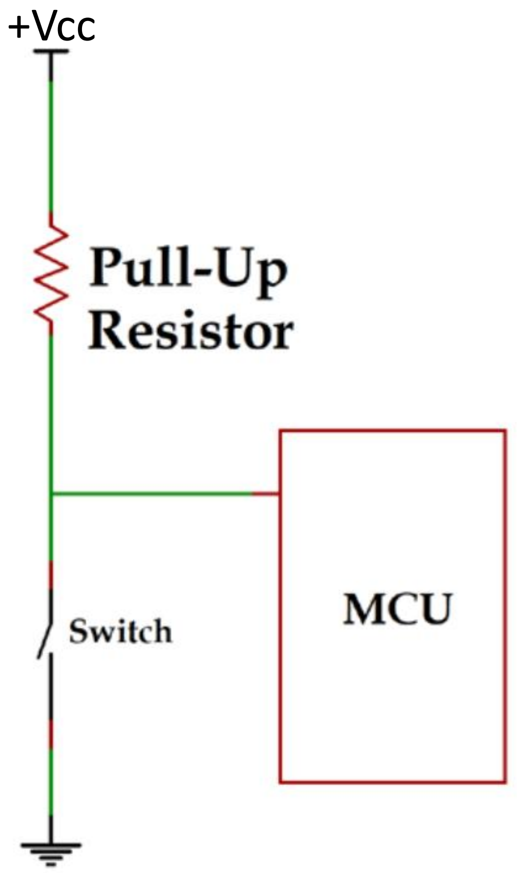
EEE 204
Pull up/down Resistors and Timers

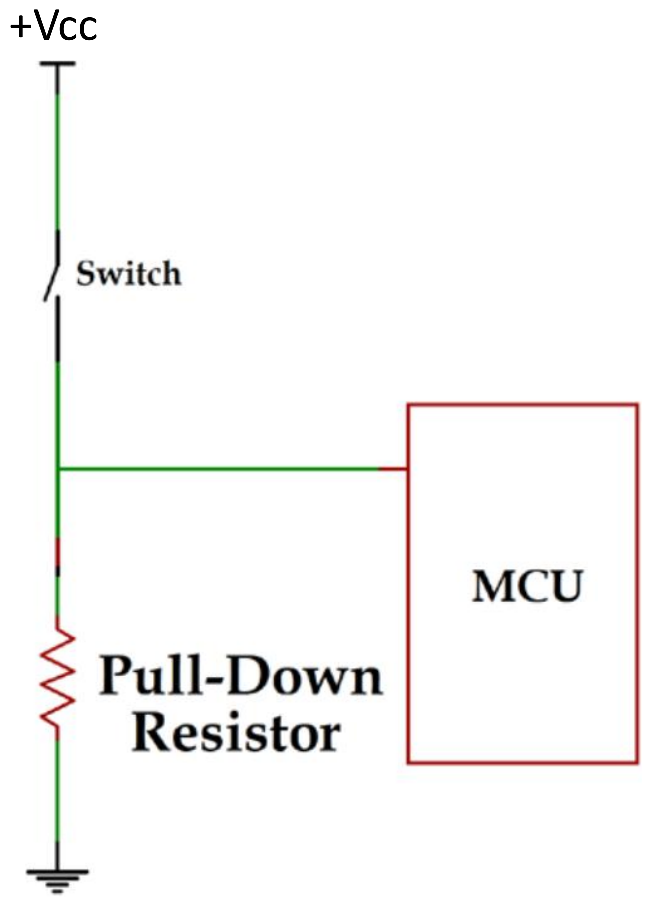
Asst. Prof. Dr. Seydi KAÇMAZ

Pull Up and Pull down Resistors

A Pull-up resistor is used to make the default state of the digital pin as **High**(Logic 1).

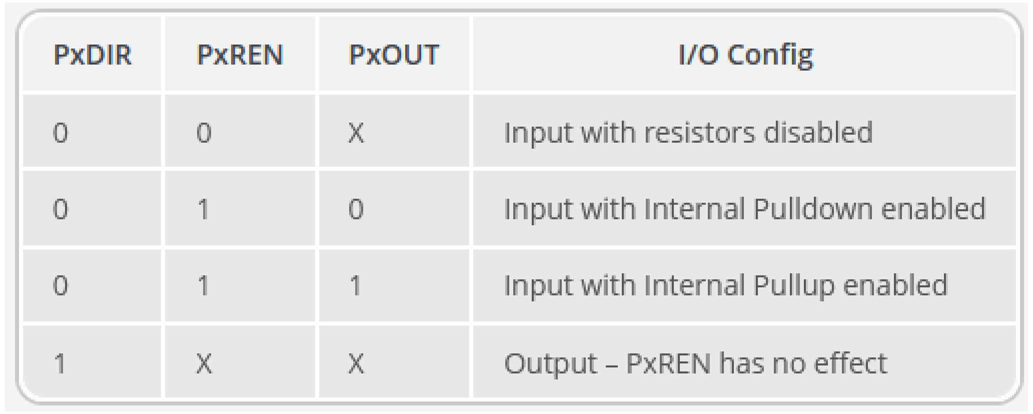
A Pull-down resistor is used to make the default state of the digital pin as **Low**(Logic 0).





Pull-Up and Pull-Down Resistors

- Pull- up and pull-down resistors are used only when the corresponding bits are **inputs.** (To determine the default state of the input pins.)
- !! Remember that MSP430 board has logic 1 on its inputs(P2.1 and P1.1) as default. Now it is possible to change the default values.
- **PxREN** register is employed to activate/deactivate the pull-up and pull-down resistors.
- To use them correctly, the table given below must be employed.



PxDIR	PxREN	PxOUT	I/O Config
0	0	X	Input with resistors disabled
0	1	0	Input with Internal Pulldown enabled
0	1	1	Input with Internal Pullup enabled
1	X	X	Output – PxREN has no effect

Table. The way of using pull-up and pull-down resistors

Pull-Up and Pull-Down Resistors

Ex. Write code block that makes the pins P1.3<—>P1.0 with **pull-down** enabled.

```
P1DIR=0xF0; //P1DIR=11110000, desired pins are inputs (P1.0…,P1.3)
P1REN=0x0F; //P1REN=00001111  Pull up or down is enabled
P1OUT=0;    //P1OUT=00000000  Pull-down is enabled
```

Ex. Do the same with **pull-up** enabled.

```
P1DIR=0xF0;// P1DIR=11110000, desired pins are inputs
P1REN=0x0F;// P1REN=00001111  Pull up or down is enabled
P1OUT=0x0F;// P1OUT=00001111  Pull-up is enabled
```

Pull-Up and Pull-Down Resistors

Ex. Write a C program by using C Language that turns the LED ON connected to P4.7 if P2.1 is pressed.

