK**. First, let's calculate how long the timer overflow period is using this configuration. We will not divide down SMCLK so the timer clock will be 1 MHz.

$$T_{overflow} = T \cdot N = (1/f) \cdot 2^n = (1/1M) \cdot 2^{16}$$
$$= 65.5 \text{ ms } (\sim 15 \text{ times per second})$$



Next, we need to decide on the settings for TB0. We want the clock source to be SMCLK with both dividers set to 1. We also want the counter length to be 16-bits and the counting mode to be continuous. We then need to enable TBIE. We can use the default values for the dividers and the counter length to make our program more compact. The following is our timer setup:



EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_SMCLK_Overflow

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.”

EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0

;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__SMCLK, &TB0CTL
        bis.w   #MC__CONTINUOUS, &TB0CTL
        bis.w   #TBIE, &TB0CTL
        bic.w   #TBIFG, &TB0CTL
        bis.w   #GIE, SR

main:
        jmp     main

;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_Overflow:
        xor.b   #BIT0, &P1OUT
        bic.w   #TBIFG, &TB0CTL
        reti
```

This is the same as the last example except for the instruction to select the clock source.

- Setup LED1 to be an output: (P1DIR.0=1)
- Put LED1's initial value at zero: (P1OUT.0=0)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select SMCLK as Timer Source: (TBSSEL=10)
- Choose Continuous Counting: (MC=10)
- Enable Overflow Interrupt: (TBIE=1)
- Clear Interrupt Flag: (TBIFG=0)
- Enable Maskable Interrupts: (GIE=1)

The main program doesn't do anything but loop forever.

The ISR toggles LED1 and clears the TBIFG flag. The ISR needs an address label to mark its starting address. The ISR needs to end with `reti` to return to the main program.

EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int42"
         short  ISR_TB0_Overflow
```

We initialize the vector table for Timer0_B3 using the CCS section name found in the linker file. Timer0_B3 has two vectors addresses associated with it. We want the one that has "TB0IFG" listed in the interrupt flag column (.int42).

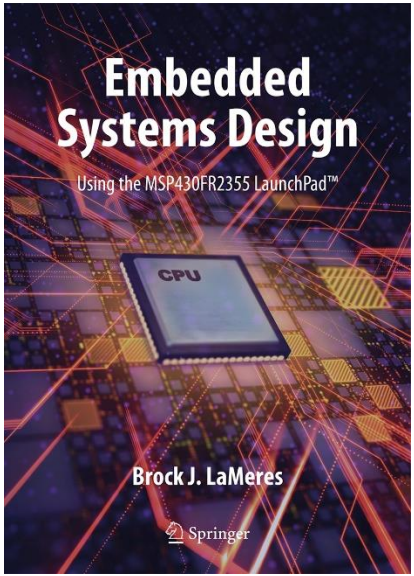
At this vector address, we want to download the starting address of the ISR.

Step 3: Debug your program and run it. You will see LED1 toggle on and off every 65.5 ms.

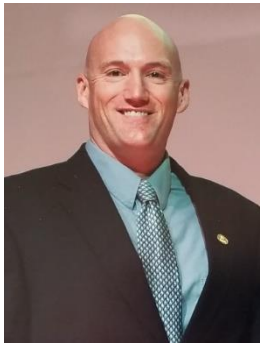
Did it work? Did you see LED1 toggle every 65.5ms? This blinks pretty fast, but you will be able to see it with the naked eye. The ISR will trigger ~15 times per second, which means LED1 will turn on ~8 times per second.

CHAPTER 12: INTRODUCTION TO TIMERS

12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (SMCLK)

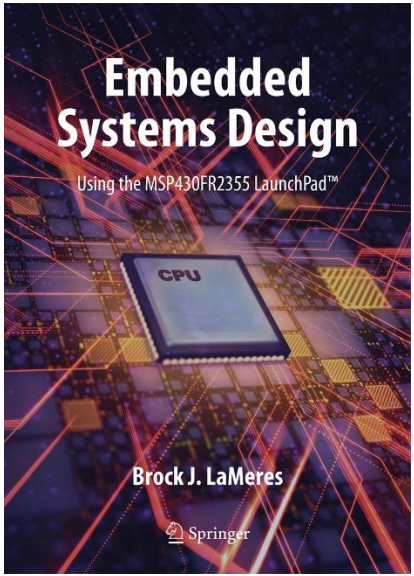


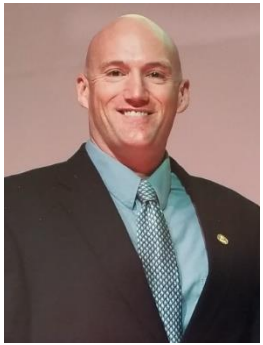
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 12: INTRODUCTION TO TIMERS

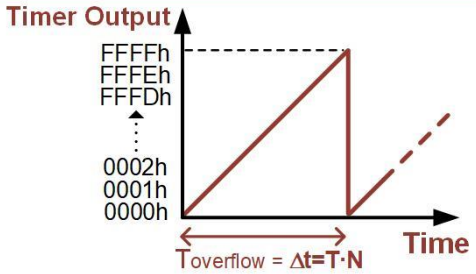
12.2 TIMER OVERFLOWS ON THE MSP430FR2355: EXAMPLE – TOGGLING LED1 ON TIMER OVERFLOW (SMCLK, DIV4)



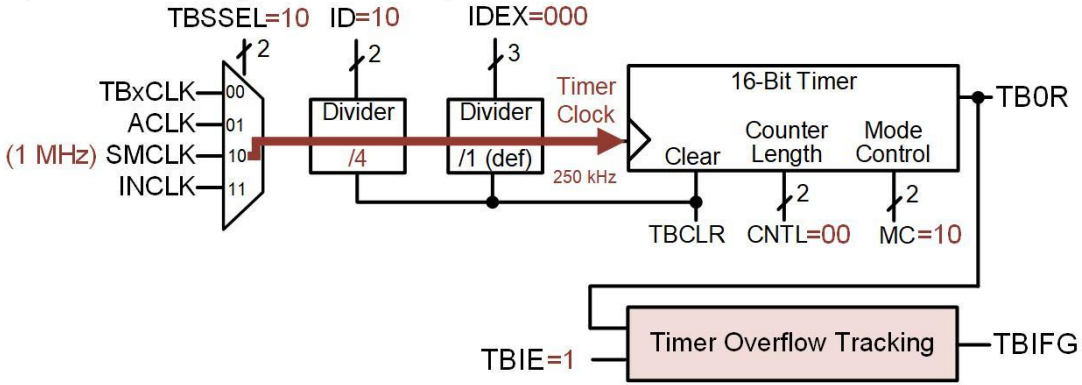


Let's design a program that will toggle LED1 every time TB0 overflows when clocked with **SMCLK**, but in order to **slow down the overflow period**, we'll divide the incoming SMCLK by 4. First, let's calculate the timer overflow period.

$$T_{overflow} = T \cdot N = (1/(f/4)) \cdot 2^n = (1/(1M/4)) \cdot 2^{16}$$
$$= 262 \text{ ms } (\sim 4 \text{ times per second})$$



Next, we need to decide on the settings for TB0. We want the clock source to be SMCLK. We also want to set the **first divider to 4** using the ID bits in TB0CTL. We want the counter length to be 16-bits and the counting mode to be continuous. We then need to enable TBIE. We can use the default values for second divider and the counter length to make our program more compact. The following is our timer setup:



EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK AND A DIVIDE-BY-4 CLOCK CONFIG

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_SMCLK_Overflow_Div4

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.”

EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK AND A DIVIDE-BY-4 CLOCK CONFIG

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0

;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__SMCLK, &TB0CTL
        bis.w   #MC__CONTINUOUS, &TB0CTL
        bis.w   #ID__4, &TB0CTL
        bis.w   #TBIE, &TB0CTL
        bic.w   #TBIFG, &TB0CTL
        bis.w   #GIE, SR

main:
        jmp     main
;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_Overflow:
        xor.b   #BIT0, &P1OUT
        bic.w   #TBIFG, &TB0CTL
        reti
```

This is the same as the last SMCLK example with one additional instruction to configure ID.

- Setup LED1 to be an output: (P1DIR.0=1)
- Put LED1's initial value at zero: (P1OUT.0=0)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select SMCLK as Source: (TBSSEL=10)
- Choose Continuous Counting: (MC=10)
- **Set Div-by-4 in 1st Divider: (ID=10)**
- Enable Overflow Interrupt: (TBIE=1)
- Clear Interrupt Flag: (TBIFG=0)
- Enable Maskable Interrupts: (GIE=1)

The main program doesn't do anything but loop forever.

The ISR toggles LED1 and clears the TBIFG flag. The ISR needs an address label to mark its starting address. The ISR needs to end with `reti` to return to the main program.

EX: TOGGLING LED1 ON TB0 OVERFLOW USING SMCLK AND A DIVIDE-BY-4 CLOCK CONFIG

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int42"
         short  ISR_TB0_Overflow
```

We initialize the vector table for Timer0_B3 using the CCS section name found in the linker file. Timer0_B3 has two vectors addresses associated with it. We want the one that has "TB0IFG" listed in the interrupt flag column (.int42).

At this vector address, we want to download the starting address of the ISR.

