Lab 9

Generating a PWM Waveform

Spring 2025

Submitted by: Mohsin Sajjad

Registration No: 22pwsce2149

Class Section: A

Mohsin Sajjad

Student Signature: ______________

Submitted to:

Engr. Faheem Jan

Month Day, Year (11 05, 2025)

Department of Computer Systems Engineering

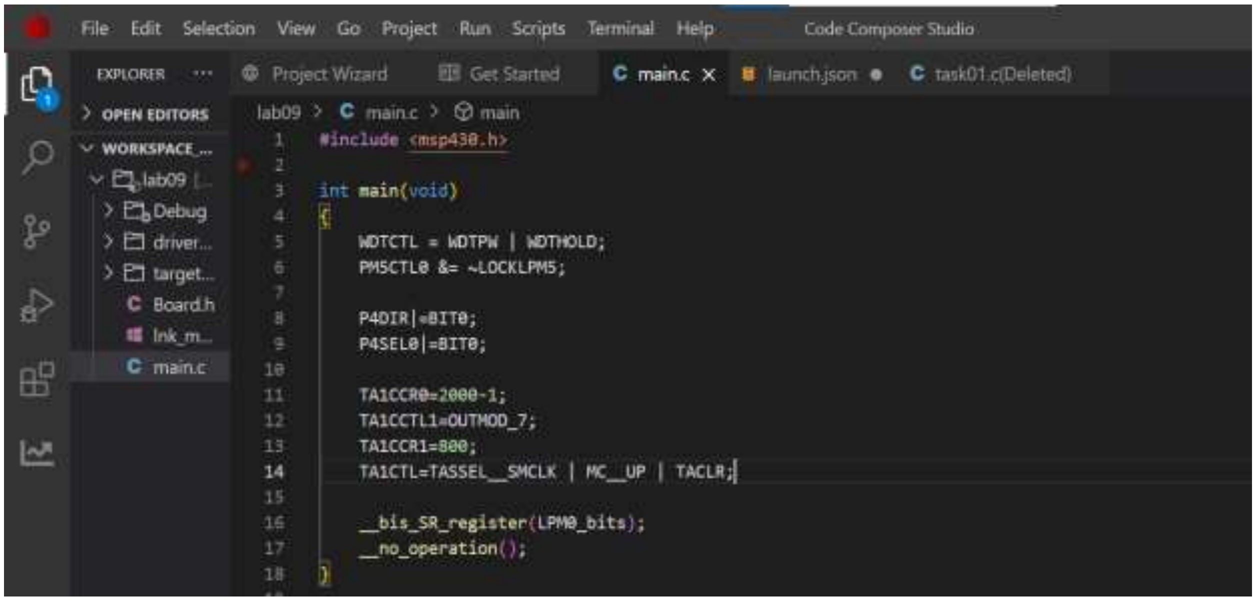
University of Engineering and Technology, Peshawar

Generating a PWM Waveform

TASKS:

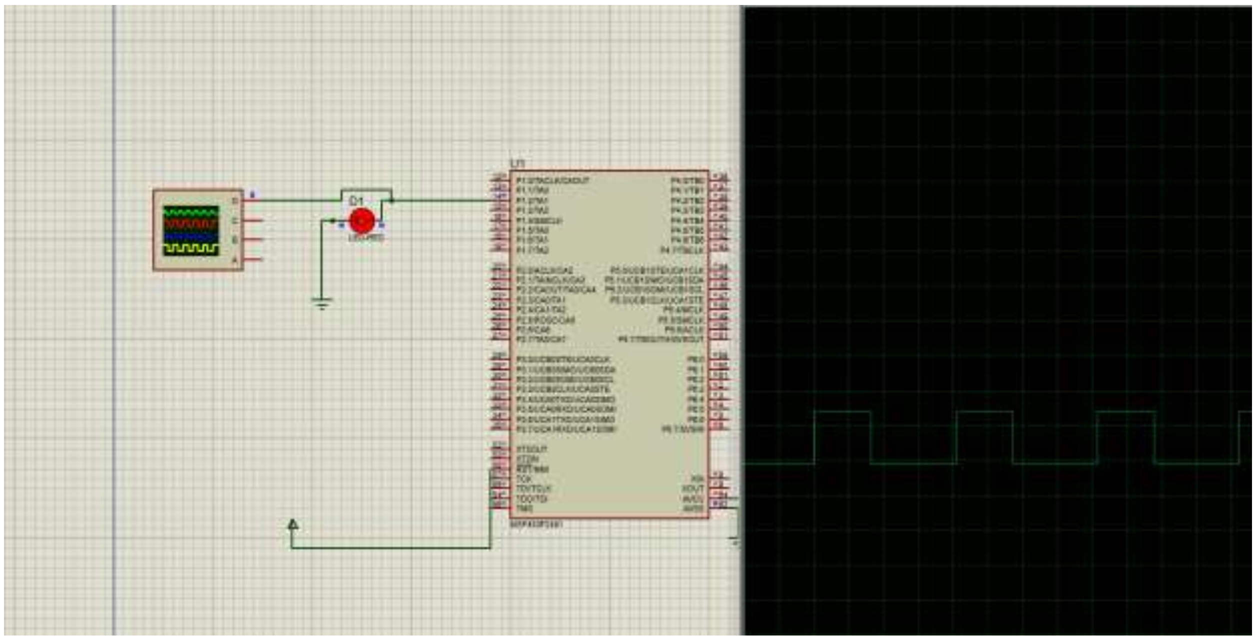
1) Generate a signal of 500Hz with 40% duty cycle.