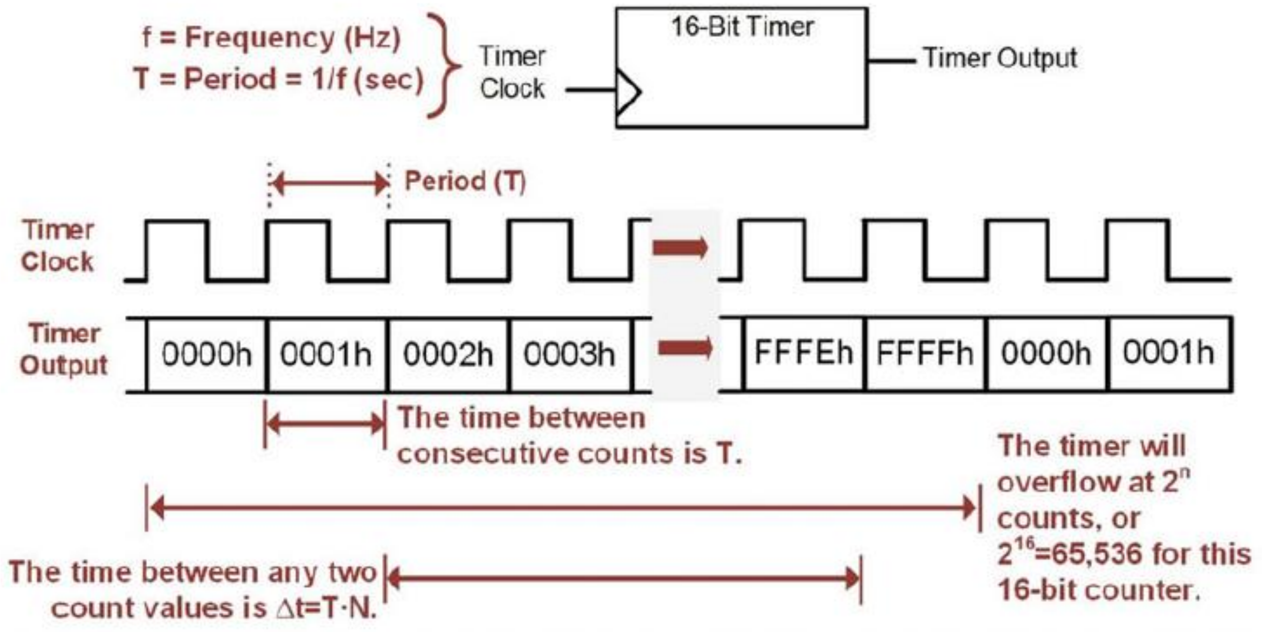
```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!
{
    if (BUTTON==0xFD) //Means "If button is pressed"
    {
        LED |= BIT7; //LED is ON
    }
   }
return 0;
}
```

