# Introduction to Embedded System Design

## MSP430 Timer Module

Dhananjay V. Gadre
Associate Professor
ECE Division
Netaji Subhas University of Technology, New Delhi

Badri Subudhi
Assistant Professor
Electrical Engineering Department
Indian Institute of Technology, Jammu

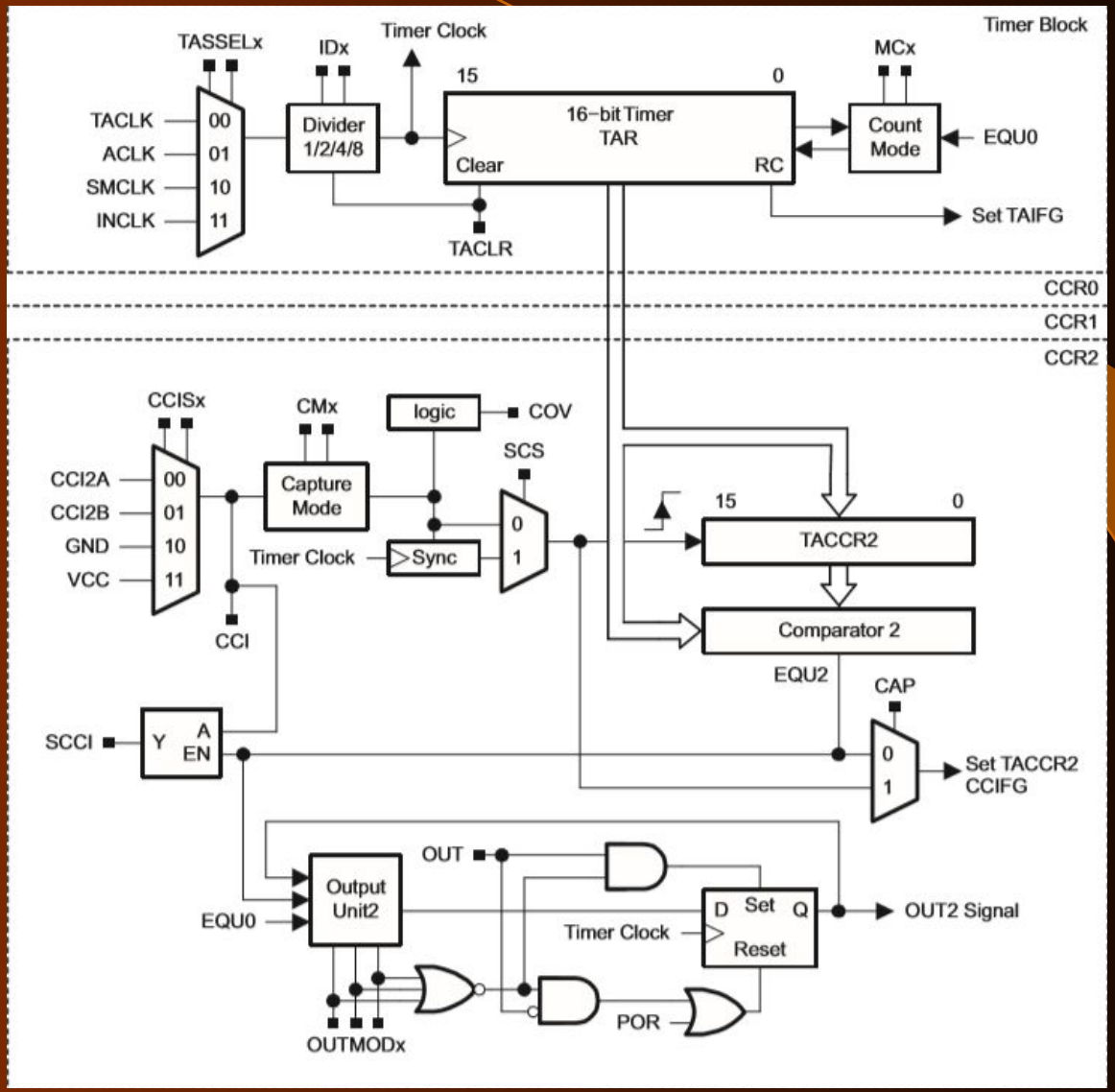
# Why Timer?