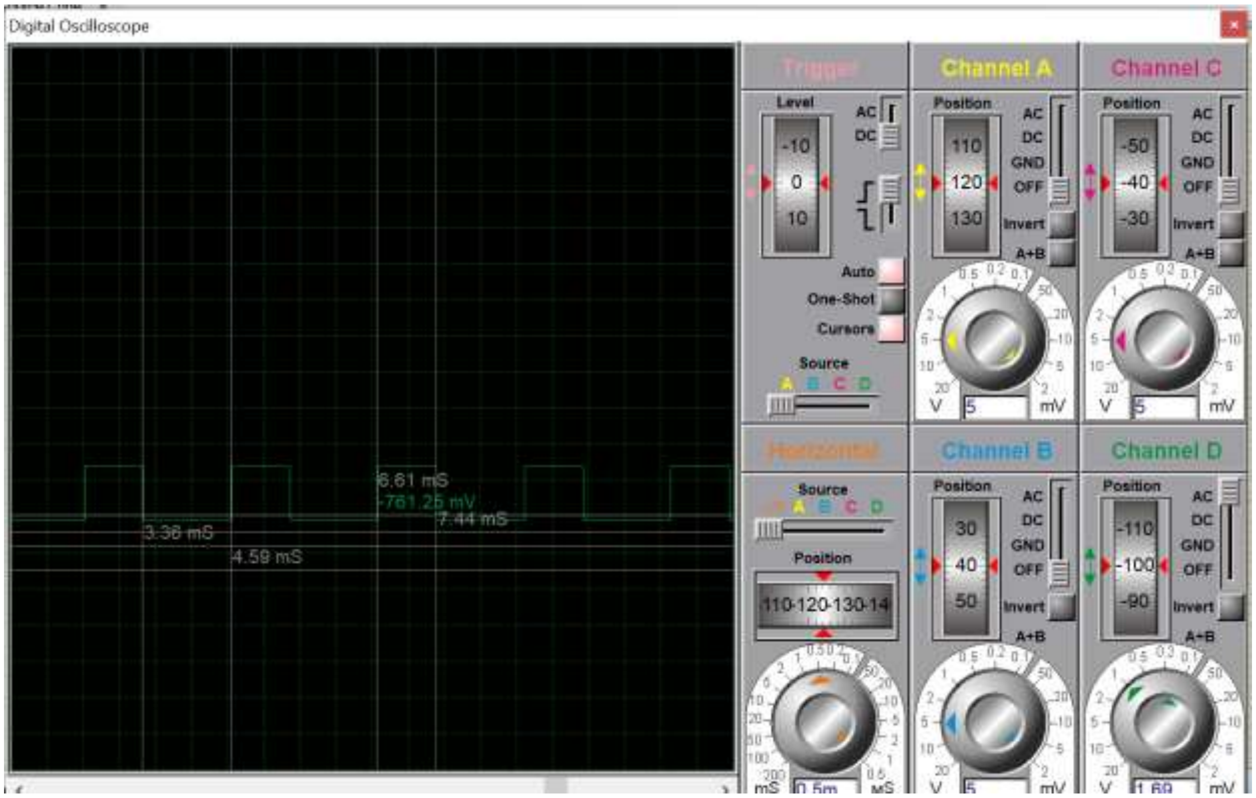
CODE:



```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;

    P4DIR|=BIT0;
    P4SEL0|=BIT0;

    TA1CCR0=2000-1;
    TA1CCTL1=OUTMOD_7;
    TA1CCR1=800;
    TA1CTL=TASSEL__SMCLK | MC__UP | TACLR;

    __bis_SR_register(LPM0_bits);
    __no_operation();
}
```