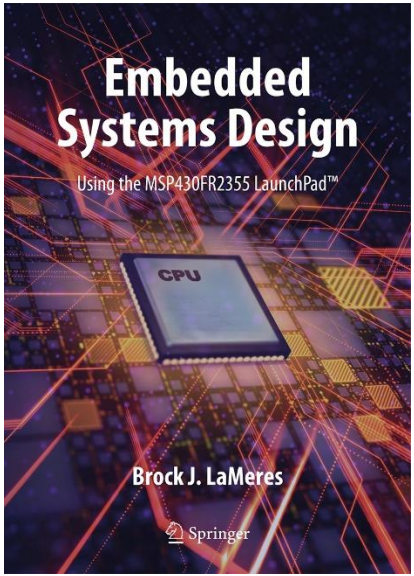
Step 3: Debug your program and run it. You will see LED1 toggle on and off every 262 ms.

Did it work? Did you see LED1 toggle every 262 ms? The ISR will trigger ~4 times per second, which means LED1 will turn on ~2 times per second.

Embedded Systems Design

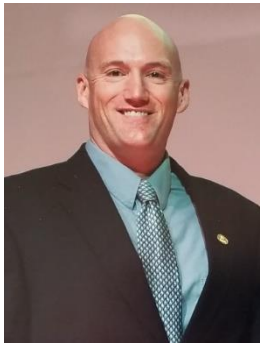
Chapter 12: Introduction to Timers

12.2 Timer Overflows on the MSP430FR2355: Example – Toggling LED1 on Timer Overflow (SMCLK, Div4)



Subscribe

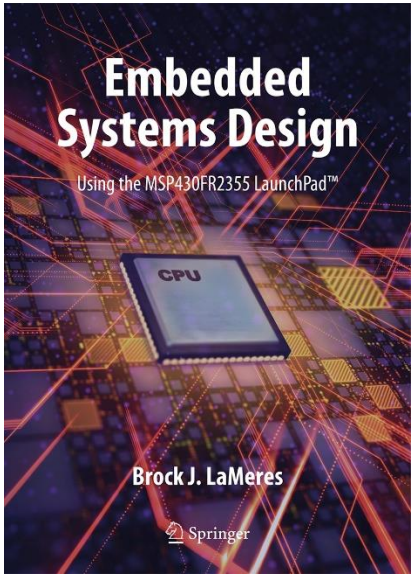
www.youtube.com/c/DigitalLogicProgramming_LaMeres

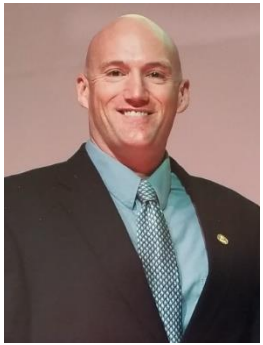


Embedded Systems Design

Chapter 12: Introduction to Timers

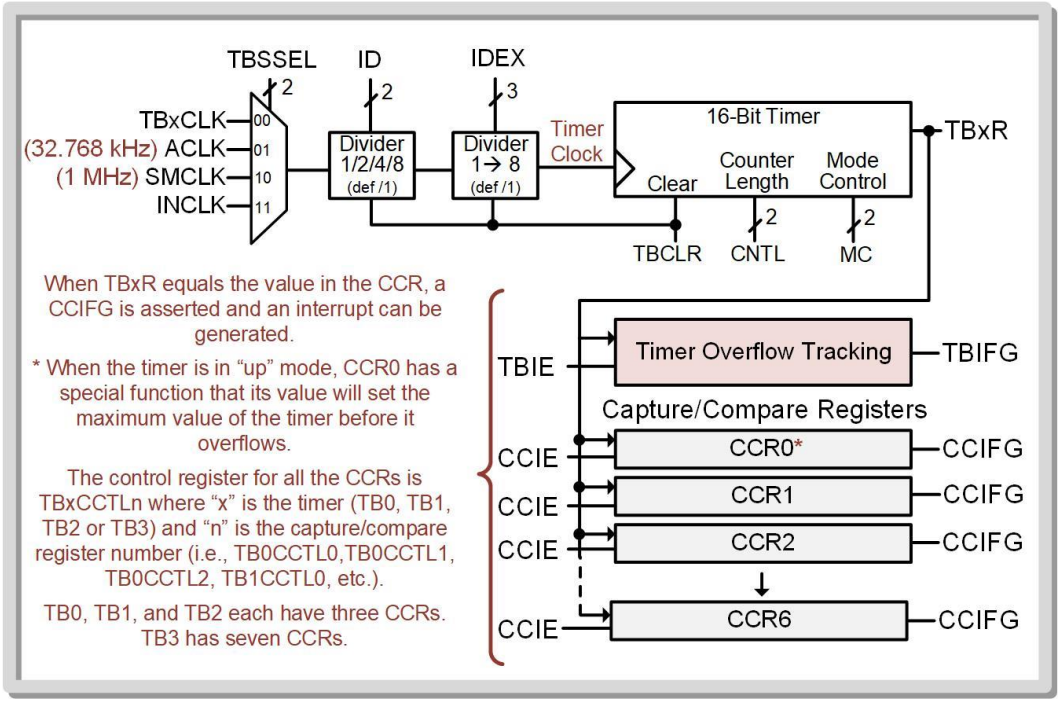
12.3 Timer Compares on the MSP430FR2355





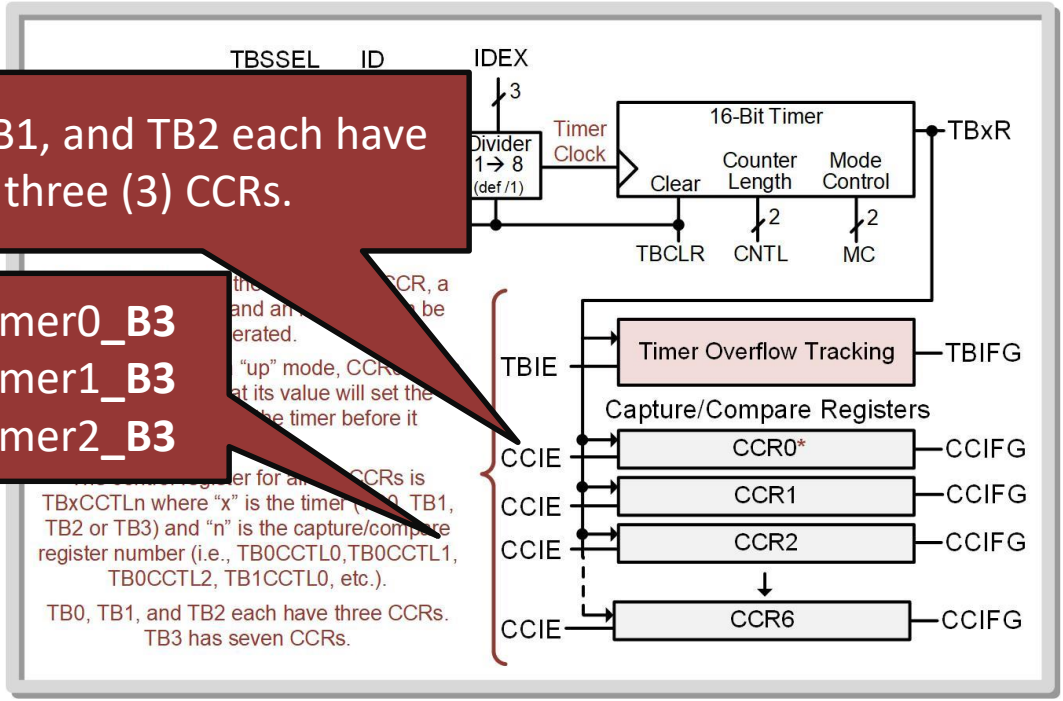
TIMER COMPARES ON THE MSP430FR2355

- **Timer Compare** – triggers an event when the main timer value equals a value stored in one of the MSP430's **capture/compare registers (CCR)**.



TIMER COMPARES ON THE MSP430FR2355

- **Timer compare** – triggers an event when the main timer value equals a value stored in one of the MSP430's **capture/compare registers (CCR)**.