Ex: Same example with different way…

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2DIR=0000000, P2 is input
P2REN=0xFD; //P2REN=11111101, all input pins are
pull-up/down enabled except P2.1
P2OUT=0; //Pull-down input pins except P2.1
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!
{
    if (BUTTON==0x00) //Means "If button is pressed"
    {
        LED |= BIT7; //LED is ON
    }
   }
return 0;
}
```

Timers

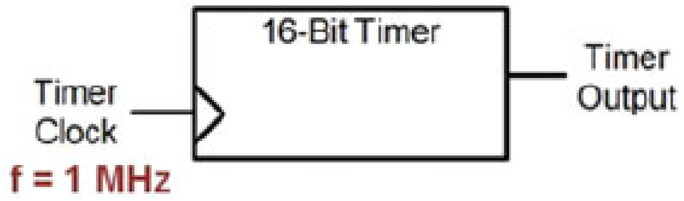
- A **timer** is a binary counter that is clocked from a free-running clock with a known frequency. Since the binary counter will increment on the triggering edge of the clock and the clock frequency is known, then the time between count values is deterministic. The time elapsed can be found by simply multiplying the period of the clock (T = 1/f) by the number of counts that have occurred (N).

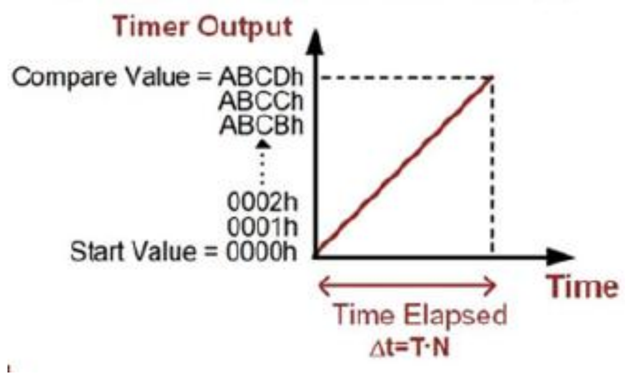


f = Frequency (Hz)
T = Period = 1/f (sec)
Timer Clock
16-Bit Timer
Timer Output
Period (T)
Timer Clock
Timer Output
0000h
0001h
0002h
0003h
FFFEh
FFFFh
0000h
0001h
The time between consecutive counts is T.
The timer will overflow at 2^n counts, or 2^{16}=65,536 for this 16-bit counter.
The time between any two count values is $\Delta t = T \cdot N$.

Timers

Ex. Calculate how much time (Δt) elapses between when a 16-bit timer is cleared and when it reaches the value of ABCDH if clock frequency is 1MHz.





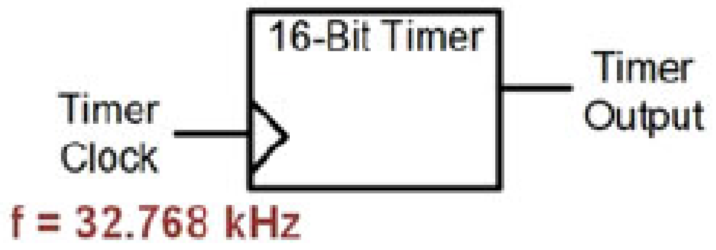
F=1/T→ T=1/f= 1/(1MHz)= 1us (time that elapses for each count)

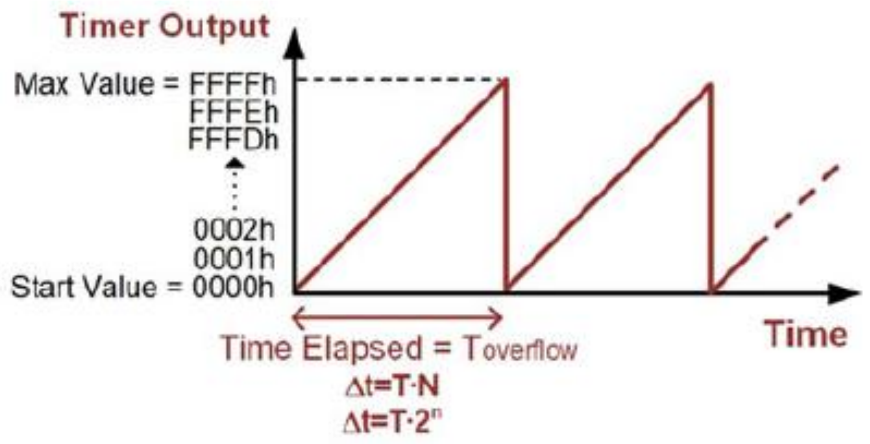
N= ABCDH= 43981 in decimal (total number of counts)

Δt= T·N= 1u·43981= 43, 981ms

Timers

Ex: Calculate the time **overflow** period of a 16-bit timer if the clock frequency is 32.768kHz.





F=1/T→ T=1/f= 1/(32.768kHz)= 30,518us (time that elapses for each count)

N= FFFFH= 65536 in decimal (total number of counts)

Δt= T·N= 30,518u · 65536 ≈ 2s

Timers

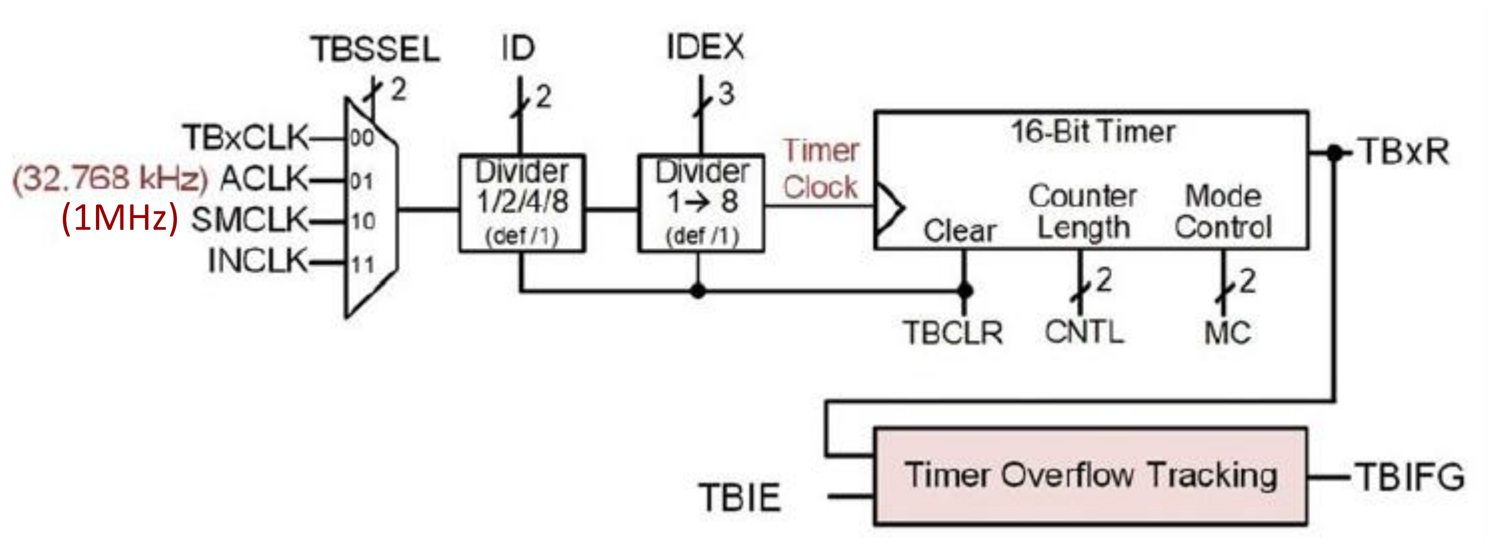
The full MSP430 architecture contains three distinct timer sub-systems: Timer_A, Timer_B, and the real-time clock counter (RTC). Within the Timer_A and Timer_B systems, there are multiple, independent binary counters that provide separate timing capability. Each timer can generate interrupts when its value either matches a value placed into a compare register, or when it overflows. The timers also have the ability to capture the current count value and store it into a register upon a triggering event. The **Capture and Compare Registers (CCRs)** are shared and referred to as capture/compare blocks in the MSP430 documentation.

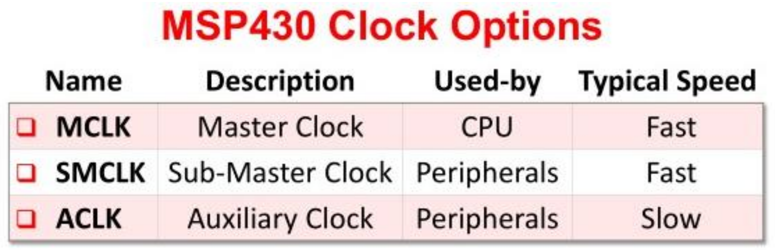
We will use ONLY Timer_B for the sake of simplicity

Timer Overflows

The MSP430F5529 Timer_B system provides one independent timer (TB0) with selectable clock inputs and the ability to divide down the clock to get slower counting frequencies. Timers TB0 has seven capture/compare (CC) registers associated with them.

Figure shows an overview of the Timer_B architecture implemented on the MSP430F5529.

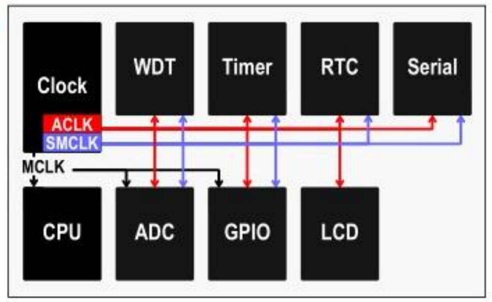



MSP430 Clock Options

	Name	Description	Used-by	Typical Speed
❑	MCLK	Master Clock	CPU	Fast
❑	SMCLK	Sub-Master Clock	Peripherals	Fast
❑	ACLK	Auxiliary Clock	Peripherals	Slow

Clocks – Fast or Slow

- All MSP430 devices provide at least 3 clocks
- Tune system peripherals by choice of clock:
  - Fast = Performance
  - Slow = Low-power
- Fast/slow clocks also provide wider timing



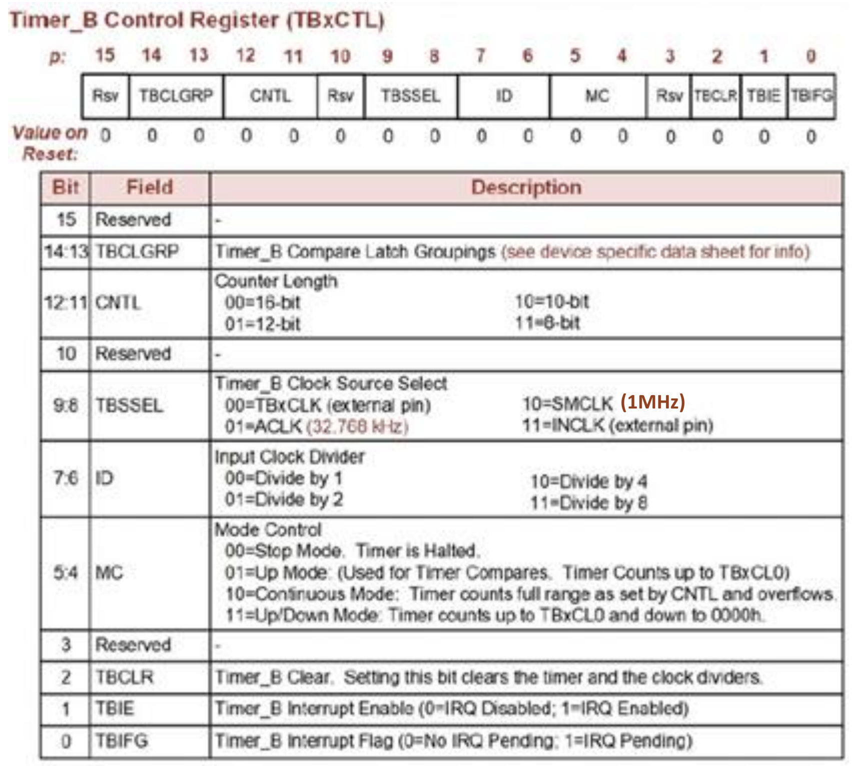
Timer Overflows

The first setting available for the timer clock is its source **(TBSSEL)**. The timer system clock can come from one of two external pins (TBxCLK or INCLK) or from one of two on-chip clock sources (ACLK or SMCLK). **On the MSP430F5529 LaunchPad, ACLK has a frequency of 32.768 kHz and SMCLK has a frequency of 1 MHz.** The timer system also allows the user to divide down the incoming clock source in order to achieve even slower counting frequencies. There are two clock dividers implemented in series in the Timer_B system. The first divider (**ID**) can divide the clock by 1, 2, 4, or 8. The second divider **(IDEX)** can divide the clock by 1, 2, 3, 4, 5, 6, 7 or 8. Since these two dividers are in series, there are 32 different divider settings that can be applied to the clock ranging from a minimum divider of 1 to a maximum divider of 64.