OUTPUT:

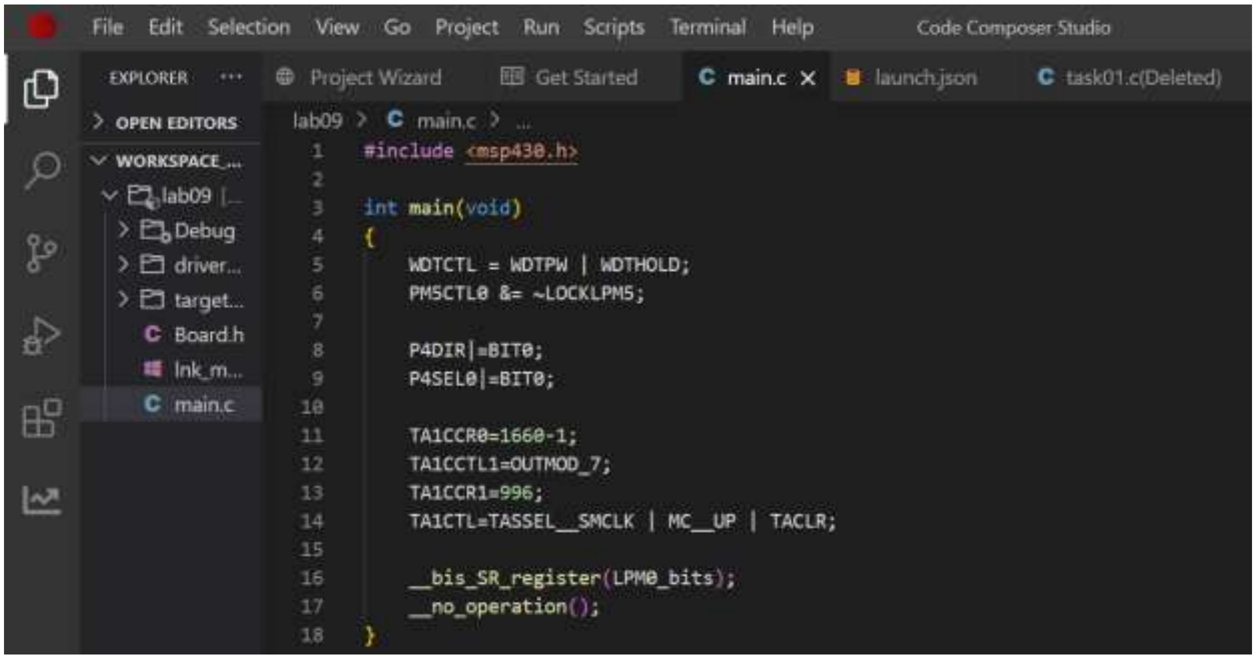




Task 2:

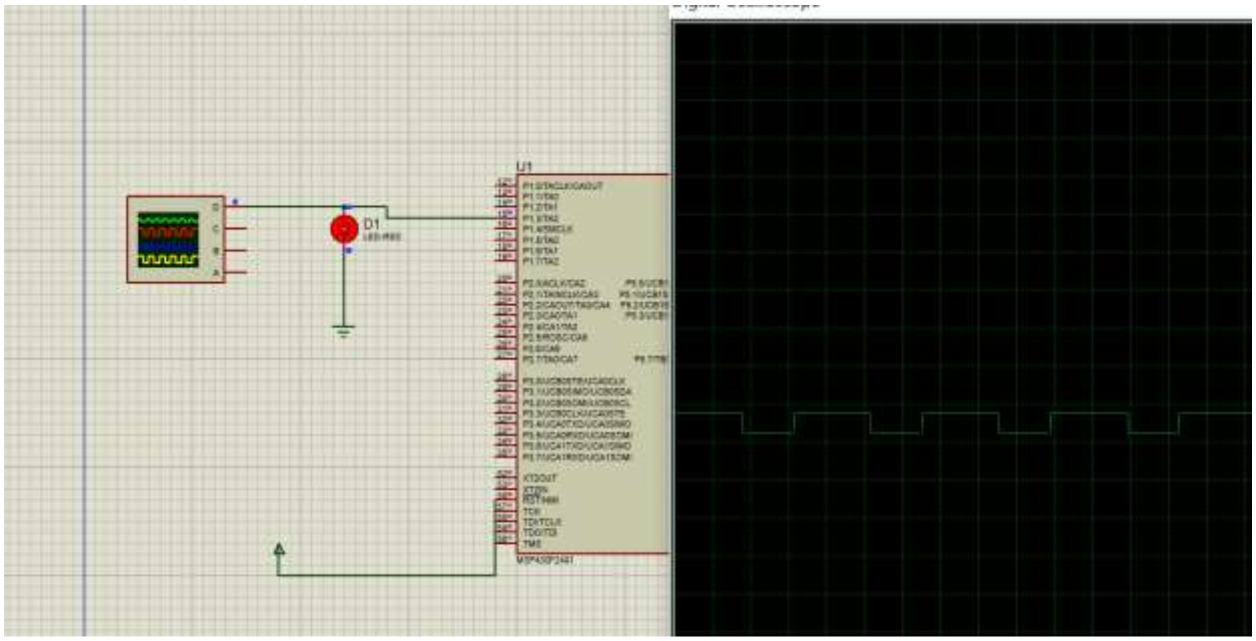
Generate a signal of 600Hz with 60% duty cycle on P1.3 Hint: use timer.