- Timers are used for time-specific tasks like generating delays, baud rates or for counting events in fixed intervals of time.
- A dedicated hardware timer frees up the processor so that it can perform more non-time critical process.
- Software delay loops are a waste of the processor's capabilities. Also, delays written in C are not precise. So, a hardware timer peripheral inside a microcontroller helps!
- MSP430G2553 has 2 16-Bit Timer_A: Timer0_A and Timer1_A.

# Features of TIMER_A In MSP430G2553

- Timer_A is a 16-bit counter with four operating modes. It can count till 65535 ($2^{16}$-1).
- Timer_A can be used for capture/compare and PWM signals.
- It can also be used to generate interrupts, which may be generated from the counter on overflow conditions or from any of the capture/compare registers.
- It can measure frequency or take time-stamp of the external inputs directly.
- Outputs can be driven at specified times precisely, either once or periodically.

# TIMER_A Block Diagram



# TIMER_A Major Components

- **Timer_A counter register - TAR**
  - This is a 16-bit register whose value increments or decrements with every rising edge of clock signal.
  - Behaviour (increment or decrement) of TAR register can be selected by configuring the mode of the timer.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TARx | | | | | | | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TARx | | | | | | | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |

| | | |
|---|---|---|
| **TARx** | Bits 15-0 | Timer_A register. The TAR register is the count of Timer_A. |

# TIMER_A Major Components

- **Timer_A capture/compare register (TACCRx)**
  - There are three capture/compare registers in each timer:
    - TA0CCR0, TA0CCR1, TA0CCR2 for Timer0_A
    - TA1CCR0, TA1CCR1, TA1CCR2 for Timer1_A.
  - All the three channels share the same TAR.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **TACCRx** | | | | | | | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TACCRx | | | | | | | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |

| | | |
|---|---|---|
| **TACCRx** | Bits 15-0 | Timer_A capture/compare register.<br>Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR.<br>Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed. |

# Capture vs Compare Mode

- In capture mode, the Timer value (TAR) is captured in this TACCRx register in an external or internal input event.
- In compare mode, Timer value (TAR) is compared with this TACCRx register and operations are performed based on the compare event.
- The compare mode is used to generate PWM output signals or interrupts at specific time intervals.

# Clock Source for Timer_A

- The timer clock can be sourced from ACLK, SMCLK, or externally via TACLK or INCLK (For Timer1_A in case of MSP430G2553, TACLK and INCLK is not present).
- The clock source is selected with the TASSELx bits.
- The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits.
- The timer clock divider is reset when TACLR is set.

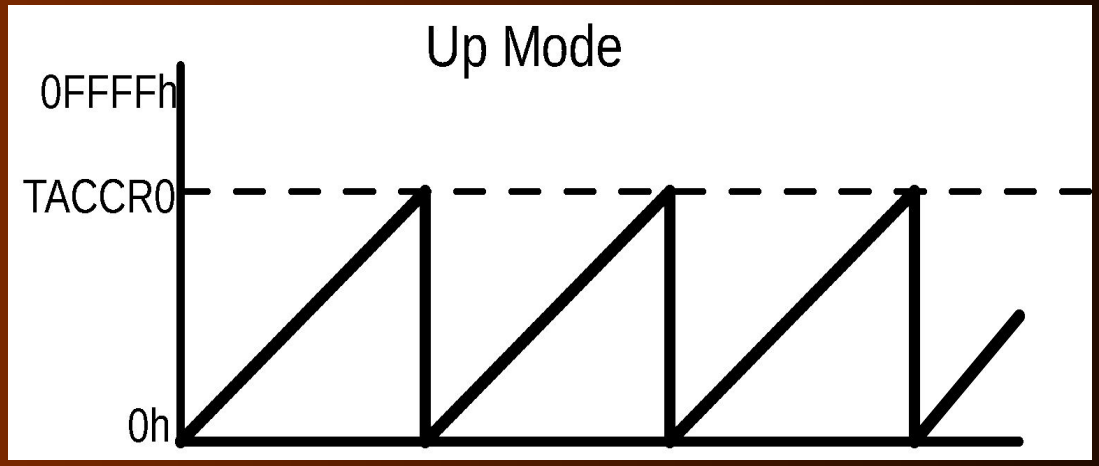
# Timer Modes

Timer's operation mode can be selected by configuring 'MCx' bits in the Timer Control TACTL register

| MC | Mode | Description |
|---|---|---|
| 00 | Stop | Timer is stopped. |
| 01 | Up | Timer repeatedly counts from zero to value stored in TACCR0 Register |
| 10 | Continuous | Timer repeatedly counts from zero to 0xFFFF hex. |
| 11 | Up/Down | Timer repeatedly counts from zero to value stored in TACCR0 and then back to zero. |