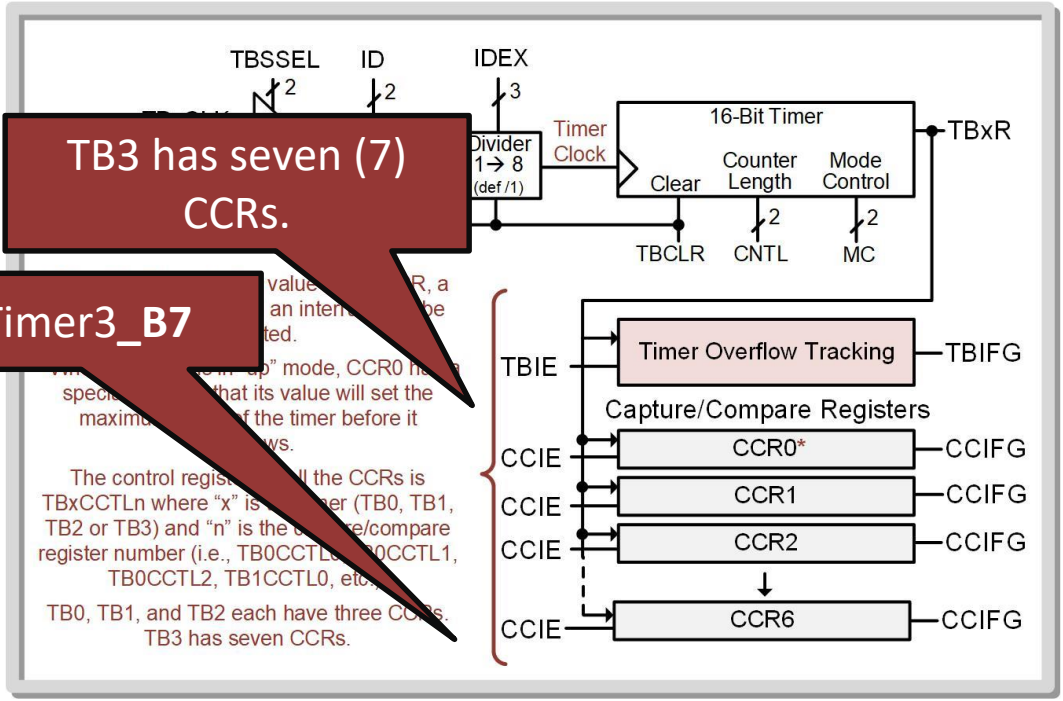


TB0, TB1, and TB2 each have three (3) CCRs.

i.e., Timer0_**B3**
i.e., Timer1_**B3**
i.e., Timer2_**B3**

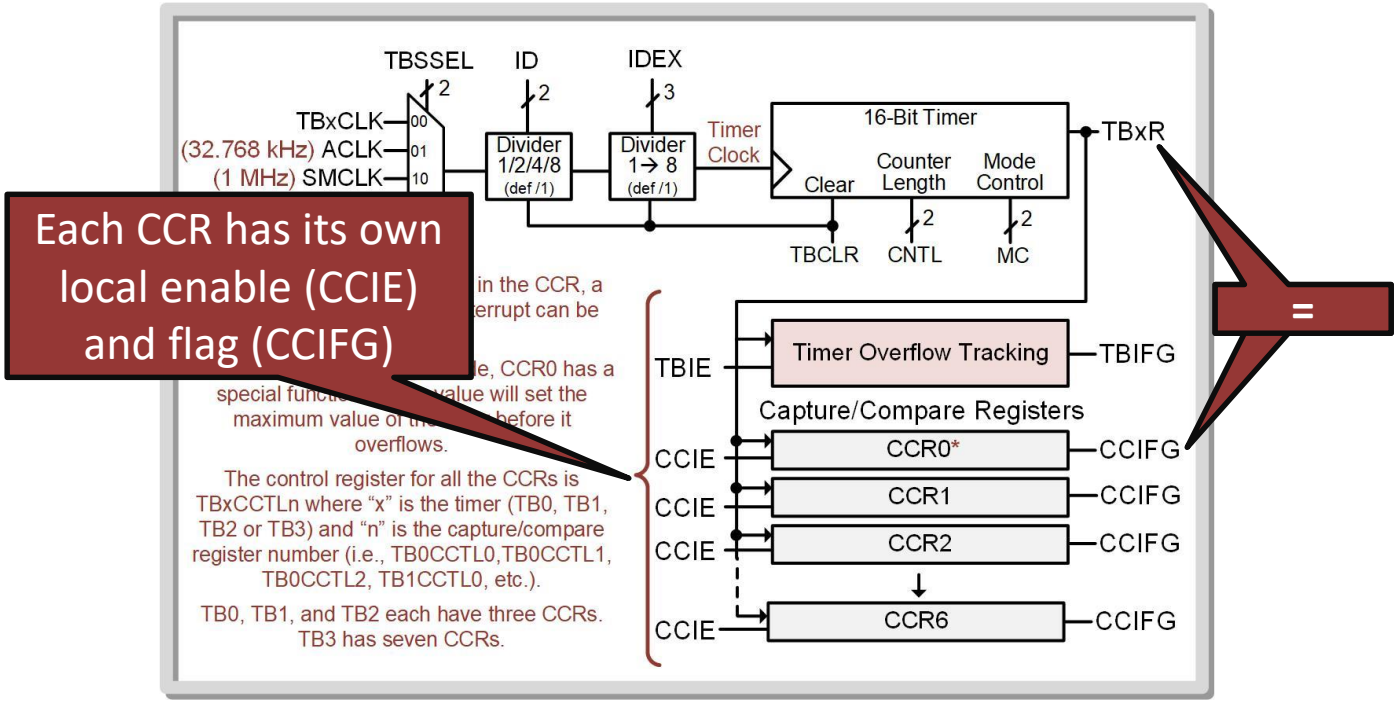
TIMER COMPARES ON THE MSP430FR2355

- **Timer compare** – triggers an event when the main timer value equals a value stored in one of the MSP430's **capture/compare registers (CCR)**.



TIMER COMPARES ON THE MSP430FR2355

- When TBxR and CCR values match, the CCR will assert a flag (CCIFG) and can trigger an interrupt if enabled.



TIMER COMPARES ON THE MSP430FR2355

- The interrupt sources share the two interrupt vectors for each timer.

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDRESS	CCS SECTION
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	.reset
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	.int45
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	.int43
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	.int42
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	.int41
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIFG2, TB1IFG (TB1IV)	Maskable	FFF2h	.int40
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	.int39
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIFG2, TB2IFG (TB2IV)	Maskable	FFEEh	.int38
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	.int37
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	.int36
RTC counter	RTCIFG	Maskable	FFE8h	.int35
Watchdog interval	WDTIFG	Maskable	FFE6h	.int34

TIMER COMPARES ON THE MSP430FR2355

- The interrupt sources share the two interrupt vectors for each timer.

Notice the CCR0 flags have dedicated vectors?

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDRESS	CCS SECTION
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	.reset
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	.int45
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	.int43
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	.int42
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	.int41
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIFG2, TB1IFG (TB1IV)	Maskable	FFF2h	.int40
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	.int39
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIFG2, TB2IFG (TB2IV)	Maskable	FFEEh	.int38
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	.int37
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	.int36
RTC counter	RTCIFG	Maskable	FFE8h	.int35
Watchdog interval	WDTIFG	Maskable	FFE6h	.int34

TIMER COMPARES ON THE MSP430FR2355

- The interrupt sources share the two interrupt vectors for each timer.

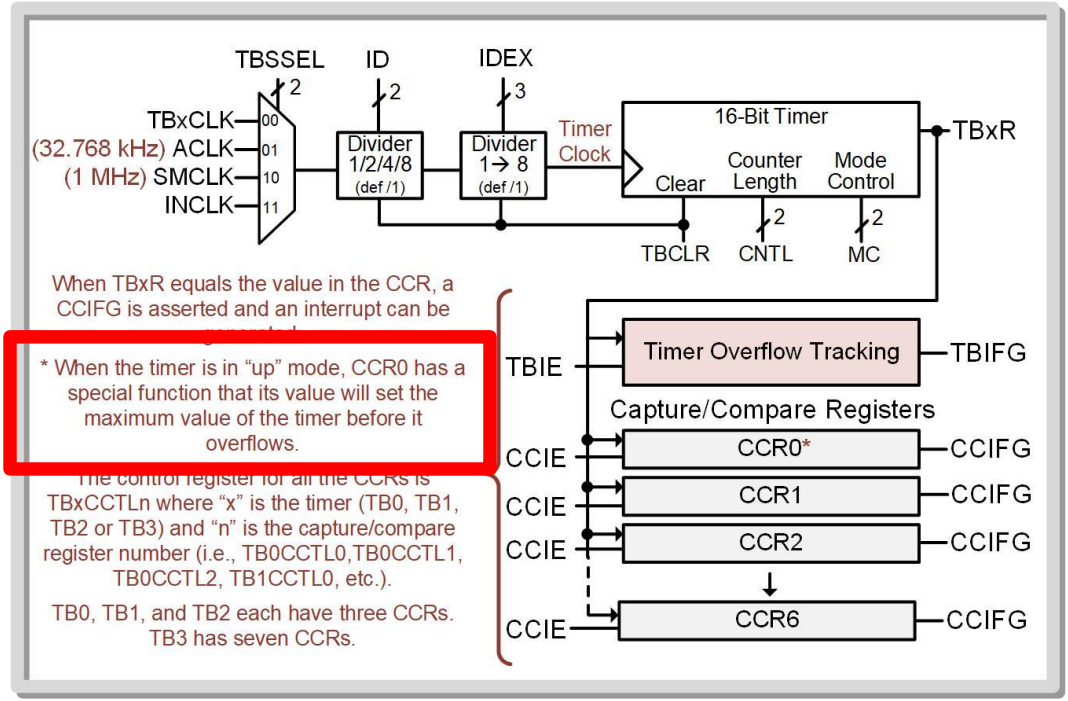
Notice the CCR0 flags have dedicated vectors?

They have "special" behavior when in UP mode.

INTERRUPT SOURCE	INTERRUPT FLAG			
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC			
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG			
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Timer0_B3	TB0CCR0 CCIFG0		FFF8h	.int43
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	[illegible]able	FFF6h	.int42
Timer1_B3	TB1CCR0 CCIFG0	[illegible]askable	FFF4h	.int41
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIF[illegible], TB1IFG (TB1IV)	Maskable	FFF2h	.int40
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	.int39
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIF[illegible]2, TB2IFG (TB2IV)	Maskable	FFEEh	.int38
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	.int37
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	.int36
RTC counter	RTCIFG	Maskable	FFE8h	.int35
Watchdog interval	WDTIFG	Maskable	FFE6h	.int34

TIMER COMPARES ON THE MSP430FR2355

- When in UP mode, CCR0 sets the maximum overflow count value for the timer.