CODE:



```c
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;

    P4DIR|=BIT0;
    P4SEL0|=BIT0;

    TA1CCR0=1660-1;
    TA1CCTL1=OUTMOD_7;
    TA1CCR1=996;
    TA1CTL=TASSEL__SMCLK | MC__UP | TACLR;

    __bis_SR_register(LPM0_bits);
    __no_operation();
}
```

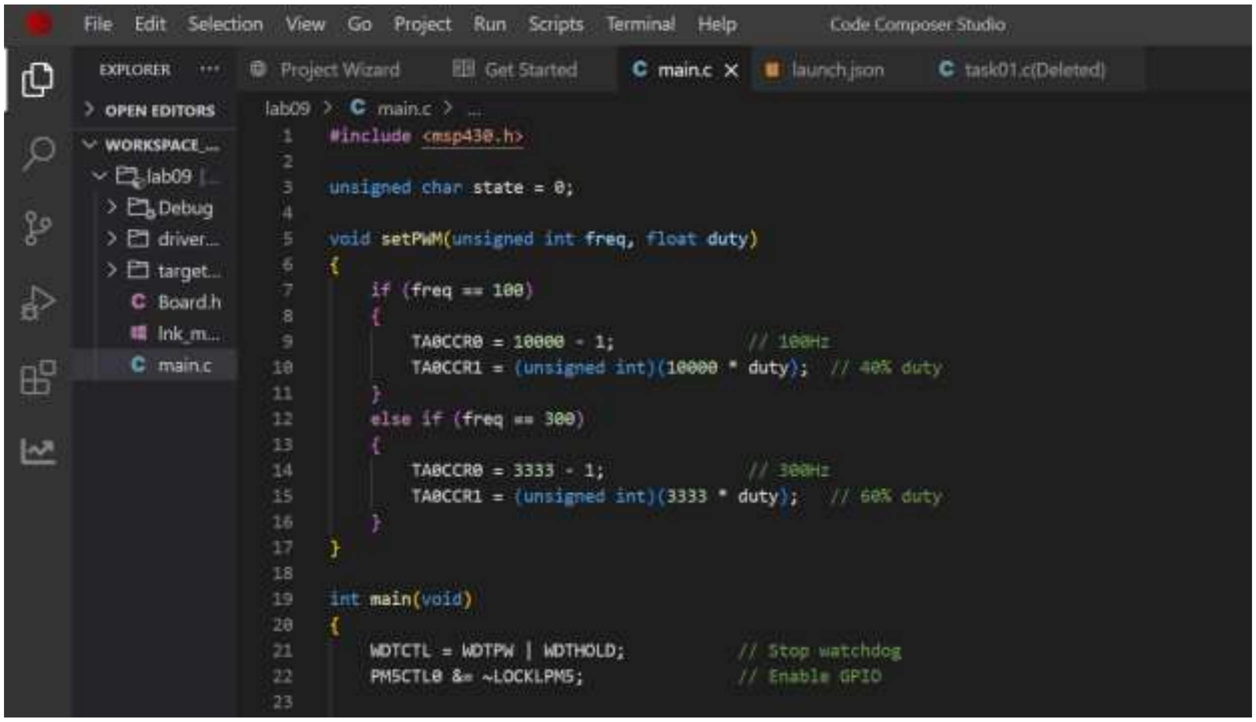
OUTPUT:



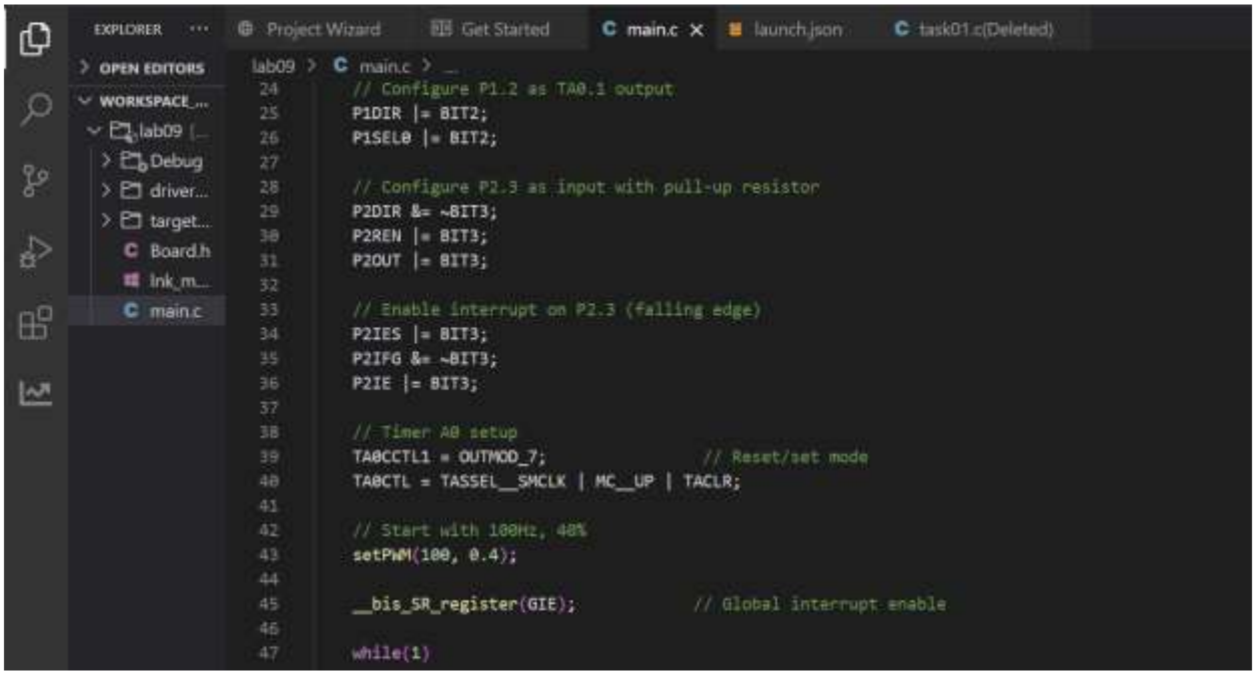
TASK 03:

Generate a signal of 100Hz with 40% duty cycle on P1.2 When a user presses a button at P2.3 the signal change to 300Hz with 60% duty cycle. Button pressed means Press and release.