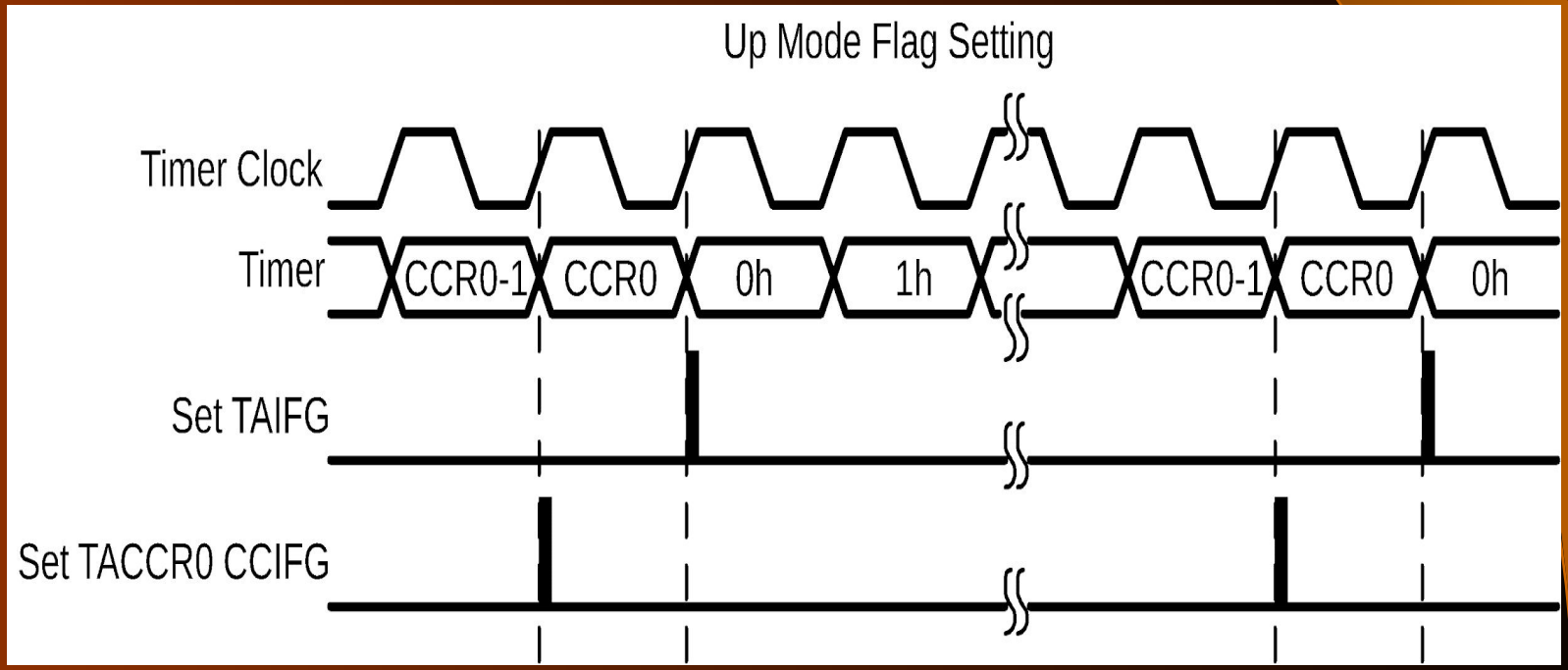
# Up Mode

- The timer repeatedly counts up to the value of compare register TACCR0 starting from 0000h.
- The number of timer counts in the period is TACCR0+1. When the timer value equals TACCR0 the timer restarts counting from zero.
- The up mode is used if the timer period is required to be different from 0FFFFh counts..