TIMER COMPARES ON THE MSP430FR2355

- Each CCR has its own register to hold its compare value – **Timer B Capture/Compare Register (TBxCCRn)**

Table 14-5. Timer_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TBxCTL	Timer_B Control	Read/write	Word	0000h	Section 14.3.1
02h	TBxCCTL0	Timer_B Capture/Compare Control 0	Read/write	Word	0000h	Section 14.3.3
04h	TBxCCTL1	Timer_B Capture/Compare Control 1	Read/write	Word	0000h	Section 14.3.3
06h	TBxCCTL2	Timer_B Capture/Compare Control 2	Read/write	Word	0000h	Section 14.3.3
08h	TBxCCTL3	Timer_B Capture/Compare Control 3	Read/write	Word	0000h	Section 14.3.3
0Ah	TBxCCTL4	Timer_B Capture/Compare Control 4	Read/write	Word	0000h	Section 14.3.3
0Ch	TBxCCTL5	Timer_B Capture/Compare Control 5	Read/write	Word	0000h	Section 14.3.3
0Eh	TBxCCTL6	Timer_B Capture/Compare Control 6	Read/write	Word	0000h	Section 14.3.3
10h	TBxR	Timer_B Counter	Read/write	Word	0000h	Section 14.3.2
12h	TBxCCR0	Timer_B Capture/Compare 0	Read/write	Word	0000h	Section 14.3.4
14h	TBxCCR1	Timer_B Capture/Compare 1	Read/write	Word	0000h	Section 14.3.4
16h	TBxCCR2	Timer_B Capture/Compare 2	Read/write	Word	0000h	Section 14.3.4
18h	TBxCCR3	Timer_B Capture/Compare 3	Read/write	Word	0000h	Section 14.3.4
1Ah	TBxCCR4	Timer_B Capture/Compare 4	Read/write	Word	0000h	Section 14.3.4
1Ch	TBxCCR5	Timer_B Capture/Compare 5	Read/write	Word	0000h	Section 14.3.4
1Eh	TBxCCR6	Timer_B Capture/Compare 6	Read/write	Word	0000h	Section 14.3.4
2Eh	TBxIV	Timer_B Interrupt Vector	Read only	Word	0000h	Section 14.3.5
20h	TBxEX0	Timer_B Expansion 0	Read/write	Word	0000h	Section 14.3.6

TIMER COMPARES ON THE MSP430FR2355

- Each CCR has its own control register called **Timer B Capture/Compare Control Register (TBxCCTLn)**

Table 14-5. Timer_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TBxCTL	Timer_B Control	Read/write	Word	0000h	Section 14.3.1
02h	TBxCCTL0	Timer_B Capture/Compare Control 0	Read/write	Word	0000h	Section 14.3.3
04h	TBxCCTL1	Timer_B Capture/Compare Control 1	Read/write	Word	0000h	Section 14.3.3
06h	TBxCCTL2	Timer_B Capture/Compare Control 2	Read/write	Word	0000h	Section 14.3.3
08h	TBxCCTL3	Timer_B Capture/Compare Control 3	Read/write	Word	0000h	Section 14.3.3
0Ah	TBxCCTL4	Timer_B Capture/Compare Control 4	Read/write	Word	0000h	Section 14.3.3
0Ch	TBxCCTL5	Timer_B Capture/Compare Control 5	Read/write	Word	0000h	Section 14.3.3
0Eh	TBxCCTL6	Timer_B Capture/Compare Control 6	Read/write	Word	0000h	Section 14.3.3
10h	TBxR	Timer_B Counter	Read/write	Word	0000h	Section 14.3.2
12h	TBxCCR0	Timer_B Capture/Compare 0	Read/write	Word	0000h	Section 14.3.4
14h	TBxCCR1	Timer_B Capture/Compare 1	Read/write	Word	0000h	Section 14.3.4
16h	TBxCCR2	Timer_B Capture/Compare 2	Read/write	Word	0000h	Section 14.3.4
18h	TBxCCR3	Timer_B Capture/Compare 3	Read/write	Word	0000h	Section 14.3.4
1Ah	TBxCCR4	Timer_B Capture/Compare 4	Read/write	Word	0000h	Section 14.3.4
1Ch	TBxCCR5	Timer_B Capture/Compare 5	Read/write	Word	0000h	Section 14.3.4
1Eh	TBxCCR6	Timer_B Capture/Compare 6	Read/write	Word	0000h	Section 14.3.4
2Eh	TBxIV	Timer_B Interrupt Vector	Read only	Word	0000h	Section 14.3.5
20h	TBxEX0	Timer_B Expansion 0	Read/write	Word	0000h	Section 14.3.6

TIMER COMPARES ON THE MSP430FR2355

Timer_B Capture/Compare Control Register n (TBxCCTLn)

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CM		CCIS		SCS	CLLD		CAP	OUTMOD			CCIE	CCI	OUT	COV	CCIFG
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:14	CM	Capture Mode 00=No Capture 01=Capture on Rising Edge 10=Capture on Falling Edge 11=Capture on Both Edges
13:12	CCIS	Capture/Compare Input Select 00=CCIxA 01=CCIxB 10=GND 11=VCC
11	SCS	Synchronize Capture Source (0=Asynchronous Capture; 1=Synchronous Capture).
10:9	CLLD	Compare Latch Load 00=TBxCLn Loads on Write to TBxCCRn. 01=TBxCLn Loads when TBxR Counts to 0. 10=TBxCLn Loads when TBxR Counts to 0 (up or continuous mode); TBxCLn Loads when TBxR Counts to TBxCL0 or 0 (up/down mode). 11=TBxCLn Loads when TBxR Counts to TBxCLn.
8	CAP	Capture Mode (0=Compare Mode; 1=Capture Mode).
7:5	OUTMOD	Output Mode 000=OUT bit value 001=Set 010=Toggle/Reset 011=Set/Reset 100=Toggle 101=Reset 110=Toggle/Set 111=Reset/Set
4	CCIE	Capture/Compare Interrupt Enable (0=IRQ Disabled; 1=IRQ Enabled).
3	CCI	Capture/Compare Input.
2	OUT	Output Level (0=Low; 1=High).
1	COV	Capture Overflow (0=No overflow occurred; 1=Overflow occurred).
0	CCIFG	Capture/Compare Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending).

TIMER COMPARES ON THE MSP430FR2355

- **Timer_B x Interrupt Vector Register (TBxIV)** – holds the same code that represents the highest CCR IRG that has occurred.

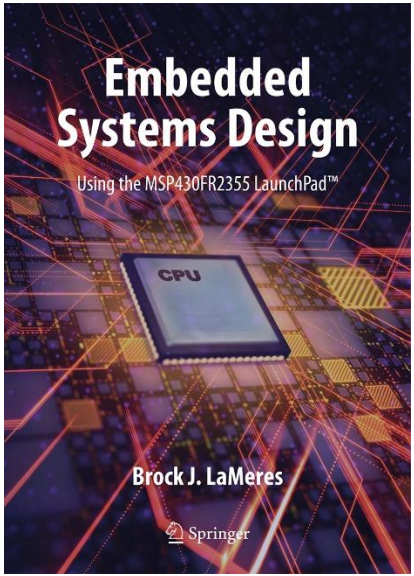
Table 14-5. Timer_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TBxCTL	Timer_B Control	Read/write	Word	0000h	Section 14.3.1
02h	TBxCCTL0	Timer_B Capture/Compare Control 0	Read/write	Word	0000h	Section 14.3.3
04h	TBxCCTL1	Timer_B Capture/Compare Control 1	Read/write	Word	0000h	Section 14.3.3
06h	TBxCCTL2	Timer_B Capture/Compare Control 2	Read/write	Word	0000h	Section 14.3.3
08h	TBxCCTL3	Timer_B Capture/Compare Control 3	Read/write	Word	0000h	Section 14.3.3
0Ah	TBxCCTL4	Timer_B Capture/Compare Control 4	Read/write	Word	0000h	Section 14.3.3
0Ch	TBxCCTL5	Timer_B Capture/Compare Control 5	Read/write	Word	0000h	Section 14.3.3
0Eh	TBxCCTL6	Timer_B Capture/Compare Control 6	Read/write	Word	0000h	Section 14.3.3
10h	TBxR	Timer_B Counter	Read/write	Word	0000h	Section 14.3.2
12h	TBxCCR0	Timer_B Capture/Compare 0	Read/write	Word	0000h	Section 14.3.4
14h	TBxCCR1	Timer_B Capture/Compare 1	Read/write	Word	0000h	Section 14.3.4
16h	TBxCCR2	Timer_B Capture/Compare 2	Read/write	Word	0000h	Section 14.3.4
18h	TBxCCR3	Timer_B Capture/Compare 3	Read/write	Word	0000h	Section 14.3.4
1Ah	TBxCCR4	Timer_B Capture/Compare 4	Read/write	Word	0000h	Section 14.3.4
1Ch	TBxCCR5	Timer_B Capture/Compare 5	Read/write	Word	0000h	Section 14.3.4
1Eh	TBxCCR6	Timer_B Capture/Compare 6	Read/write	Word	0000h	Section 14.3.4
2Eh	TBxIV	Timer_B Interrupt Vector	Read only	Word	0000h	Section 14.3.5
20h	TBxEX0	Timer_B Expansion 0	Read/write	Word	0000h	Section 14.3.6