When the Timer_B is put into continuous counting mode, it will count up to its maximum value and then roll-over to 0. When it goes from its maximum value (i.e., FFFFh for 16-bit counting mode) to 0000h, a timer overflow is detected and can generate an interrupt. The local enable for this overflow interrupt is **TBIE**. This interrupt is maskable, so its global enable is **GIE**. When enabled, the interrupt will assert the timer overflow flag **TBIFG**.

Timer Overflows

All of the settings to control the Timer_B system(s) and use its timer overflow interrupts are held in two configuration registers, the Timer_B Control Register (TBxCTL) and the Timer_B Expansion Register 0 (TBxEX0). Figures 1 and 2 give the details of the TBxCTL and TBxEX0 registers respectively.



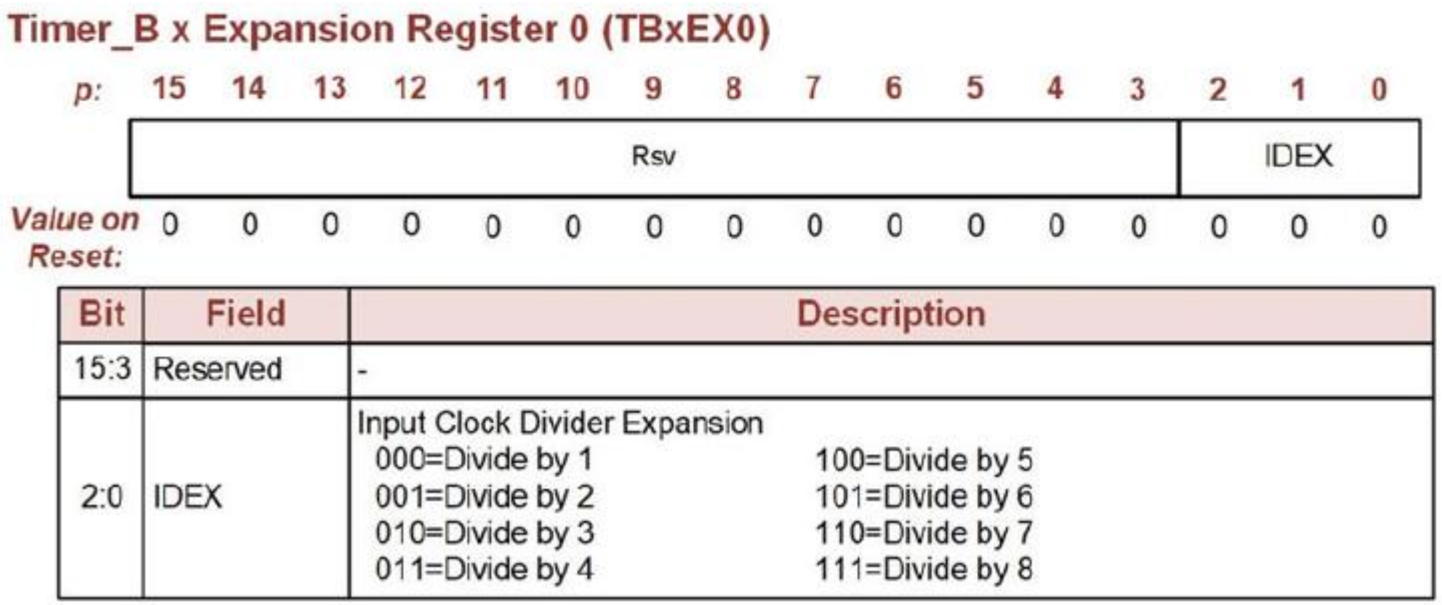
Timer_B Control Register (TBxCTL)

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 data sheet for info)
12:11	CNTL	Counter Length 00=16-bit 10=10-bit 01=12-bit 11=8-bit
10	Reserved	-
9:8	TBSSEL	Timer_B Clock Source Select 00=TBxCLK (external pin) 10=SMCLK **(1MHz)** 01=ACLK (32.768 kHz) 11=INCLK (external pin)
7:6	ID	Input Clock Divider 00=Divide by 1 10=Divide by 4 01=Divide by 2 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)

Fig.1 Timer_B control register (TBxCTL) details

x=0



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 1 100=Divide by 5 001=Divide by 2 101=Divide by 6 010=Divide by 3 110=Divide by 7 011=Divide by 4 111=Divide by 8

Fig.2. Timer_B expansion register 0 (TBxEX0) details

Timer Overflows

Let's now look at using a timer overflow to generate an event at a specific time interval. The recommended sequence of programming steps to configure the counter is as follows:

1. Write a 1 to the TBCLR bit (TBCLR =1) to clear TBxR.

2. Apply desired configurations to **TBxCTL** (Figure 1 in the previous page)

TBSSEL: Clock source selection (mostly ACLK or SMCLK)

ID: First frequency divider (1, 2, 4 or 8)

IDEX: Second frequency divider (1, 2, 3, 4, 5, 6, 7 or 8. Apart from others, It is in TBxEX0, check Figure 2)

CNTL: Counter length (8-bit, 10-bit, 12-bit, 16-bit)

MC: Mode Control (Counting style and direction, check Figure 1)

TBIE: Enabling interrupt

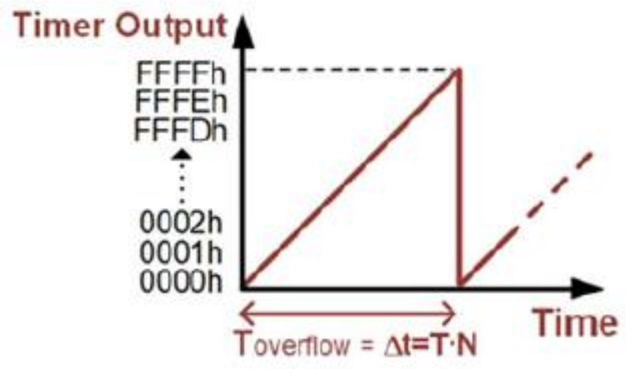
3. Clear interrupt flag **TBIFG**, for further use before and after

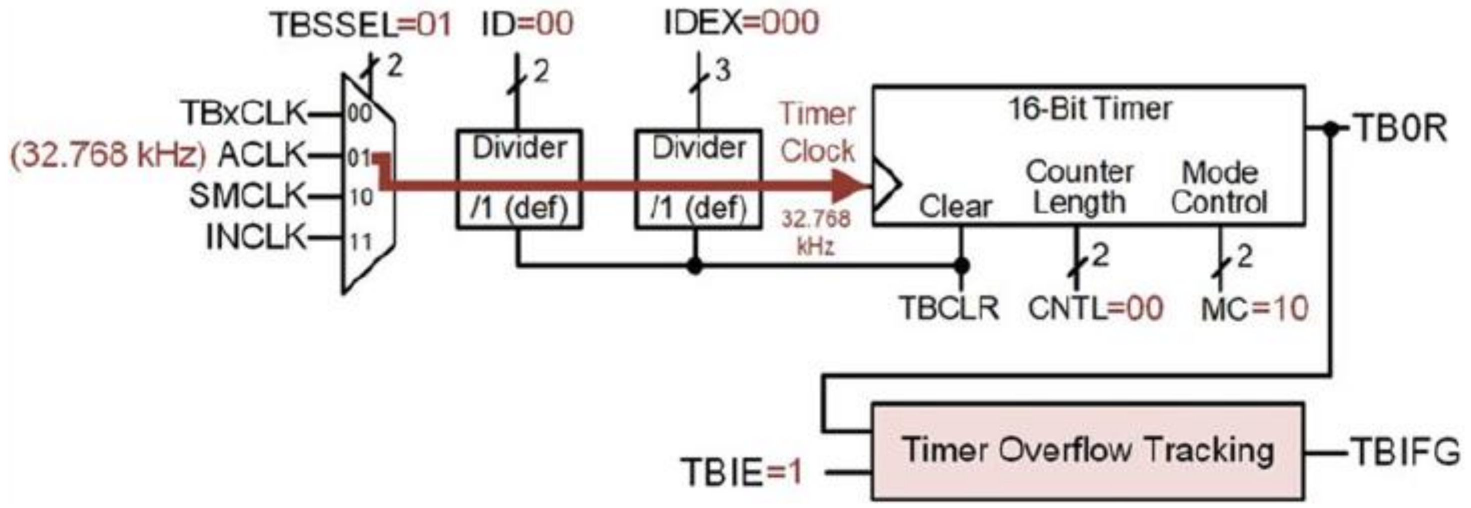
Timer Overflows

Ex. An example of using the TB0 timer to generate an interrupt every 2s. In this example, TB0 will use ACLK as its source and use the default settings of the two clock dividers (i.e., divide-by-1). The timer will run with a 16-bit length (default) and in continuous mode so that overflows happen indefinitely. When a timer overflow occurs, an interrupt will be triggered.

ACLK=32.768kHz and there is no division of the clock source

Toverflow= T·N= (1/f) ·2^n=(1/32.768) · 2^16= 2 seconds





From Figure 1 and 2

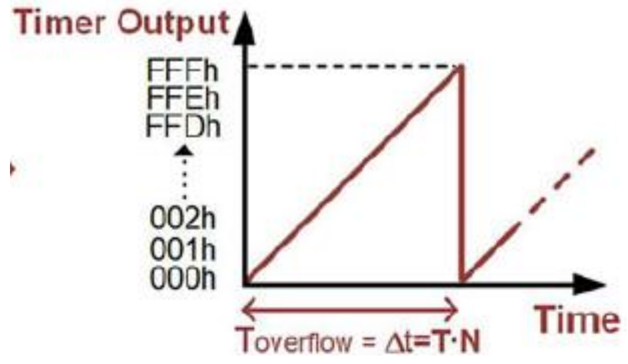
TBSSEL=01 (ACLK is active, 32.768kHz)
ID=0 (1st divider is 1)
IDEX=000 (2nd divider is 1)
CNTL=00 (16-bit counter)
MC=10 (Continuous mode, count up to end)
TBIE= 1 (Timer overflow interrupt is enabled, otherwise timer doesn't create interrupt)

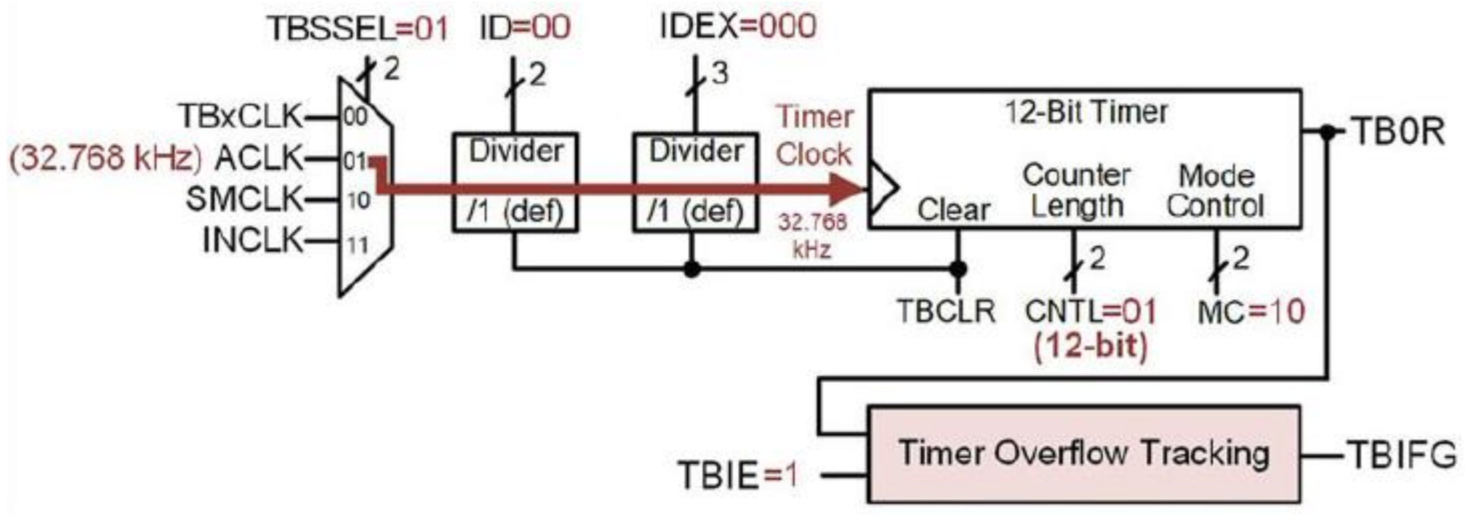
Timer Overflows

Ex. An example of using the TB0 timer to generate an interrupt every 125ms. In this example, TB0 will use ACLK as its source and use the default settings of the two clock dividers (i.e., divide-by-1). The timer will run with a 12-bit length and in continuous mode so that overflows happen indefinitely. When a timer overflow occurs, an interrupt will be triggered.