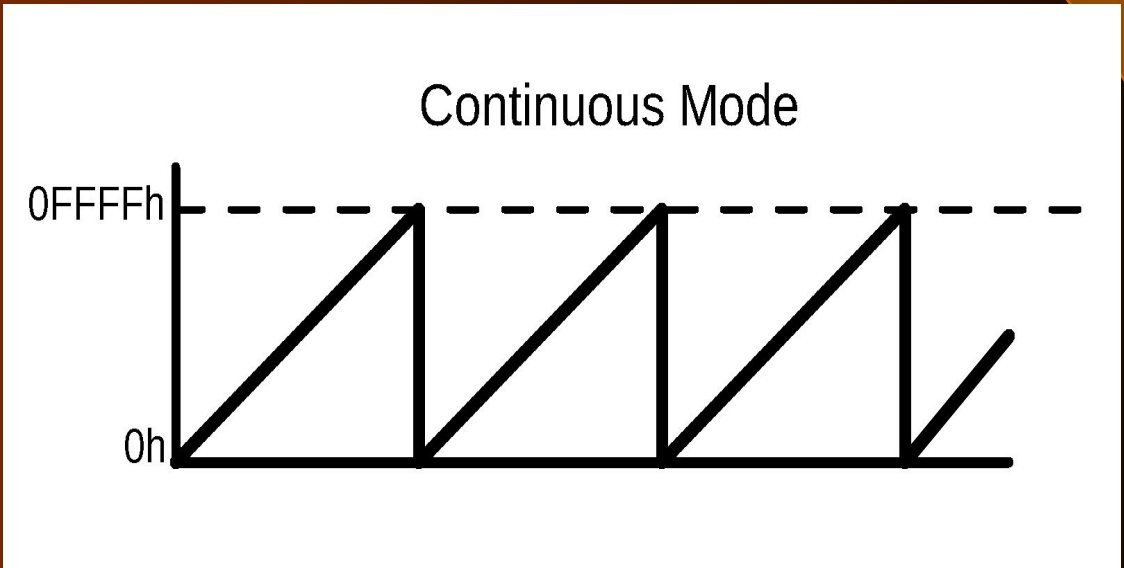
# Timer Interrupts in Up Mode

- The TACCR0 CCIFG interrupt flag is set when the timer *counts* to the TACCR0 value.
- The TAIFG interrupt flag is set when the timer *counts* from TACCR0 to zero.



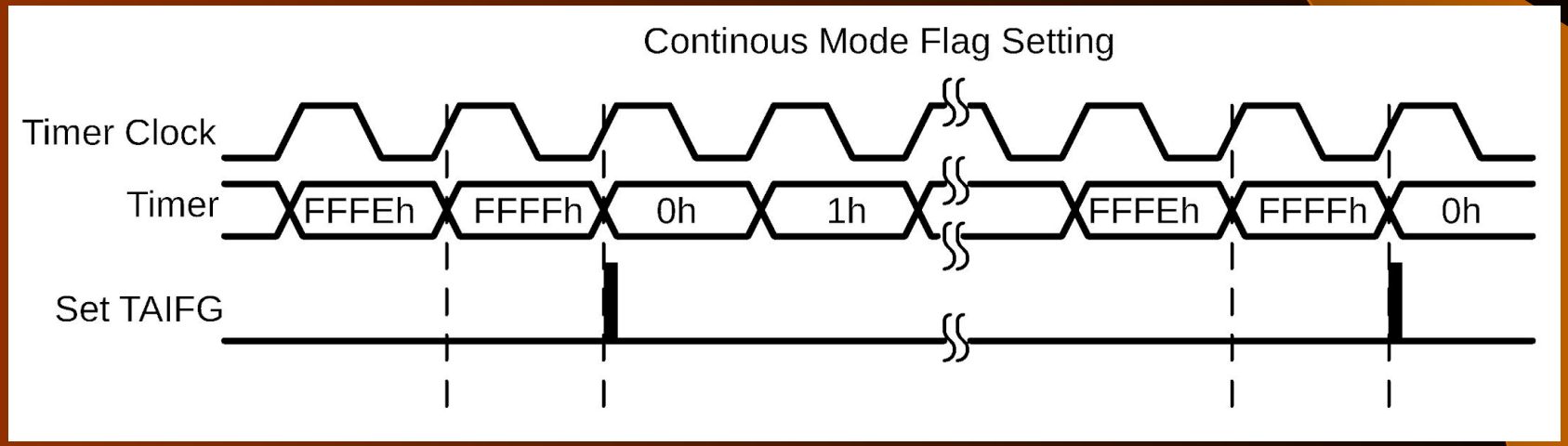
# Continuous Mode

- In the continuous mode, the timer repeatedly counts up to FFFFh and restarts from zero.