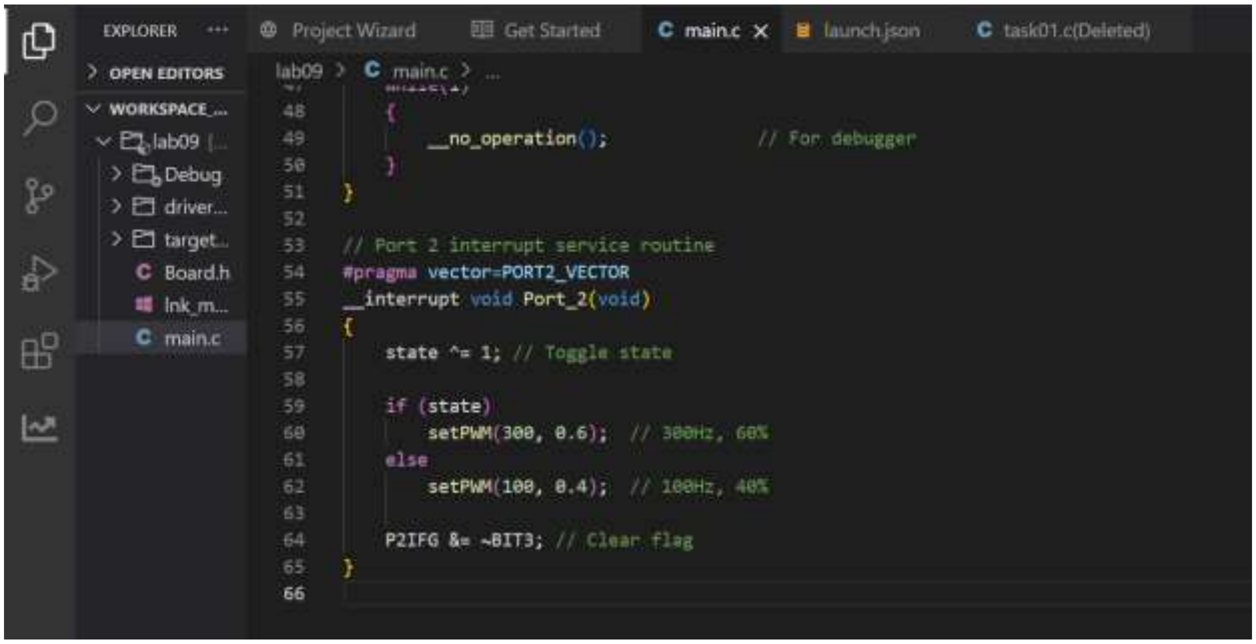
CODE:



```c
#include <msp430.h>

unsigned char state = 0;

void setPWM(unsigned int freq, float duty)
{
    if (freq == 100)
    {
        TA0CCR0 = 10000 - 1;            // 100Hz
        TA0CCR1 = (unsigned int)(10000 * duty);  // 40% duty
    }
    else if (freq == 300)
    {
        TA0CCR0 = 3333 - 1;             // 300Hz
        TA0CCR1 = (unsigned int)(3333 * duty);   // 60% duty
    }
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog
    PM5CTL0 &= ~LOCKLPM5;              // Enable GPIO

```



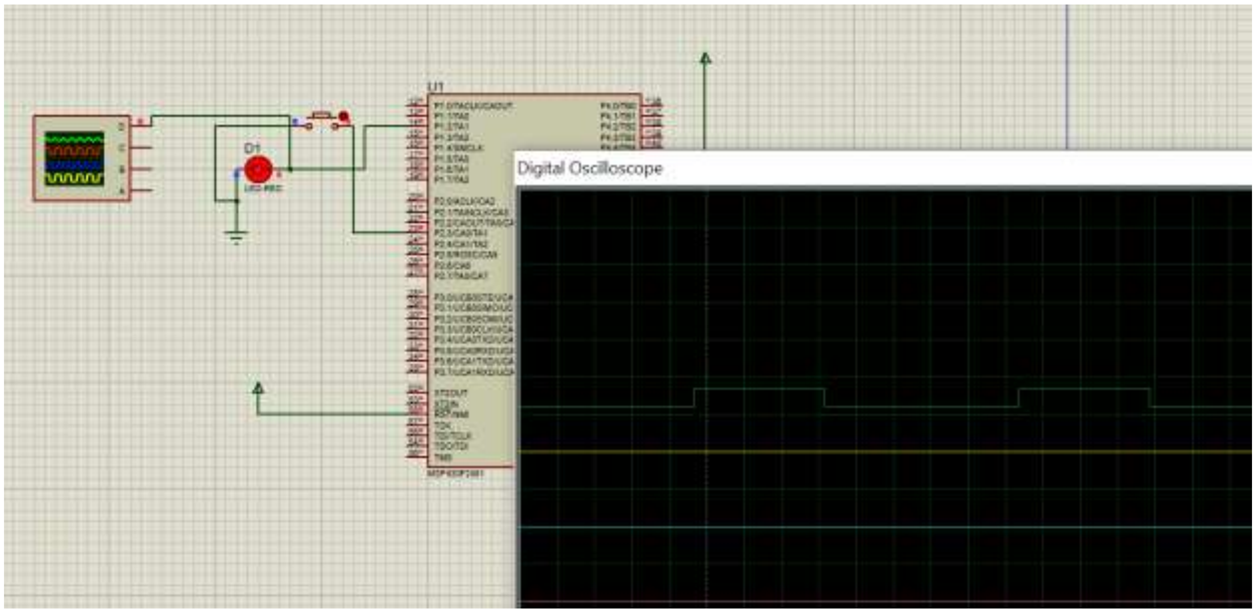
```c
    // Configure P1.2 as TA0.1 output
    P1DIR |= BIT2;
    P1SEL0 |= BIT2;

    // Configure P2.3 as input with pull-up resistor
    P2DIR &= ~BIT3;
    P2REN |= BIT3;
    P2OUT |= BIT3;

    // Enable interrupt on P2.3 (falling edge)
    P2IES |= BIT3;
    P2IFG &= ~BIT3;
    P2IE |= BIT3;

    // Timer A0 setup
    TA0CCTL1 = OUTMOD_7;                // Reset/set mode
    TA0CTL = TASSEL__SMCLK | MC__UP | TACLR;

    // Start with 100Hz, 40%
    setPWM(100, 0.4);

    __bis_SR_register(GIE);             // Global interrupt enable

    while(1)
```



