# Lab 7
# More on Timers Capture Mode

**1. Watchdog Timers (WDT)**

•A **Watchdog Timer (WDT)** is a safety feature used to reset a system if it becomes unresponsive.

•It operates by counting down from a preset value. If the system does not reset the timer before it expires, the WDT triggers a **system reset**.

•**Use Case:** Prevents system hang-ups in critical applications like automotive and industrial control.

**2. Real-time Clocks (RTC)**

•**RTC** is used for accurate timekeeping in **seconds, minutes, hours, and even days**.

•It runs independently using a **low-power 32.768 kHz crystal oscillator**.

•**Use Case:** Time-stamping, logging, and scheduling tasks (e.g., alarms in embedded systems).

**3. Baud Rate Generators (BRG)**

•A **Baud Rate Generator (BRG)** generates precise timing signals for **serial communication protocols** like UART, SPI, and I2C.

•It ensures that **data is transmitted and received at a correct rate** by synchronizing the transmitter and receiver clocks.

•**Use Case:** UART communication in microcontrollers.

**4. Pulse-width Modulation (PWM)**

•**PWM** is used to generate **variable duty cycle waveforms** for controlling power, speed, or brightness.

•The duty cycle determines the **ON vs OFF time ratio**, useful for motor speed control, LED dimming, and signal generation.

•**Use Case:** Motor drivers, LED dimming, and audio signal generation.