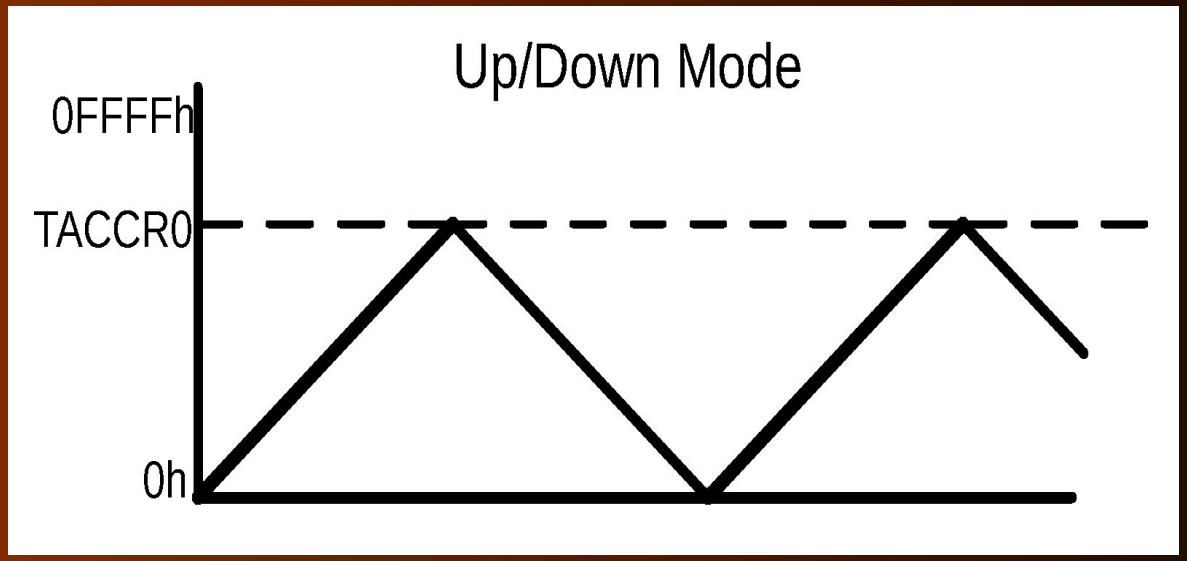
# Timer Interrupts in Continuous Mode

- The TAIFG interrupt flag is set when the timer *counts* from FFFFh to zero.