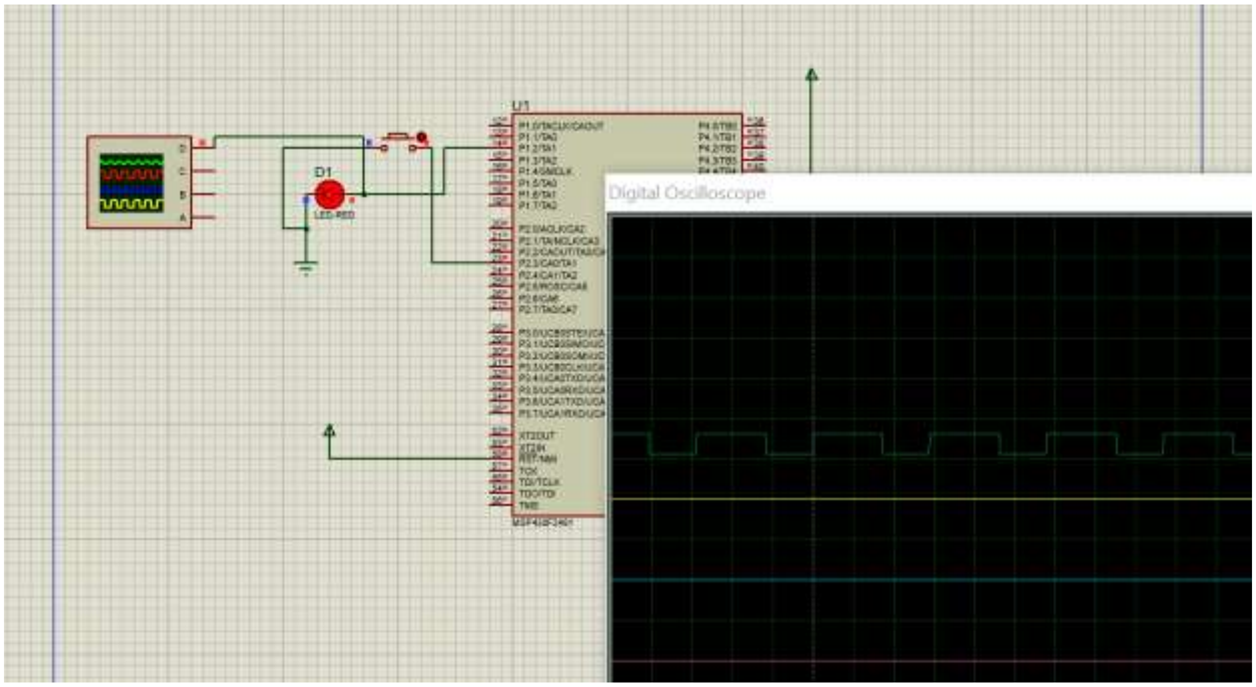
```c
    {
        __no_operation();               // For debugger
    }
}

// Port 2 interrupt service routine
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
    state ^= 1; // Toggle state

    if (state)
        setPWM(300, 0.6);  // 300Hz, 60%
    else
        setPWM(100, 0.4);  // 100Hz, 40%

    P2IFG &= ~BIT3; // Clear flag
}
```

OUTPUT:

When button not pressed.