ACLK=32.768kHz and there is no division of the clock source

Toverflow= T·N= (1/f) ·2^n=(1/32.768) · 2^12= 125ms





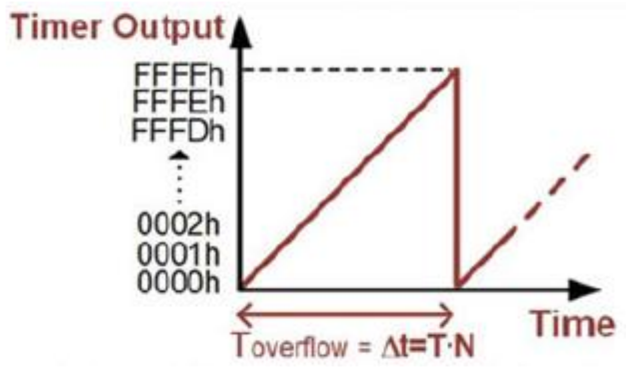
TBSSEL=01 (ACLK is active, 32.768kHz)
ID=00 (1st divider is 1)
IDEX=000 (2nd divider is 1)
CNTL=01 (12-bit counter)
MC=10 (Continuous mode, count up to end)
TBIE= 1 (Timer overflow interrupt is enabled, otherwise timer doesn't create interrupt)

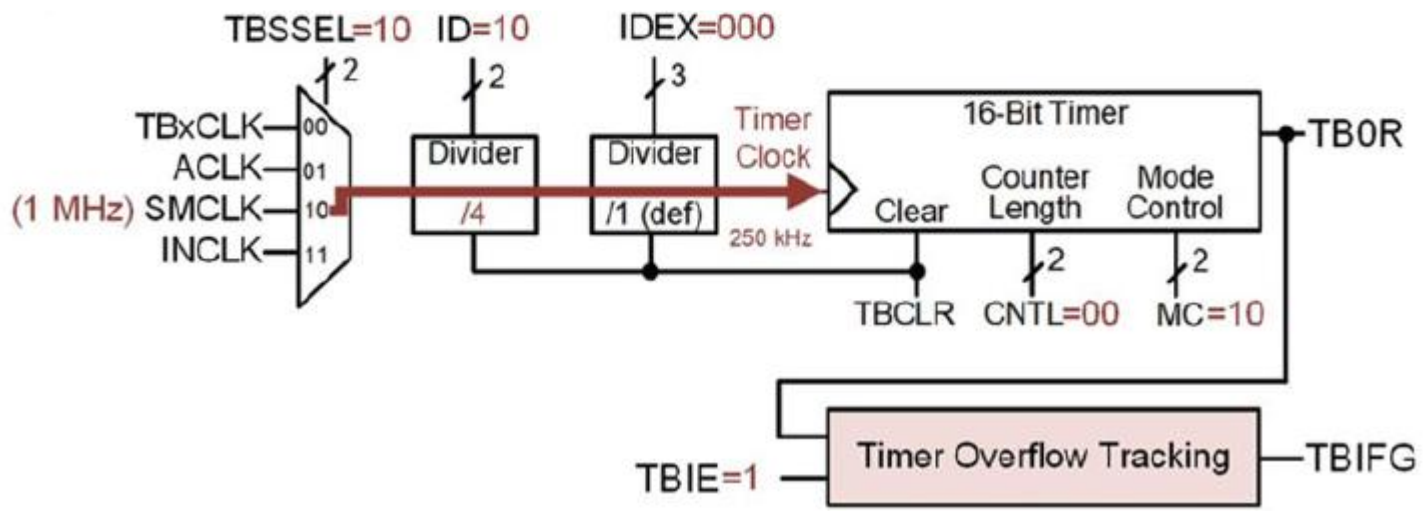
Timer Overflows

Ex. An example of using the TB0 timer to generate an interrupt every 262ms. In this example, TB0 will use **SMCLK** as its source and use the 1st divider by 4 in continuous mode so that overflows happen indefinitely. When a timer overflow occurs, an interrupt will be triggered. Assume SMCLK= 1MHz

SMCLK= 1MHz and there is a division of the clock source by 4

Toverflow= T·N= (1/**(f/4)**) ·2^n=(1/32.768) · 2^16= 262ms

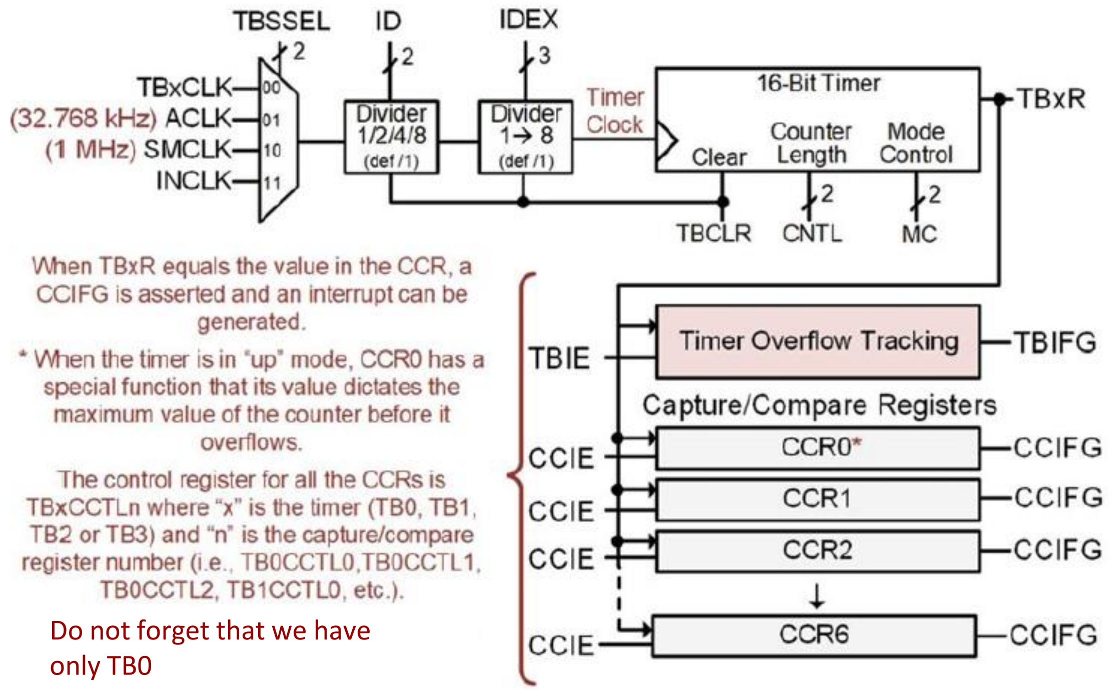




TBSSEL=10 (SMCLK is active, 1 MHz)
ID=10 (1st divider is 4)
IDEX=000 (2nd divider is 1)
CNTL=00 (16-bit counter)
MC=10 (Continuous mode, count up to end)
TBIE= 1 (Timer overflow interrupt is enabled, otherwise timer doesn't create interrupt)

Timer Compares

A timer compare will trigger an event when the main timer value equals a value stored in one of the MSP430's capture/compare registers (CCR). These registers are used for either the compare function or the capture function, which is why they are always referred to as CCRs and not simply compare registers. When the values match, the CCR will assert a flag (CCIFG = capture/compare flag) and can trigger an interrupt if enabled. Each CCR has its own enable (CCIE = capture/compare interrupt enable) and is maskable with the GIE bit.



When TBxR equals the value in the CCR, a CCIFG is asserted and an interrupt can be generated.

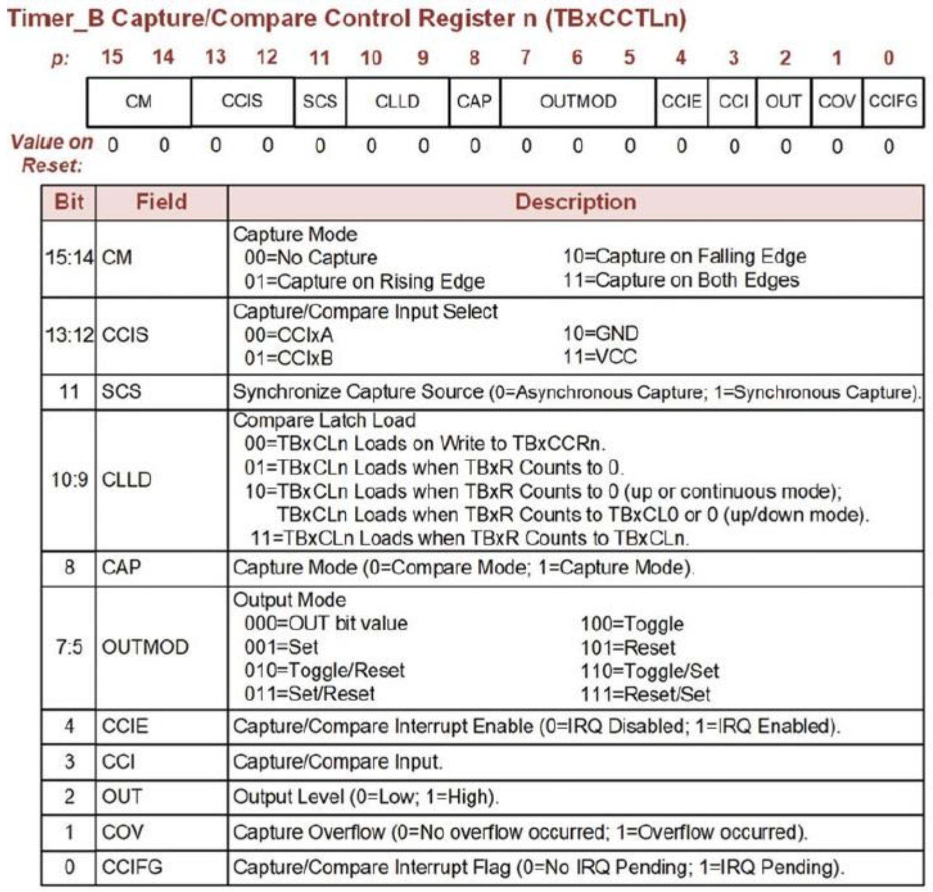
* When the timer is in "up" mode, CCR0 has a special function that its value dictates the maximum value of the counter before it overflows.

The control register for all the CCRs is TBxCCTLn where "x" is the timer (TB0, TB1, TB2 or TB3) and "n" is the capture/compare register number (i.e., TB0CCTL0,TB0CCTL1, TB0CCTL2, TB1CCTL0, etc.).

Do not forget that we have only TB0

Timer Compares

Each Timer_B CCR register is configured by its own Timer B Capture/Compare Control Register (TBxCCTLn). The notation for this register is that “x” stands for the timer (TB0, x=0 in our examples) and the “n” stands for the CCR number (TBxCCTL0, TBxCCTL1, TBxCCTL2, etc.). Figure shows the bit functionality of the TBxCCTLn registers.



Timer_B Capture/Compare Control Register n (TBxCCTLn)

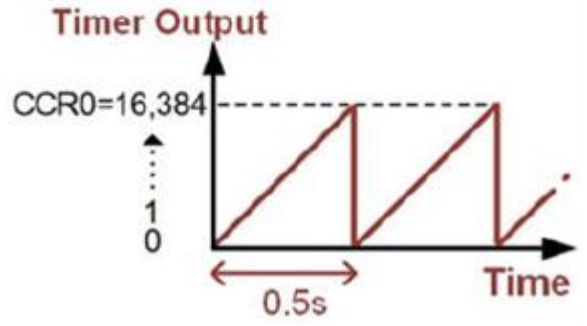
p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CM		CCIS		SCS	CLLD		CAP	OUTMOD			CCIE	CCI	OUT	COV	CCIFG
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:14	CM	Capture Mode 00=No Capture 10=Capture on Falling Edge 01=Capture on Rising Edge 11=Capture on Both Edges
13:12	CCIS	Capture/Compare Input Select 00=CCIxA 10=GND 01=CCIxB 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 100=Toggle 001=Set 101=Reset 010=Toggle/Reset 110=Toggle/Set 011=Set/Reset 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).

Fig. Timer_B capture/compare control register (TBxCCTLn) details

Timer Compares

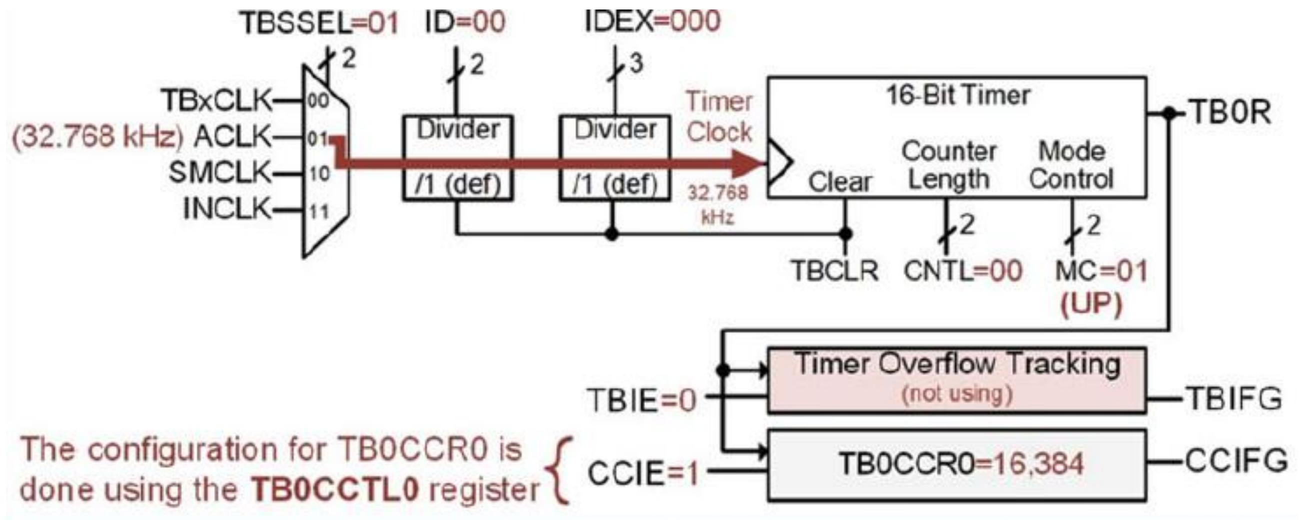
Ex. An example of using a timer compare to generate an event every 0.5s. We will use ACLK as the timer source without any division. We need to put the timer into “UP” mode to enable the compare functionality for CCR0. We then need to load CCR0 with the compare value that we want to use as the maximum value of the timer before it overflows and starts counting at 0.



ACLK= 32.768 kHz and there is no division

Δt= T·N= (1/**32.768)**)·N=(1/32.768)· N= 500ms →N=16.384

(n must be 16. Because 12-bit counter isn't enough)



Timer Overflow Examples

Ex. Write a C language program that toggles the LED on P4.7 in every 2s.