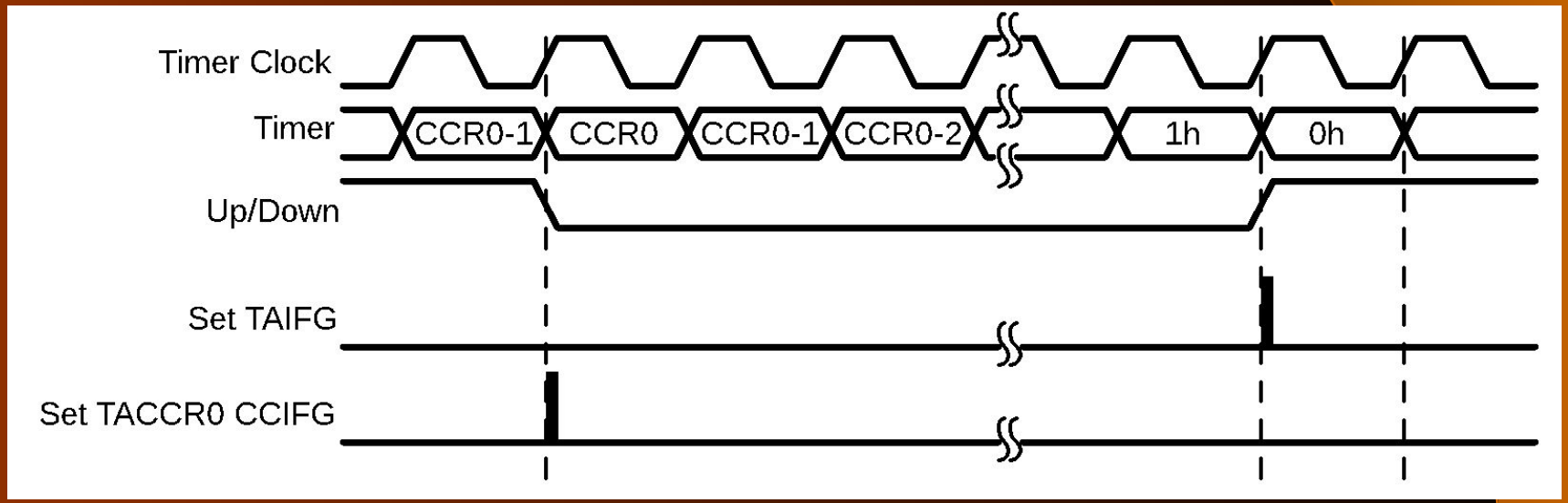
# Up/Down Mode

- The timer repeatedly counts up to the value of compare register TACCR0 and back down to zero.
- The up/down mode is used if the timer period must be different from 0FFFFh counts, and if a symmetrical pulse generation is needed.



# Timer Interrupts in Up/Down Mode

- The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0 – 1 to TACCR0
- TAIFG is set when the timer completes counting down from 0001h to 0000h.



# TIMER_A Registers

1. Timer_A control - TACTL
2. Timer_A counter - TAR
3. Timer_A capture/compare control 0 - TACCTL0
4. Timer_A capture/compare 0 - TACCR0
5. Timer_A capture/compare control 1 - TACCTL1
6. Timer_A capture/compare 1 - TACCR1
7. Timer_A capture/compare control 2 - TACCTL2
8. Timer_A capture/compare 2 - TACCR2
9. Timer_A Interrupt vector - TAIV

# TACTL, Timer_A Control Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Unused | | | | | | TASSELx | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IDx | | MCx | | Unused | TACLR | TAIE | TAIFG |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |

# TACTL Continued

| | | |
|---|---|---|
| **TASSELx** | Bits 9-8 | Timer_A clock source select |
| | | 00 TACLK |
| | | 01 ACLK |
| | | 10 SMCLK |
| | | 11 INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet) |
| **IDx** | Bits 7-6 | Input divider. These bits select the divider for the input clock. |
| | | 00 /1 |
| | | 01 /2 |
| | | 10 /4 |
| | | 11 /8 |
| **MCx** | Bits 5-4 | Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. |
| | | 00 Stop mode: the timer is halted. |
| | | 01 Up mode: the timer counts up to TACCR0. |
| | | 10 Continuous mode: the timer counts up to 0FFFFh. |
| | | 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h. |
| **Unused** | Bit 3 | Unused |
| **TACLR** | Bit 2 | Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero. |
| **TAIE** | Bit 1 | Timer_A interrupt enable. This bit enables the TAIFG interrupt request. |
| | | 0 Interrupt disabled |
| | | 1 Interrupt enabled |
| **TAIFG** | Bit 0 | Timer_A interrupt flag |
| | | 0 No interrupt pending |
| | | 1 Interrupt pending |

# TACCTLx, Capture/Compare Control Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CMx | | CCISx | | SCS | SCCI | Unused | CAP |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | r | r0 | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OUTMODx | | | CCIE | CCI | OUT | COV | CCIFG |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | r | rw-(0) | rw-(0) | rw-(0) |

Timer_A has 3 channels each being controlled by TACCTLx. Each channel has its TACCRx register.