# Extending the Timer Count

- **Cascading Multiple Timers**

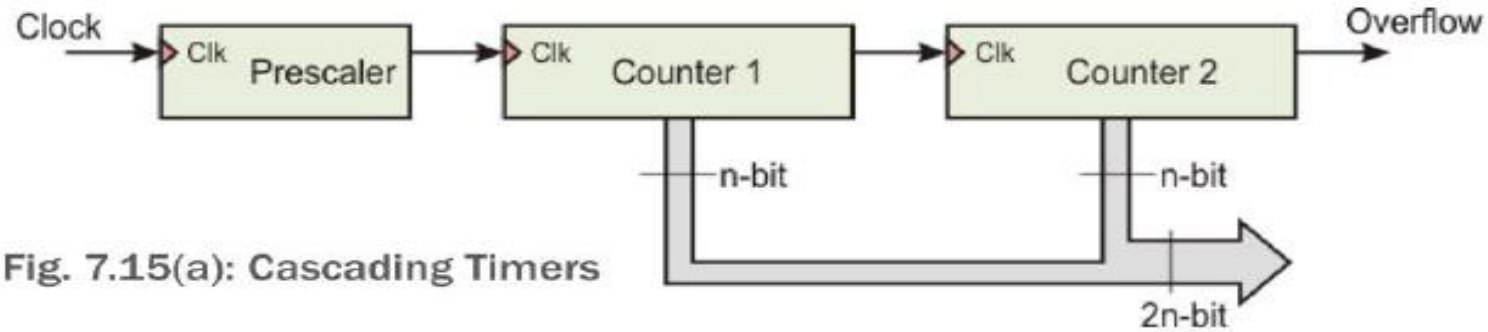


Fig. 7.15(a): Cascading Timers

- **Using Software Variable**

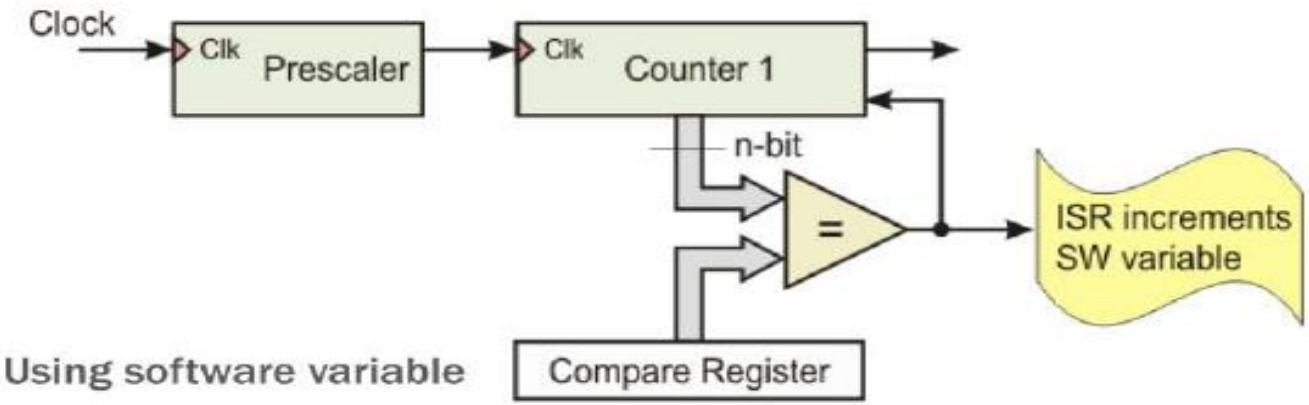


Fig. 7.15(b): Using software variable

# Timer_Ax Control Register

### 13.3.1 TAxCTL Register

Timer_Ax Control Register

**Figure 13-16. TAxCTL Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | TASSEL | |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ID | | MC | | Reserved | TACLR | TAIE | TAIFG |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | w-(0) | rw-(0) | rw-(0) |

## Table 13-4. TAxCTL Register Description

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-10 | Reserved | RW | 0h | Reserved |
| 9-8 | TASSEL | RW | 0h | Timer_A clock source select<br>00b = TAxCLK<br>01b = ACLK<br>10b = SMCLK<br>11b = INCLK |
| 7-6 | ID | RW | 0h | Input divider. These bits along with the TAIDEX bits select the divider for the input clock.<br>00b = /1<br>01b = /2<br>10b = /4<br>11b = /8 |
| 5-4 | MC | RW | 0h | Mode control. Setting MC = 0 when Timer_A is not in use conserves power.<br>00b = Stop mode: Timer is halted<br>01b = Up mode: Timer counts up to TAxCCR0<br>10b = Continuous mode: Timer counts up to 0FFFFh<br>11b = Up/Down mode: Timer counts up to TAxCCR0 then down to 0000h |
| 3 | Reserved | RW | 0h | Reserved |
| 2 | TACLR | RW | 0h | Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and always reads as zero. |
| 1 | TAIE | RW | 0h | Timer_A interrupt enable. This bit enables the TAIFG interrupt request.<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 0 | TAIFG | RW | 0h | Timer_A interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |

# Timer_Ax Capture/Compare Control n Register

### *13.3.3 TAxCCTLn Register*

Timer_Ax Capture/Compare Control n Register

**Figure 13-18. TAxCCTLn Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CM | | CCIS | | SCS | SCCI | Reserved | CAP |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | r-(0) | r-(0) | rw-(0) |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OUTMOD | | | CCIE | CCI | OUT | COV | CCIFG |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | r | rw-(0) | rw-(0) | rw-(0) |

## Table 13-6. TAxCCTLn Register Description

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | CM | RW | 0h | Capture mode<br>00b = No capture<br>01b = Capture on rising edge<br>10b = Capture on falling edge<br>11b = Capture on both rising and falling edges |
| 13-12 | CCIS | RW | 0h | Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections.<br>00b = CCIxA<br>01b = CCIxB<br>10b = GND<br>11b = VCC |
| 11 | SCS | RW | 0h | Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.<br>0b = Asynchronous capture<br>1b = Synchronous capture |
| 10 | SCCI | RW | 0h | Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read from this bit. |
| 9 | Reserved | R | 0h | Reserved. Reads as 0. |
| 8 | CAP | RW | 0h | Capture mode<br>0b = Compare mode<br>1b = Capture mode |
| 7-5 | OUTMOD | RW | 0h | Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0.<br>000b = OUT bit value<br>001b = Set<br>010b = Toggle/reset<br>011b = Set/reset<br>100b = Toggle<br>101b = Reset<br>110b = Toggle/set<br>111b = Reset/set |

| | | | | |
|---|---|---|---|---|
| 4 | CCIE | RW | 0h | Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 3 | CCI | R | 0h | Capture/compare input. The selected input signal can be read by this bit. |
| 2 | OUT | RW | 0h | Output. For OUTMOD = 0, this bit directly controls the state of the output.<br>0b = Output low<br>1b = Output high |

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 1 | COV | RW | 0h | Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.<br>0b = No capture overflow occurred<br>1b = Capture overflow occurred |
| 0 | CCIFG | RW | 0h | Capture/compare interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 13.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