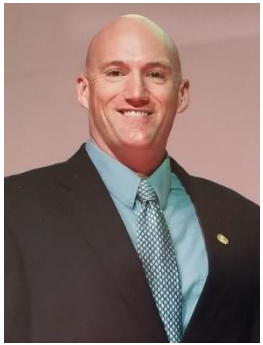
Embedded Systems Design

Chapter 12: Introduction to Timers

12.3 Timer Compares on the MSP430FR2355



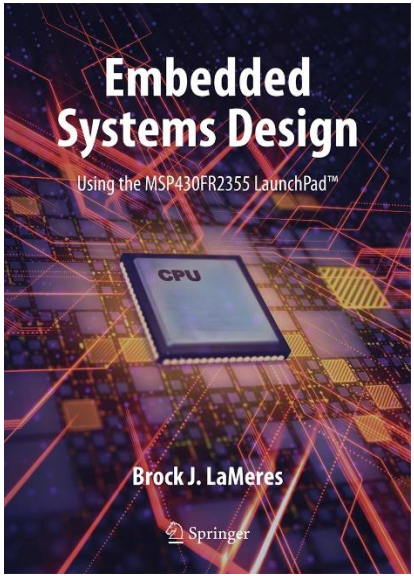
www.youtube.com/c/DigitalLogicProgramming_LaMeres

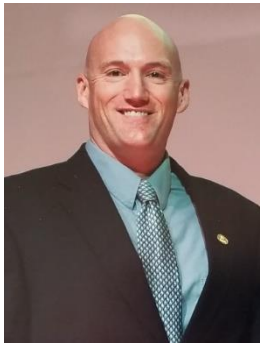


Brock J. LaMeres, Ph.D.

CHAPTER 12: INTRODUCTION TO TIMERS

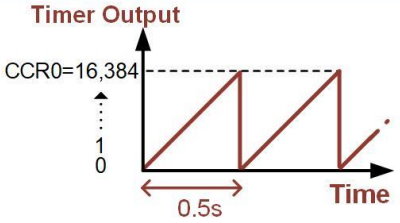
12.3 TIMER COMPARES ON THE MSP430FR2355 – EXAMPLE: TOGGLING LED1 EVERY 0.5 SECONDS USING CCR





EX: TOGGLING LED1 USING A CCR0 COMPARE

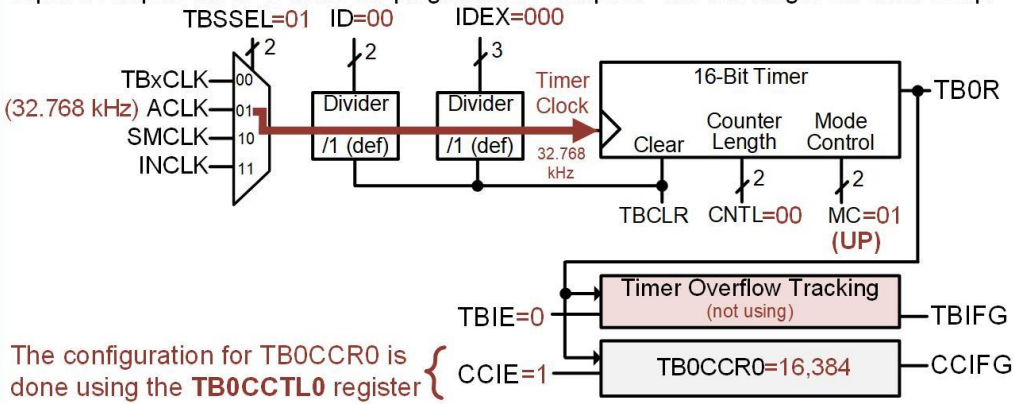
Let's design a program that will toggle LED1 every 0.5 s using a timer compare. We will use **ACLK** without any dividers. We will use CCR0, which allows us to simply enter a new maximum value for TB0R and then generate an IRQ upon overflow. First, let's calculate the value to put into CCR0 to achieve an overflow at 0.5 s.



$\Delta t = T \cdot N$

$0.5\ s = (1/32.768k) \cdot N \rightarrow N=16,384$

Next, we need to decide on the settings for TB0. We want the clock source to be ACLK with both dividers set to 1. We also want the counter length to be 16-bits and the counting mode to be **up** so that CCR0 is used as the maximum TB0R value. We then need to load CCR0 with 16,384 and assert CCIE. We can use the default values for the dividers, counter length, capture/compare mode to make our program more compact. The following is our timer setup:



The configuration for TB0CCR0 is done using the **TB0CCTL0** register

EX: TOGGLING LED1 USING A CCR0 COMPARE

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_Compare_CCR0

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.”

EX: TOGGLING LED1 USING A CCR0 COMPARE

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0
;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__ACLK, &TB0CTL
        bis.w   #MC__UP, &TB0CTL

        mov.w   #16384, &TB0CCR0
        bis.w   #CCIE, &TB0CCTL0
        bic.w   #CCIFG, &TB0CCTL0

        bis.w   #GIE, SR
main:
        jmp     main
;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_CCR0:
        xor.b   #BIT0, &P1OUT
        bic.w   #CCIFG, &TB0CCTL0
        reti
```

- Setup LED1 to be an output: (P1DIR.0=1)
- Put LED1's initial value at zero: (P1OUT.0=0)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select ACLK as Timer Source: (TBSSEL=01)
- Choose **UP** Counting: (MC=01)
- Initialize CCR0 to 16,384.
- Enable Capture/Compare IRQ: (CCIE=1)
- Clear Interrupt Flag: (CCIFG=0)
- Enable Maskable Interrupts: (GIE=1)

The main program doesn't do anything but loop forever.

The ISR toggles LED1 and clears the CCIFG flag.

EX: TOGGLING LED1 USING A CCR0 COMPARE

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int43"
         short  ISR_TB0_CCR0
```

We initialize the vector table for Timer0_B3 using the CCS section name found in the linker file. Timer0_B3 has two vector addresses associated with it. We want the one that has "CCIFG0" listed in the interrupt flag column (.int43).

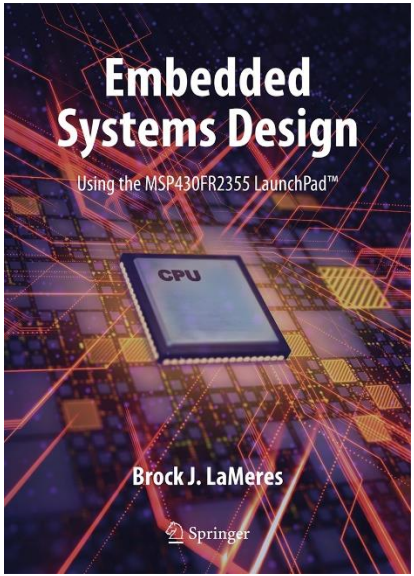
At this vector address, we want to download the starting address of the ISR (i.e., ISR_TB0_CCR0).

Step 3: Debug your program and run it. You will see LED1 toggle on and off every 0.5 seconds.

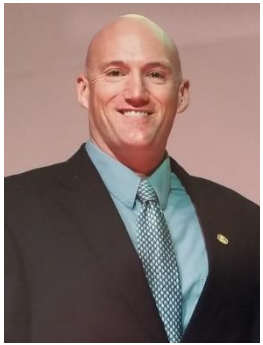
Did it work? Did you see LED1 toggle every 0.5 seconds? If it didn't, you should inspect the TB0 configuration settings in the Register Viewer.

CHAPTER 12: INTRODUCTION TO TIMERS

12.3 TIMER COMPARES ON THE MSP430FR2355 – EXAMPLE: TOGGLING LED1 EVERY 0.5 SECONDS USING CCR



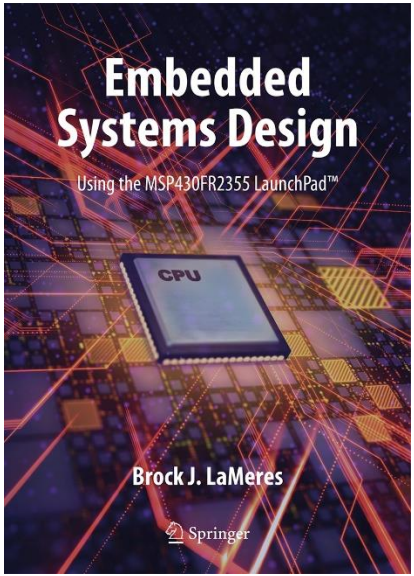
www.youtube.com/c/DigitalLogicProgramming_LaMeres

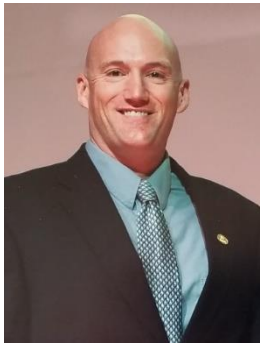


Embedded Systems Design

Chapter 12: Introduction to Timers

12.4 Creating Pulse Width Modulated Signals using Timer Compares

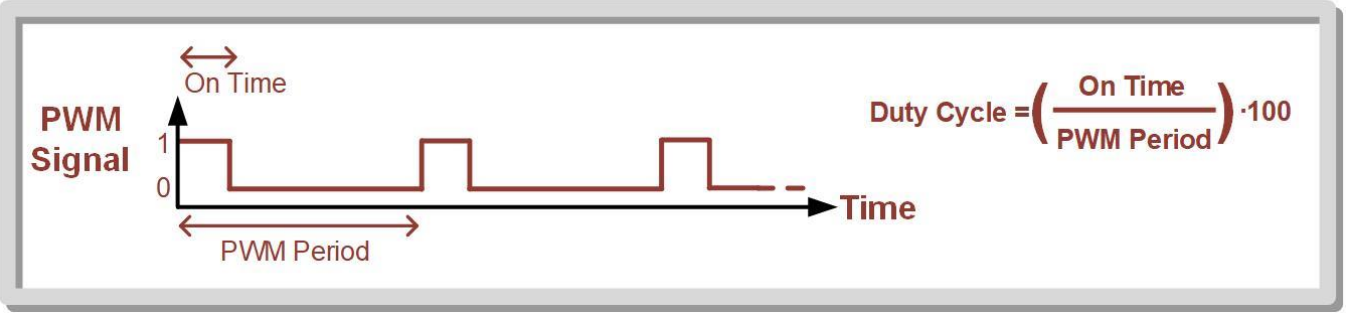




Brock J. LaMeres, Ph.D.