```
#include <msp430.h>
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
//-- Setting up timer
TB0CTL|=TBCLR;          //Clear timer and dividers
TB0CTL|=TBSSEL__ACLK;   //Source=ACLK
TB0CTL|=MC__CONTINUOUS; //Mode= Continuous
//-- Setting up timer Overflow IRQ (Interrupt Request)
TB0CTL|=TBIE;          //Enable TB0 Overflow IRQ
TB0CTL &=~TBIFG;       //Clear TB0 flag
__enable_interrupt();  //Enable maskable IRQs

//-- Main loop
while(1)  // Loop forever
  {}
return 0;
}
#pragma vector = TIMER0_B1_VECTOR // TIMER0_B1_VECTOR is the vector for TB0IFG
__interrupt void ISR_TB0_Overflow(void) // This function is called in every 2s
{
    LED^=BIT7;       // Toggle the LED on P4.7
    TB0CTL &=~TBIFG; //Clear TB0 flag, it is required. Otherwise, next interrupt call will not be realized, program stops after 1st run
}
```

!! If the other settings for timer are not done, their default values are used. These are:

ID=00 (1st divider is 1)
IDEX=000 (2nd divider is 1)
CNTL=00 (16-bit counter)
TBIFG flag must be cleared before enabling interrupt and at the end of the interrupt function!

Timer Overflow Examples

Ex. Write the same program that speeds up the toggling by 16 times, which means toggling in every 125ms.