# TACCTLx Continued

| | | |
|---|---|---|
| **CMx** | Bit 15-14 | Capture mode |
| | | 00 No capture |
| | | 01 Capture on rising edge |
| | | 10 Capture on falling edge |
| | | 11 Capture on both rising and falling edges |
| **CCISx** | Bit 13-12 | Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections. |
| | | 00 CCIxA |
| | | 01 CCIxB |
| | | 10 GND |
| | | 11 $V_{CC}$ |
| **SCS** | Bit 11 | Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. |
| | | 0 Asynchronous capture |
| | | 1 Synchronous capture |
| **SCCI** | Bit 10 | Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit |
| **Unused** | Bit 9 | Unused. Read only. Always read as 0. |
| **CAP** | Bit 8 | Capture mode |
| | | 0 Compare mode |
| | | 1 Capture mode |

# TACCTLx Continued

| | | |
|---|---|---|
| **OUTMODx** | Bits 7-5 | Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0. |
| | | 000 OUT bit value |
| | | 001 Set |
| | | 010 Toggle/reset |
| | | 011 Set/reset |
| | | 100 Toggle |
| | | 101 Reset |
| | | 110 Toggle/set |
| | | 111 Reset/set |
| **CCIE** | Bit 4 | Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. |
| | | 0 Interrupt disabled |
| | | 1 Interrupt enabled |
| **CCI** | Bit 3 | Capture/compare input. The selected input signal can be read by this bit. |
| **OUT** | Bit 2 | Output. For output mode 0, this bit directly controls the state of the output. |
| | | 0 Output low |
| | | 1 Output high |
| **COV** | Bit 1 | Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. |
| | | 0 No capture overflow occurred |
| | | 1 Capture overflow occurred |
| **CCIFG** | Bit 0 | Capture/compare interrupt flag |
| | | 0 No interrupt pending |
| | | 1 Interrupt pending |

# TAIV, Timer_A Interrupt Vector Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **TAIVx** | | | **0** |
| r0 | r0 | r0 | r0 | r-(0) | r-(0) | r-(0) | r0 |

**TAIVx** Bits 15-0 Timer_A interrupt vector value

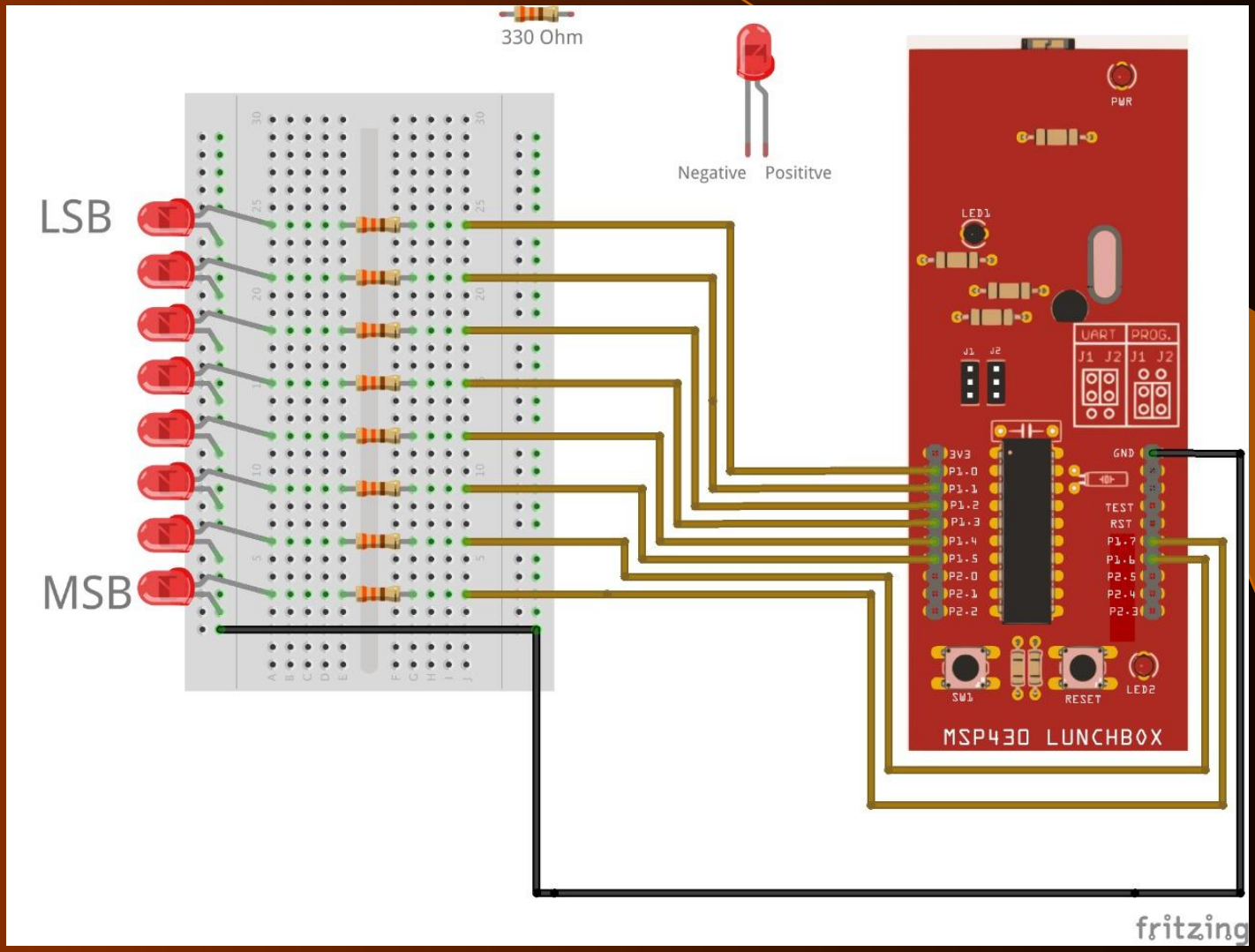
| TAIV Contents | Interrupt Source | Interrupt Flag | Interrupt Priority |
|---|---|---|---|
| 00h | No interrupt pending | - | |
| 02h | Capture/compare 1 | TACCR1 CCIFG | Highest |
| 04h | Capture/compare 2 (1) | TACCR2 CCIFG | |
| 06h | Reserved | - | |
| 08h | Reserved | - | |
| 0Ah | Timer overflow | TAIFG | |
| 0Ch | Reserved | - | |
| 0Eh | Reserved | - | Lowest |