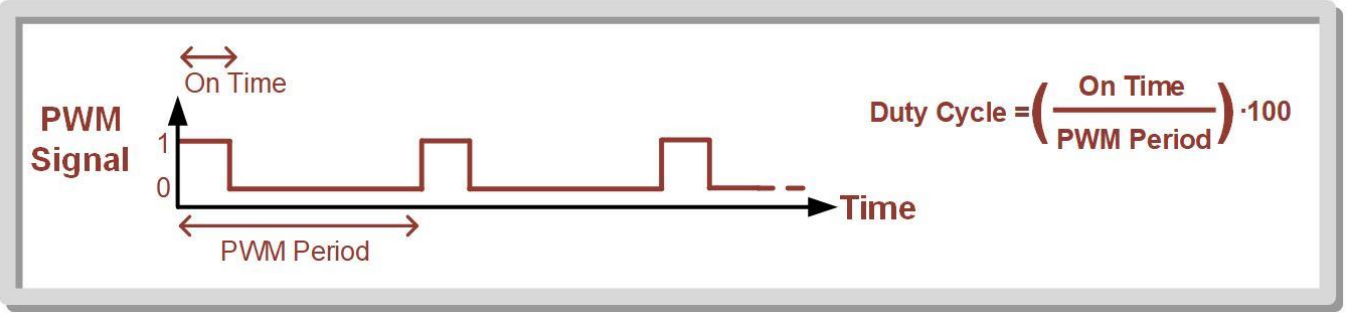
Creating Pulse Width Modulated Signals using Timer Compares

- **Pulse Width Modulated (PWM) signal** – a popular type of electrical signal that can be generated with timer compares.
- PWM signals are used for motor control, dimming LEDs, and to communicate information.
- **On Time** – the amount of time that a periodic signal is high
- **Duty cycle** – the percentage of the period that is high.

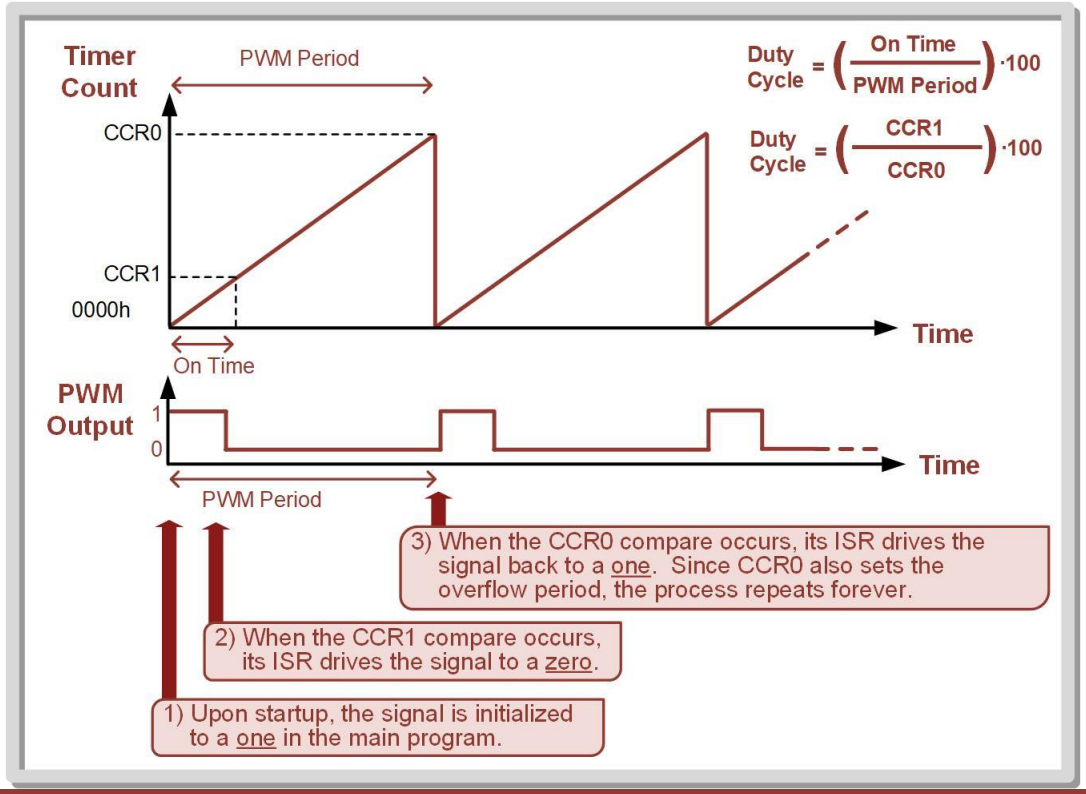


CREATING PULSE WIDTH MODULATED SIGNALS USING TIMER COMPARES

- A PWM can be created using the Timer_B system by setting up two compare registers – CCR0 and CCR1.
- CCR0 – holds the count value corresponding to the period of the PWM signal.
- CCR1 - holds the count value corresponding to the amount of on time that is desired.



CREATING PULSE WIDTH MODULATED SIGNALS USING TIMER COMPARES



EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

Let's design a program that will drive LED1 with a PWM signal with a period of 1 second and a duty cycle of 5%. This will result in a momentary flash on the LED every second. 5% of 1 second is only 50 ms, but it is still long enough to see with the naked eye. We will use ACLK without dividers to create a timer clock of 32.768 kHz. We will use a CCR0 compare to set the period of the PWM to 1 second. We will use a CCR1 compare to set the duty cycle to 5%. We will initialize LED1 to a one upon startup. When the first compare on CCR1 occurs after 50 ms, we will drive LED1 to a zero. When the second compare on CCR0 occurs after 1 second, we will drive LED1 to a zero. Since CCR0 also sets the overflow period, as long as we running in "up" mode, this will repeat forever. First, let's calculate the values to put into CCR0 to achieve a period of 1 second and the value to put into CCR1 to achieve a duty cycle of 5%=50ms..

CCR0

$\Delta t = T \cdot N$

$1\ s = (1/32.768k) \cdot N$

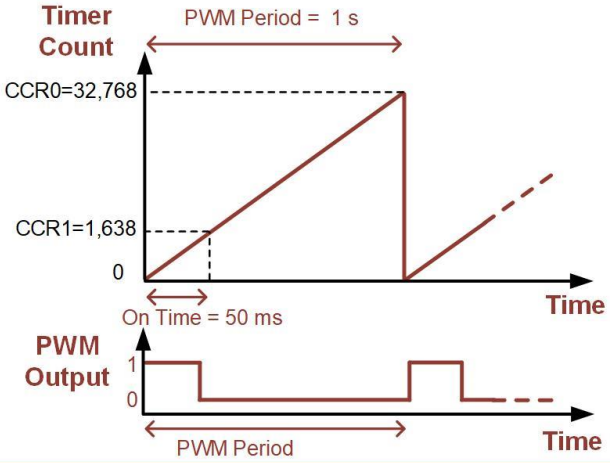
$\rightarrow N=32,768$

CCR1

$\Delta t = T \cdot N$

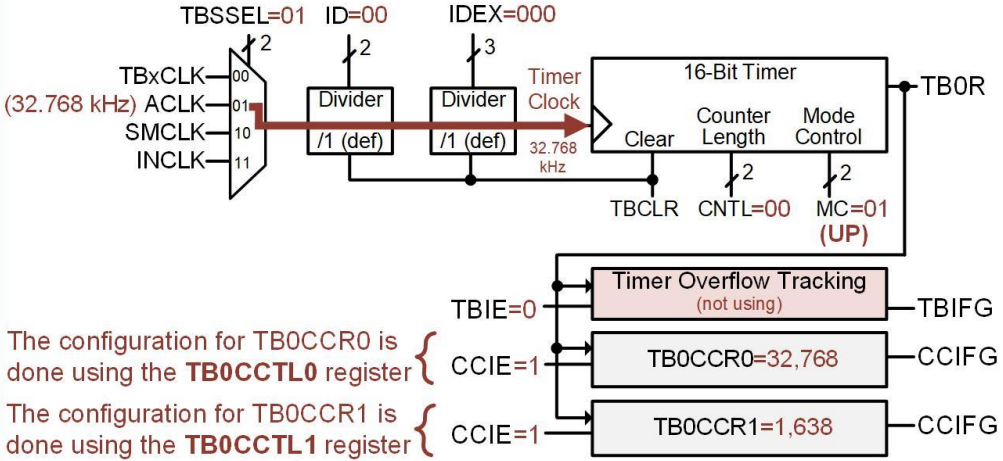
$50m = (1/32.768k) \cdot N$

$\rightarrow N=1,638$



EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

Next, we need to decide on the settings for TB0. We want the clock source to be ACLK with both dividers set to 1. We also want the counter length to be 16-bits and the counting mode to be **up** so that CCR0 is used as the maximum TB0R value. We then need to load CCR0 with 32,768 and CCR1 with 1,638. We also need to enable the interrupt for CCR0 and the interrupt for CCR1. These each have their own CCIE bit in their respective TB0CCTLn control registers. These two IRQs exist on separate vector addresses, so each can generate its own dedicated interrupt. The following is our timer setup:



EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDRESS	CCS SECTION
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	.reset
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	.int45
User NMI	NMIIFG, OFIFG	Non-Maskable	FFFAh	.int44
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	.int43
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	.int42
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	.int41
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIFG2, TB1IFG (TB1IV)	Maskable	FFF2h	.int40
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	.int39
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIFG2, TB2IFG (TB2IV)	Maskable	FFEEh	.int38
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	.int37
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	.int36
RTC counter	RTCIFG	Maskable	FFE8h	.int35
Watchdog interval	WDTIFG	Maskable	FFE6h	.int34

EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

Step 1: Create a new Empty Assembly-only CCS project titled:

Asm_Timers_Compare_CCR1_n_CCR0_PWM

Step 2: Type in the following code into the main.asm file where the comments say “Main loop here.”

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

```
init:
;-- Setup LED1
        bis.b   #BIT0, &P1DIR
        bis.b   #BIT0, &P1OUT
        bic.b   #LOCKLPM5, &PM5CTL0

;-- Setup Timer B0
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__ACLK, &TB0CTL
        bis.w   #MC__UP, &TB0CTL

;-- Setup Compare Registers
        mov.w   #32768, &TB0CCR0
        mov.w   #1638,  &TB0CCR1

        bis.w   #CCIE, &TB0CCTL0
        bic.w   #CCIFG, &TB0CCTL0

        bis.w   #CCIE, &TB0CCTL1
        bic.w   #CCIFG, &TB0CCTL1

        bis.w   #GIE, SR
main:
        jmp     main
;-----------------------------------
; Interrupt Service Routines
;-----------------------------------
ISR_TB0_CCR1:
        bic.b   #BIT0, &P1OUT
        bic.w   #CCIFG, &TB0CCTL1
        reti

ISR_TB0_CCR0:
        bis.b   #BIT0, &P1OUT
        bic.w   #CCIFG, &TB0CCTL0
        reti
```

- Setup LED1 to be an output: (P1DIR.0=1)
- Set LED1's initial value to **one**: (P1OUT.0=1)
- Clear LOCKLPM5 Bit.

- Clear Timer & Dividers: (TBCLR=1)
- Select ACLK as Timer Source: (TBSSEL=01)
- Choose **UP** Counting: (MC=01)

- Initialize CCR0 to 32,768.
- Initialize CCR1 to 1,638.

- Enable TB0CCR0 Interrupt.
- Clear TB0CCR0 Flag.

- Enable TB0CCR1 Interrupt.
- Clear TB0CCR1 Flag.

- Enable Maskable Interrupts.

The main loop doesn't do anything but loop forever.

The CCR1 IRQ will trigger first. It needs to drive LED1 to a zero and clear the CCR1 flag.

The CCR0 IRQ will trigger second . It needs to drive LED1 back to a one and clear the CCR0 flag.

EX: FLASHING LED1 WITH A 5% DUTY CYCLE PWM USING MULTIPLE COMPARES

```
;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int43"
        .short  ISR_TB0_CCR0

        .sect   ".int42"
        .short  ISR_TB0_CCR1
```

The TB0CCR1 CCIFG1 interrupt vector is FFF6h, which uses the literal ".int43".

The TB0CCR0 CCIFG0 interrupt vector is FFF8h, which uses the literal ".int42".

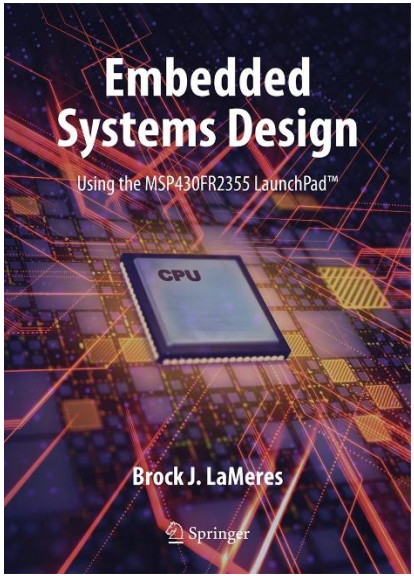
Step 3: Debug your program and run it. You will see LED1 flash briefy every 1 second.

Did it work? Did you see LED1 flash quickly every 1 second? If it didn't, you should inspect the TB0 and TB1 configuration settings in the Register Viewer.

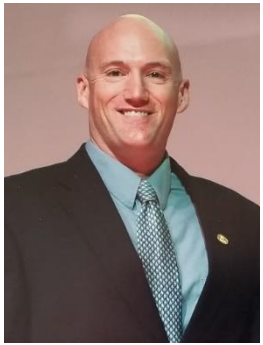
Embedded Systems Design

Chapter 12: Introduction to Timers

12.4 Creating Pulse Width Modulated Signals using Timer Compares



www.youtube.com/c/DigitalLogicProgramming_LaMeres

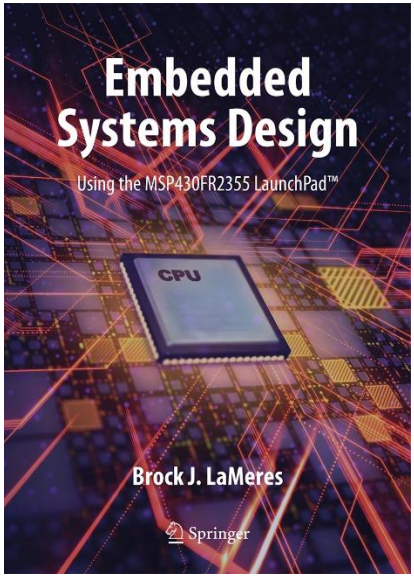


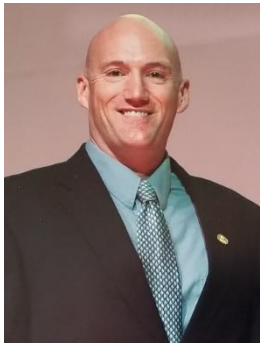
Brock J. LaMeres, Ph.D.

Embedded Systems Design

Chapter 12: Introduction to Timers

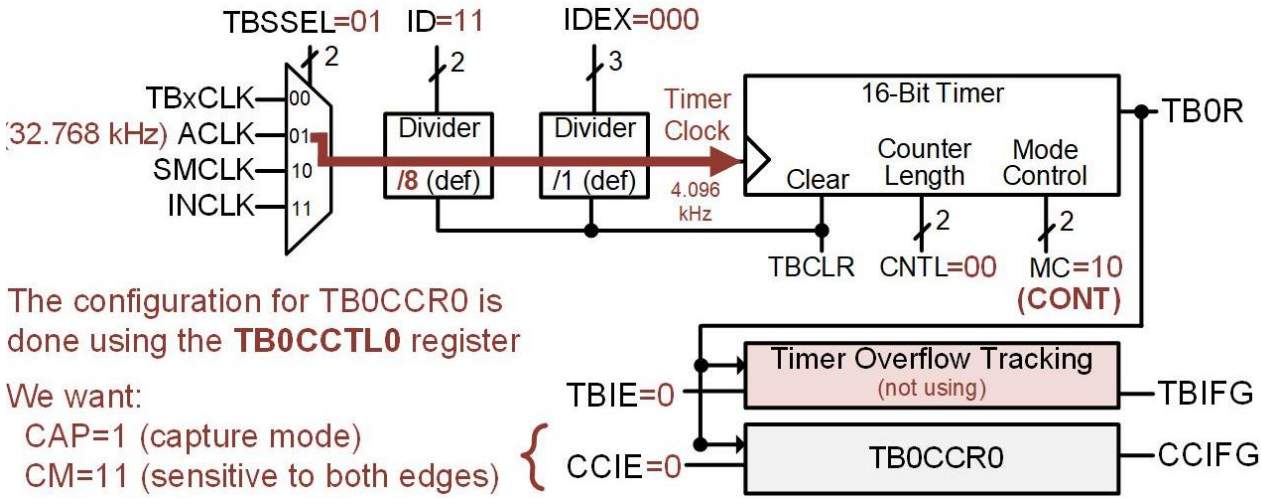
12.5 Timer Captures on the MSP430FR2355





Brock J. LaMeres, Ph.D.