**Figure 13-19. TAxCCRn Register**

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

**Table 13-7. TAxCCRn Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | TAxCCRn | RW | 0h | Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR.<br>Capture mode: The Timer_A register, TAxR, is copied into the TAxCCRn register when a capture is performed. |

### 13.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

**Figure 13-20. TAxIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TAIV | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TAIV | | | | | | | |
| r0 | r0 | r0 | r0 | r-(0) | r-(0) | r-(0) | r0 |

**Table 13-8. TAxIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | TAIV | R | 0h | Timer_A interrupt vector value<br>00h = No interrupt pending<br>02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest<br>04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG<br>06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG<br>08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG<br>0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG<br>0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG<br>0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest |

```c
#include <msp430fr4133.h>
#include <stdint.h>

uint16_t last_time = 0;  // Last captured time
uint16_t cap_diff, new_time = 0;

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;  // Stop watchdog timer

    // Configure LED on P1.0
    P1DIR |= BIT0;   // Set P1.0 as output (LED)
    P1OUT &= ~BIT0;  // Ensure LED starts OFF

    // Configure P1.6 for TA0.2 Capture Mode
    P1DIR &= ~BIT6;  // Set P1.6 as input
    P1SEL0 |= BIT6;  // Select Timer_A capture functionality for P1.6

    // Disable high-impedance mode
    PM5CTL0 &= ~LOCKLPM5;

    // Configure Timer_A Capture Mode on TA0.2
    TA0CCTL2 = CM_3 | CCIS_0 | SCS | CAP | CCIE;  // Capture both edges, enable interrupt
    TA0CTL = TASSEL_2 | MC_2 | TACLR;  // SMCLK, Continuous mode, Clear timer

    __bis_SR_register(LPM4_bits | GIE);  // Enter low-power mode with interrupts
    return 0;
}
```

```
// Timer_A Capture ISR
#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void) {
    if (TA0IV == TA0IV_TACCR2) { // Ensure it's TA0CCR2 interrupt
        new_time = TA0CCR2;
        cap_diff = new_time - last_time;
        last_time = new_time;

        P1OUT ^= BIT0; // Toggle LED on each button press

        __bic_SR_register_on_exit(LPM4_bits); // Exit low-power mode
    }
}
```

```c
#include <msp430f2418.h> // Specific device sizes
#include <stdint.h>

uint16_t last_time = 0; // Last time captured
uint16_t cap_diff, new_time = 0;

int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    BCSCTL1 = CALBC1_1MHZ; // Set range to calibrated 1MHz
    DCOCTL = CALDCO_1MHZ; // Set DCO step and modulation to calibrated 1MHz

    P1DIR &= ~BIT2; // P1.2 is input, others output
    P1SEL |= BIT2; // P1.2 to Timer_A CCIA

    // TAOCCTL1 - Timer A Capture Control 1, CM 3 - both edges,
    // CCIS_0 = Capture input select: 0 - CCIXA
    // SCS - Capture synchronize
    // CAP Capture Mode, CCIE - Capture/compare interrupt enable
    TA0CCTL1 = CM_3 | CCIS_0 | SCS | CAP | CCIE;
```

```
    // Start timer: SMCLK, no prescale, continuous mode, no ints, clear
    TA0CTL = TASSEL_2 | MC_2 | TACLR;

    for (;;) { // Loop forever with interrupts
        __bis_SR_register(LPM4_bits); // send controller into low power mode
    }

    return 0;
}

// Interrupt service routine for TA0CCR0_CCIFG, called on capture
#pragma vector=TIMER0_A1_VECTOR
__interrupt void port_1(void) { // Flag cleared automatically
    new_time = TA0CCR1; // Save time for next capture
    cap_diff = new_time - last_time; // Calculate time difference

    last_time = new_time; // Save time for next capture
    TA0CCTL1 &= ~CCIFG; // Clear CCIFG flag

    __bic_SR_register_on_exit(LPM4_bits); // Exit LPM4
}
```

# TASKS:

Capture event ( button press on P1.6 ) and toggle LED on each capture use rising edge.

Capture event ( button press on P1.6 ) and toggle LED on each capture use falling edge.

Capture event ( button press on P1.6 ) and toggle LED on each capture both rising and falling edge.

Display the captured event on seven segment display .
If the event occur for the first time display 1 for the second time display 2 and capture upto 9 .