```
#include <msp430.h>
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
//-- Setting up timer
TB0CTL|=TBCLR;          // Clear timer and dividers
TB0CTL|=TBSSEL__ACLK;   // Source=ACLK
TB0CTL|=MC__CONTINUOUS; // Mode= Continuous
TB0CTL|=CNTL_1;         // Length=12-bit
//-- Setting up timer Overflow IRQ (Interrupt Request)
TB0CTL|=TBIE;           //Enable TB0 Overflow IRQ
TB0CTL &=~TBIFG;        //Clear TB0 flag
__enable_interrupt(); //Enable maskable IRQs
//-- Main loop
while(1)  // Loop forever
  {}
return 0;
}
#pragma vector = TIMER0_B1_VECTOR // TIMER0_B1_VECTOR is the vector for TB0IFG
__interrupt void ISR_TB0_Overflow(void) // This function is called in every 125ms
{
    LED^=BIT7;        // Toggle the LED on P4.7
    TB0CTL &=~TBIFG; //Clear TB0 flag
}
```

!! If the other settings for timer are not done, their default values are used. These are

ID=00 (1st divider is 1)
IDEX=000 (2nd divider is 1)

Timer Overflow Examples

```
#include <msp430.h>
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
//-- Setting up timer
TB0CTL|=TBCLR; //Clear timer and dividers
TB0CTL|=TBSSEL_1; //Source=01=ACLK, From Figure 1
TB0CTL|=MC_2; //Mode=10=Continuous, From Figure 1
TB0CTL|=CNTL__12; //Length=12-bit
//-- Setting up timer Overflow IRQ (Interrupt Request)
TB0CTL|=TBIE;     //Enable TB0 Overflow IRQ
TB0CTL &=~TBIFG; //Clear TB0 flag
__enable_interrupt(); //Enable maskable IRQs

//-- Main loop
while(1)  //Loop forever
  {}
return 0;
}
#pragma vector = TIMER0_B1_VECTOR //TIMER0_B1_VECTOR is the vector for TB0IFG
__interrupt void ISR_TB0_Overflow(void)
{
    LED^=BIT7; //Toggle the LED on P4.7
    TB0CTL &=~TBIFG; //Clear TB0 flag
}
```

If it is sometimes hard to remember the codes, use the words assigned to them.
* If you want to use **words instead of codes**, use double underscore, which is “__”
* If you want to use **codes instead of words**, use single underscore, which is“_”
Choose the more catchy one for yourself

Timer Overflow Examples

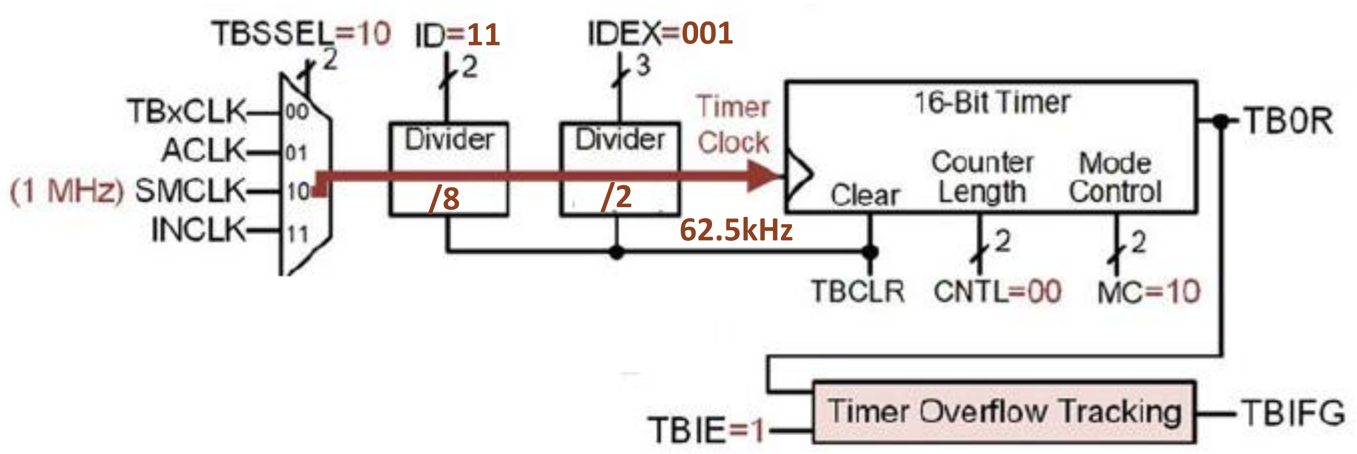
Ex. Design and write a C program that toggles the LED connected on P4.7 in every 1s. Use SMCLK (1MHz) as a clock source.

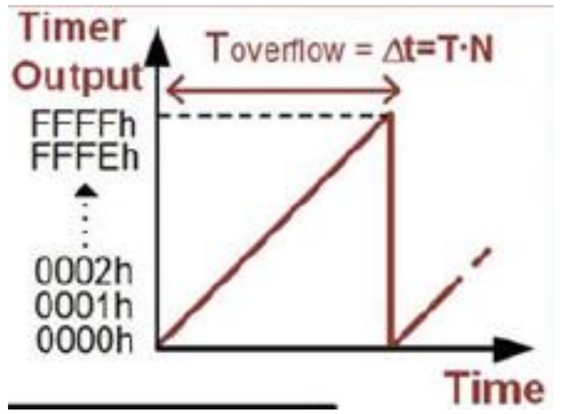
Maximum value of ID is 8 and if we use 8 for ID, let's what happens…

- Toverflow=(1/(f/8))·N = (1/(1MHz/8))·2^16 ≈ 524ms. (we can't obtain 1s delay with maximum ID)

Therefore we must employ the second divider IDEX and set it to be 2 as the divider value.

- Toverflow=(1/(f/(8*2)))·N = (1/(1MHz/16))·2^16 = 1048ms ≈ 1s.





Timer Overflow Examples