TIMER CAPTURES ON THE MSP430FR2355



The configuration for TB0CCR0 is done using the **TB0CCTL0** register

We want:

- CAP=1 (capture mode)
- CM=11 (sensitive to both edges)

TIMER CAPTURES ON THE MSP430FR2355

- **Timer Capture** – store the current value of the timer into one of the capture/compare registers upon a triggering event.
- This function can be used to measure time between events, both externally or internally.
- The capture mode is selected with **CAP=1** in the TBxCCTLn register.
- A transition on the internal signal **Capture/Compare Input (CCI)** will cause a capture.

TIMER CAPTURES ON THE MSP430FR2355

- The edge polarity on CCI that triggers the capture is dictated by the **Capture Mode (CM)** bits within TBxCCTLn and supports, rising edge, falling edge, or both edge sensitivity.

- The source for CCI is dictated by the **Capture/Compare Input Select (CCIS)** bits within TBxCCTLn.

- A capture event can also be triggered by software by manually creating an edge on CCI.

TIMER CAPTURES ON THE MSP430FR2355

Timer_B Capture/Compare Control Register n (TBxCCTLn)

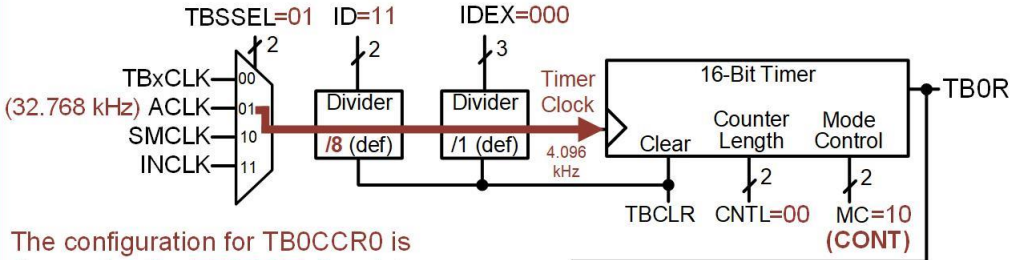
p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CM		CCIS		SCS	CLLD		CAP	OUTMOD			CCIE	CCI	OUT	COV	CCIFG
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:14	CM	Capture Mode 00=No Capture 01=Capture on Rising Edge 10=Capture on Falling Edge 11=Capture on Both Edges
13:12	CCIS	Capture/Compare Input Select 00=CCIxA 01=CCIxB 10=GND 11=VCC
11	SCS	Synchronize Capture Source (0=Asynchronous Capture; 1=Synchronous Capture).
10:9	CLLD	Compare Latch Load 00=TBxCLn Loads on Write to TBxCCRn. 01=TBxCLn Loads when TBxR Counts to 0. 10=TBxCLn Loads when TBxR Counts to 0 (up or continuous mode); TBxCLn Loads when TBxR Counts to TBxCL0 or 0 (up/down mode). 11=TBxCLn Loads when TBxR Counts to TBxCLn.
8	CAP	Capture Mode (0=Compare Mode; 1=Capture Mode).
7:5	OUTMOD	Output Mode 000=OUT bit value 001=Set 010=Toggle/Reset 011=Set/Reset 100=Toggle 101=Reset 110=Toggle/Set 111=Reset/Set
4	CCIE	Capture/Compare Interrupt Enable (0=IRQ Disabled; 1=IRQ Enabled).
3	CCI	Capture/Compare Input.
2	OUT	Output Level (0=Low; 1=High).
1	COV	Capture Overflow (0=No overflow occurred; 1=Overflow occurred).
0	CCIFG	Capture/Compare Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending).

TIMER CAPTURES ON THE MSP430FR2355

EXAMPLE: TRIGGERING A CAPTURE IN SOFTWARE (PART 1)

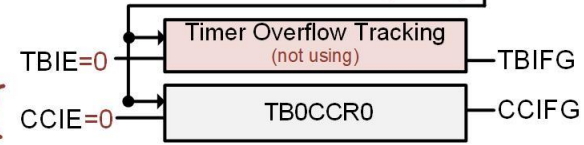
Let's design a program that trigger a capture of the timer every time SW1 is pressed. We will use ACLK with a divider of 8 in order run the timer slow enough so that we can see the count be captured using the debugger. We will setup a P4.1 interrupt so that each time SW1 is pressed, an ISR will run that manually forces a capture into CCR0. The capture will be forced by switching the capture/compare input select (CCIS) between GND and VCC in order to produce an edge to the capture system. The following is the setup of our timer.



The configuration for TB0CCR0 is done using the **TB0CCTL0** register

We want:

- CAP=1 (capture mode)
- CM=11 (sensitive to both edges)
- CCIS=10 (input = GND initially)



In our ISR, we will toggle this bit in CCIS in order to generate an edge by switching between GND→VCC or VCC→GND.

TIMER CAPTURES ON THE MSP430FR2355

EXAMPLE: TRIGGERING A CAPTURE IN SOFTWARE (PART 2)

1) Create a new Empty Assembly-only CCS project titled: **Asm_Timer_Capture_SW_CCR0.**

2) Type in the following code into the main.asm file where the comments say "Main loop here".

```
init:
;--setup Ports (LED1 and SW1)
        bis.b   #BIT0, &P1DIR
        bic.b   #BIT0, &P1OUT

        bic.b   #BIT1, &P4DIR
        bis.b   #BIT1, &P4REN
        bis.b   #BIT1, &P4OUT
        bis.b   #BIT1, &P4IES

        bic.b   #LOCKLPM5, &PM5CTL0
;--setup Port IRQ for SW1
        bis.b   #BIT1, &P4IE
        eint
        bic.b   #BIT1, &P4IFG
;--setup timer B0 to run SLOW
        bis.w   #TBCLR, &TB0CTL
        bis.w   #TBSSEL__ACLK, &TB0CTL
        bis.w   #ID__8, &TB0CTL
        bis.w   #MC__CONTINUOUS, &TB0CTL
;--setup capture
        bis.w   #CAP, &TB0CCTL0
        bis.w   #CM__BOTH, &TB0CCTL0
        bis.w   #CCIS__GND, &TB0CCTL0
;--init R4
        mov.w   #0, R4
main:
        jmp     main
;-----------------------------------
```

- Setup LED1 to be an output: (P1DIR.0=1)
- Clear LED1 initially: (P1OUT.0=0)

- Set Port Direction to Input (P4DIR.1=0)
- Enable Pull-Up/Down Resistor (P4REN.1=1)
- Configure Resistor as Pull-Up (P4OUT.1=1)
- Set IRQ Sensitivity to High-to-Low (P4IES.1=1)

- Clear LOCKLPM5 Bit.

- Clear Interrupt Flag (P4IFG.1=0)
- Assert Local Enable (P4IE.1=1)
- Assert Global Enable (GIE=1)

- Clear Timer & Dividers: (TBCLR=1)
- Select ACLK as Source: (TBSSEL=01)
- Divide-by-8 (ID=11)
- **CONTINUOUS** Mode: (MC=10)

- Put into Capture Mode: (CAP=1)
- Capture Sensitivity = Both: (CM=11)
- Capture Source Input = GND (CCIS)

We'll store the captured value into R4

The main loop doesn't do anything but loop forever.

```
; Interrupt Service Routines
;-----------------------------------
ISR_S1:
        xor.w   #CCIS0, &TB0CCTL0
        mov.w   &TB0CCR0, R4

        xor.b   #BIT0, &P1OUT
        bic.b   #BIT1, &P4IFG

        reti

;-----------------------------------
; Interrupt Vectors
;-----------------------------------
        .sect   ".reset"
        .short  RESET

        .sect   ".int22"
        .short  ISR_S1
```

The CCR0 IRQ will trigger second . It needs to drive LED1 back to a 1 and clear the CCR0 flag.

We'll use the ISR for Port 4.

3) Debug your program.

4) Set a breakpoint at the last instruction in the ISR.

5) Open the register viewer and view R4, TB0R, and TB0CCR0.

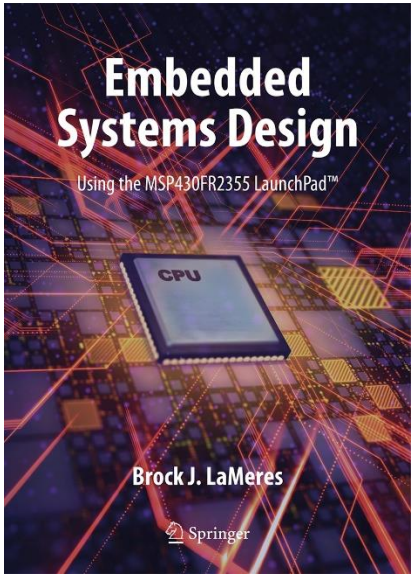
6) Run your program. Each time you press S1 it will break and you will see the timer value stored into R4 and TBCCR0.

Did it work? Can you see the timer value be stored into TB0CCR0?

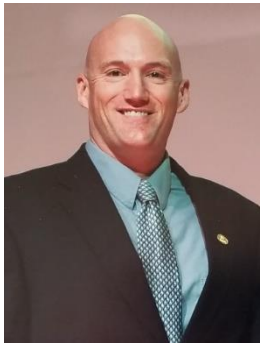
Embedded Systems Design

Chapter 12: Introduction to Timers

12.5 Timer Captures on the MSP430FR2355



www.youtube.com/c/DigitalLogicProgramming_LaMeres



Brock J. LaMeres, Ph.D.