Lab 7

More on Timers Capture Mode

Spring 2025

Submitted by: Mohsin Sajjad

Registration No: 22pwsce2149

Class Section: A

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: _____

Submitted to:

Engr. Faheem Jan

Month Day, Year (20 04, 2025)

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

More on Timers Capture Mode

TASKS:

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

CODE:

*main.c

```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_1 | CCIS_0 | SCS | CAP | CCIE; // Capture rising edge, 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
    }}
```

OUTPUT:

conclusion:

This code sets up a timer on the MSP430 to measure time between rising edges (like button presses) on pin P1.6. Each time an edge is detected, it calculates the time difference and toggles the LED on P1.0. It uses capture mode with interrupts and stays in low-power mode until triggered.

Task 2:

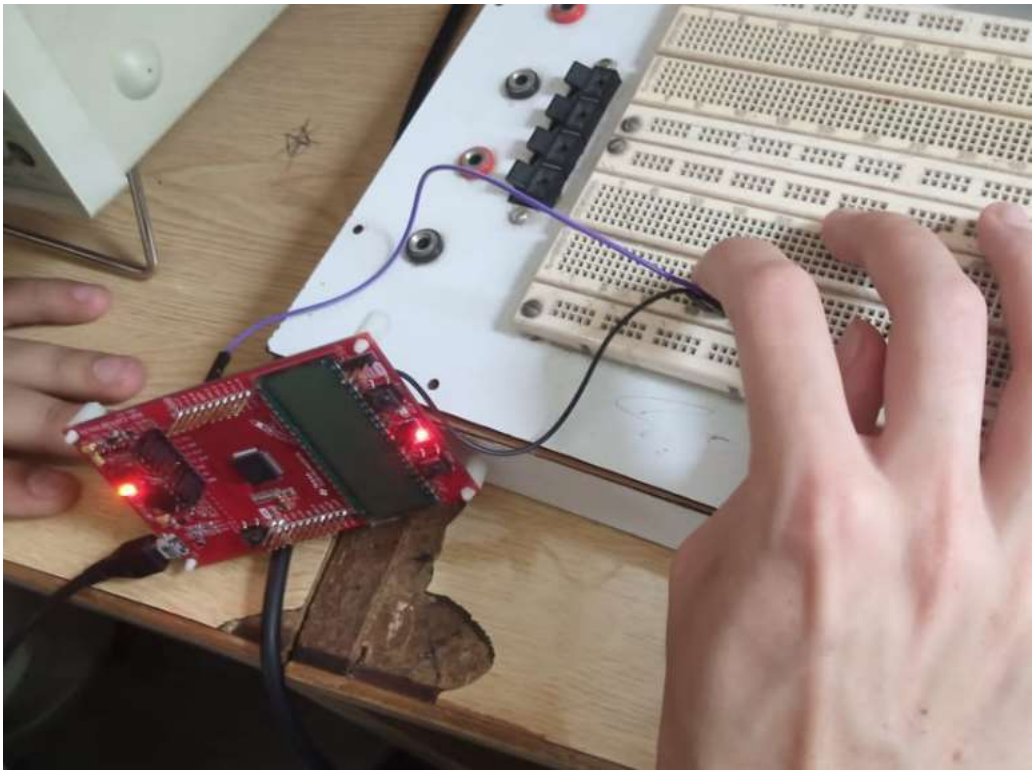
Capture event (button press on P1.6) and toggle LED on each capture use falling edge.

CODE:

*main.c

```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_2 | CCIS_0 | SCS | CAP | CCIE; // Capture falling edge, 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
    }
```

OUTPUT:



CONCLUSION:

This code measures the time between falling edges (like button releases) on pin P1.6 using Timer_A in capture mode. Each time a falling edge is detected, it calculates the time difference and toggles the LED on P1.0. It runs in low-power mode and wakes up only when an interrupt occurs.

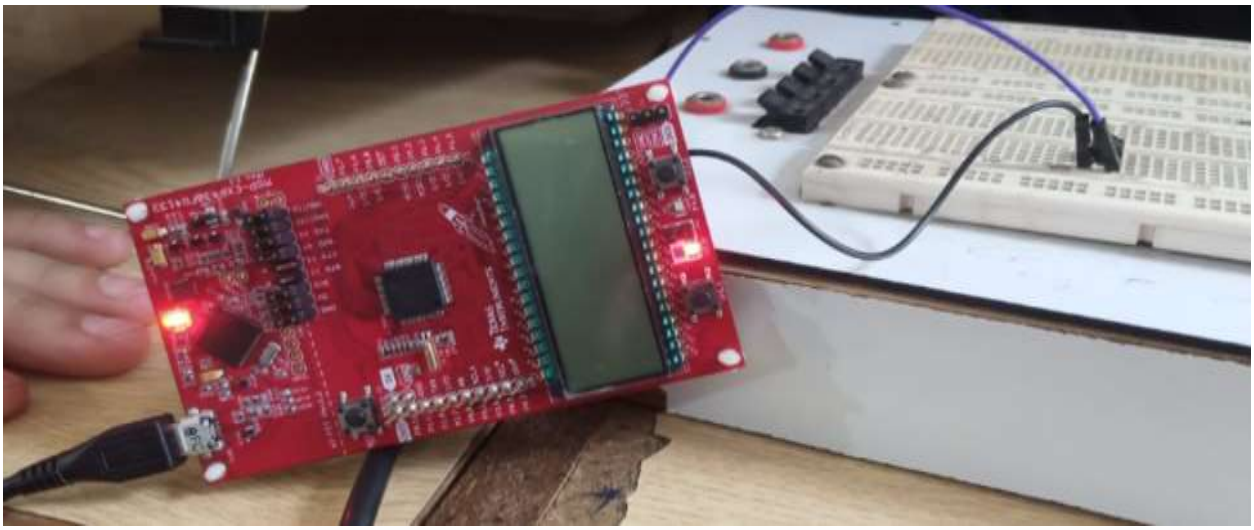
TASK 03:

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

CODE:

```c
*main.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
    }}
```

OUTPUT:



CONCLUSION:

This code uses Timer_A to capture both rising and falling edges on pin P1.6. It measures the time between each edge and toggles the LED on P1.0. The microcontroller stays in low-power mode and wakes up only when an edge is detected.

Task 04:

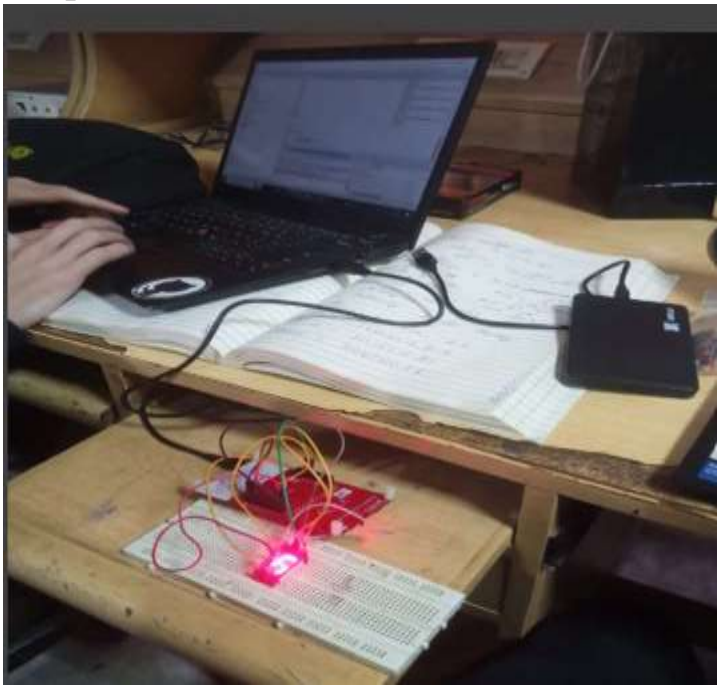
Display the captured event on seven segment display . If the event occur for the first time display 1 for the second time display 2 and capture upto 9 .

CODE:

*main.c

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

uint16_t last_time = 0;
uint16_t cap_diff, new_time = 0;
uint8_t event_count = 0;

// Function to display number on 7-segment using P1.0 - P1.6
void display_number(uint8_t num) {
    switch(num) {
        case 0: P1OUT = 0b01111011; break;  // display 0
        case 1: P1OUT = 0b00001010; break;  // display 1
        case 2: P1OUT = 0b10110011; break;  // display 2
        case 3: P1OUT = 0b10011011; break;  // display 3
        case 4: P1OUT = 0b11001010; break;  // display 4
        case 5: P1OUT = 0b11011001; break;  // display 5
        case 6: P1OUT = 0b11111001; break;  // display 6
        case 7: P1OUT = 0b00001011; break;  // display 7
        case 8: P1OUT = 0b11111011; break;  // display 8
        case 9: P1OUT = 0b11011011; break;  // display 9
        default: P1OUT &= 0x80; break;      // turn off segments (preserve BIT7 if needed)
    }
}

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

    // P1.0 - P1.6 for 7-segment output
    P1DIR |= 0x7F;              // Set P1.0 - P1.6 as output
    P1OUT &= ~0x7F;             // Clear output (all segments OFF)

    // LED on P1.7 (optional for debugging toggle)
    P1DIR |= BIT7;
    P1OUT &= ~BIT7;

    // Set P1.6 as input for Timer Capture
    P2DIR &= ~BIT6;
    P2SEL0 |= BIT6;

    // Enable GPIOs
    PM5CTL0 &= ~LOCKLPM5;

    // Timer Capture Configuration
    TA0CCTL2 = CM_3 | CCIS_0 | SCS | CAP | CCIE;  // Capture on both edges, enable interrupt
    TA0CTL = TASSEL_2 | MC_2 | TACLR;             // SMCLK, Continuous mode, clear timer

    __bis_SR_register(LPM4_bits | GIE);           // Low-power mode with global interrupt enable
    return 0;
}

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

        if (event_count < 9) {
            event_count++;
            display_number(event_count);
        }

        P1OUT ^= BIT7;  // Toggle LED on P1.7 to show event

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

Output:



Conclusion:

This code captures events on a rising or falling edge from a timer input (P1.6) and displays the count on a 7-segment display (P1.0 - P1.6). The event count is displayed from 1 to 9, and the LED (P1.7) toggles with each event. The system operates in low-power mode with interrupts enabled, efficiently counting and displaying without delays.