```
#include <msp430.h>
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD;// stop watchdog timer
P4DIR=0xFF;// P4 is output
P4OUT=0x00; // Clear P4
//-- Setting up timer
TB0CTL|=TBCLR; // Clear timer and dividers
TB0CTL|=TBSSEL__SMCLK; //Source=SMCLK
TB0CTL|=MC__CONTINUOUS; // Mode=Continuos
TB0CTL|=ID__8;  //Divide SMCLK by 8
TB0EX0|=TBIDEX__2; //Divide SMCLK by 2, total divider is 16
//-- Setting up timer Overflow IRQ (Interrupt Request)
TB0CTL|=TBIE;    //Enable TB0 Overflow IRQ
TB0CTL &=~TBIFG; //Clear TB0 flag
__enable_interrupt(); //Enable maskable IRQs
//-- Main loop
while(1)  // Loop forever
  {}
return 0;
}
#pragma vector = TIMER0_B1_VECTOR // TIMER0_B1_VECTOR is the vector for TB0IFG
__interrupt void ISR_TB0_Overflow(void)
{
    LED^=BIT7; // Toggle the LED on P4.7
    TB0CTL &=~TBIFG; //Clear TB0 flag
}
```

TBSSEL=10 (SMCLK is active, 1 MHz)
ID=10 (1st divider is 8)
IDEX=001 (2nd divider is 2)
CNTL=00 (16-bit counter)
MC=10 (Continuous mode, count up to end)
TBIE= 1 (Timer overflow interrupt is enabled, otherwise timer doesn't create output)

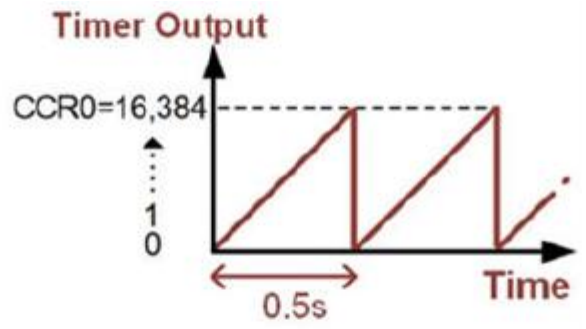
Timer Compare Examples

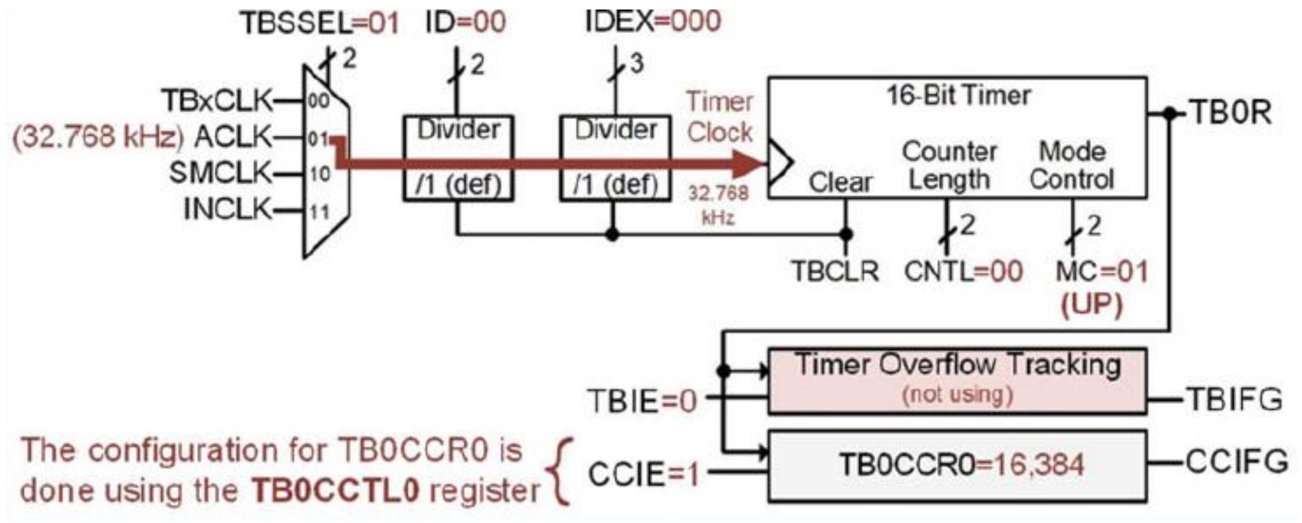
Ex. Design and write a C program that toggles the LED connected on P4.7 in every 500ms.

ACLK= 32,768 kHz and there is no division

Δt= T·N= (1/**32.768)**)·N=(1/32.768)· N= 500ms →N=16.384

(n must be 16. Because 12-bit counter isn't enough)





Timer Compare Examples

```
#include <msp430.h>
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD;// stop watchdog timer
P4DIR=0xFF;// P4 is output
P4OUT=0x00; // Clear P4
//-- Setting up timer
TB0CTL|=TBCLR;        //Clear timer and dividers
TB0CTL|=TBSSEL__ACLK; //Source=ACLK
TB0CTL|=MC__UP;       //Mode= Up, for compare, MC must be UP
TB0CCR0=16384; //Capture Compare Register is loaded with 16384 to create an interrupt
//-- Setting up timer Compare IRQ (Interrupt Request)
TB0CCTL0|=CCIE;    //Enable TB0 CCR0 Compare IRQ
TB0CCTL0 &=~CCIFG; //Clear CCR0 flag
__enable_interrupt(); //Enable maskable IRQs

//-- Main loop
while(1)  // Loop forever
  {}
return 0;
}
#pragma vector = TIMER0_B0_VECTOR // TIMER0_B0_VECTOR is the vector for CCIFG0
__interrupt void ISR_TB0_CCR0(void)
{
    LED^=BIT7; //Toggle the LED on P4.7
    TB0CCTL0 &=~CCIFG; //Clear CCR0 flag
}
```

TBSSEL=01 (ACLK is active, 32,768kHz)
MC=01 (Up mode, count up to end)
CCIE= 1 (Capture/compare interrupt is enabled, otherwise timer doesn’t create output)