(1) Not implemented in MSP430x20xx devices

# TIMER_A Interrupts

- Timer Overflow sets the flag TAIFG in TACTL.
- Interrupt vectors associated with the 16-bit Timer_A module:
  - TACCR0 interrupt vector for TACCR0 CCIFG.
  - TAIV interrupt vector for all other CCIFG flags and TAIFG.
- In capture mode any CCIFG flag is set when a timer value is captured in the associated TACCRx register.
- In compare mode, any CCIFG flag is set if TAR counts to the associated TACCRx value.
- Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

# Circuit Diagram



# Example Code : Hello_Timer

```
1 #include <msp430.h>
2 #include <inttypes.h>
3
4 volatile char count = 0;
5 volatile unsigned int i;
6
7 /**
8  * @brief
9  * These settings are wrt enabling GPIO on Lunchbox
10 **/
11 void register_settings_for_GPIO()
12 {
13     P1DIR |= 0xFF;                          //P1.0 to P1.7 are set as Output
14     P1OUT |= 0x00;                          //Initially they are set to logic zero
15 }
16
17 /**
18  * @brief
19  * These settings are w.r.t enabling TIMER0 on Lunch Box
20 **/
21 void register_settings_for_TIMER0()
22 {
23     CCTL0 = CCIE;                          // CCR0 interrupt enabled
24     TACTL = TASSEL_1 + MC_1;               // ACLK = 32768 Hz, upmode
25     CCR0 =  32768;                         // 1 Hz
26 }
```

# Example Code : Hello_Timer

```
27
28 /*@brief entry point for the code*/
29 void main(void)
30 {
31   WDTCTL = WDTPW + WDTHOLD;            //! Stop Watch dog (Not recommended for code in production and devices working in field)
32
33   do{
34         IFG1 &= ~OFIFG;                    // Clear oscillator fault flag
35         for (i = 50000; i; i--);           // Delay
36     } while (IFG1 & OFIFG);               // Test osc fault flag
37
38   register_settings_for_TIMER0();
39   register_settings_for_GPIO();
40   _BIS_SR(LPM3_bits + GIE);               // Enter LPM3 w/ interrupt
41 }
42
43 /*@brief entry point for TIMER0 interrupt vector*/
44 #pragma vector= TIMER0_A0_VECTOR
45 __interrupt void Timer_A (void)
46 {
47     count++;
48     P1OUT = count;                       //Assign value of Count to PORT 1 to represent it as binary number
49 }
```

Thank you!