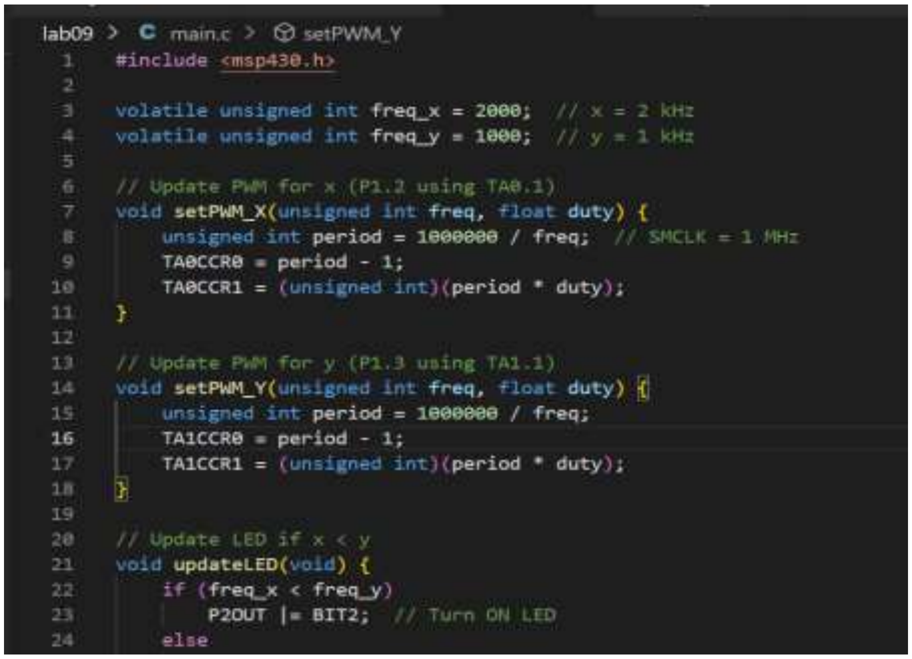
When button pressed.



Task 04:

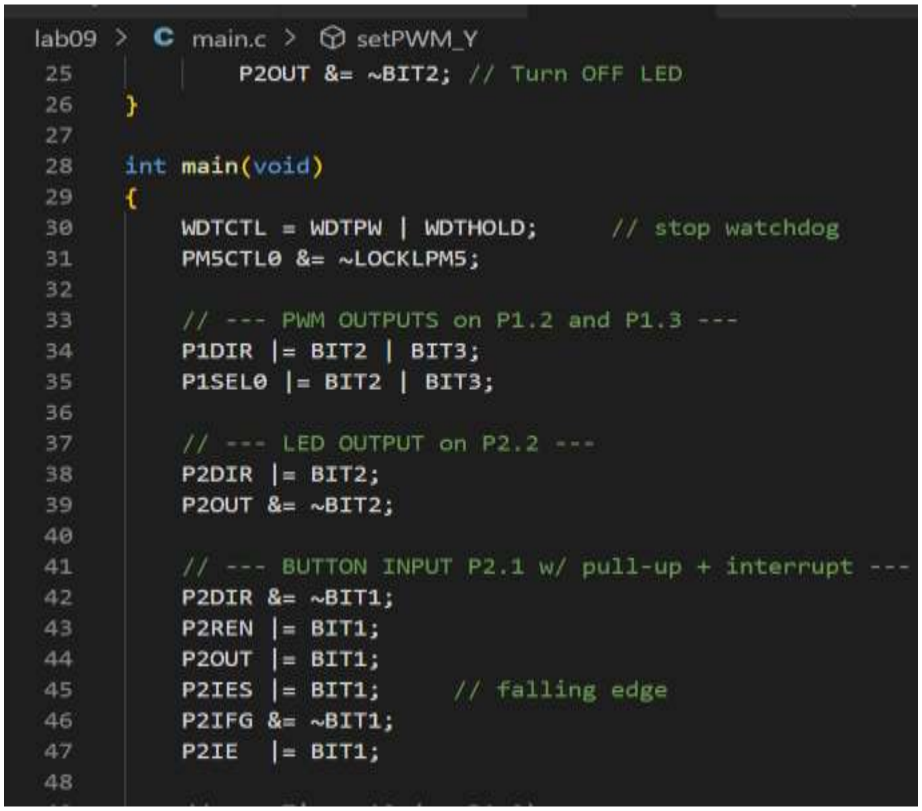
This task consists of two parts, A. Generate a signal x of 2KHz with 75% duty cycle on P1.2. Similarly, generate another signal y of 1KHz with 25% duty cycle on P1.3. As soon a user presses a button on P2.1, x frequency drops by 100Hz and y increases by 100Hz. If x crosses y, an LED at P2.2 is turned ON. Use low power mode when nothing is happening. Additionally, use interrupts and not polling in your program. a. Use Timer interrupt for delay creation

CODE:



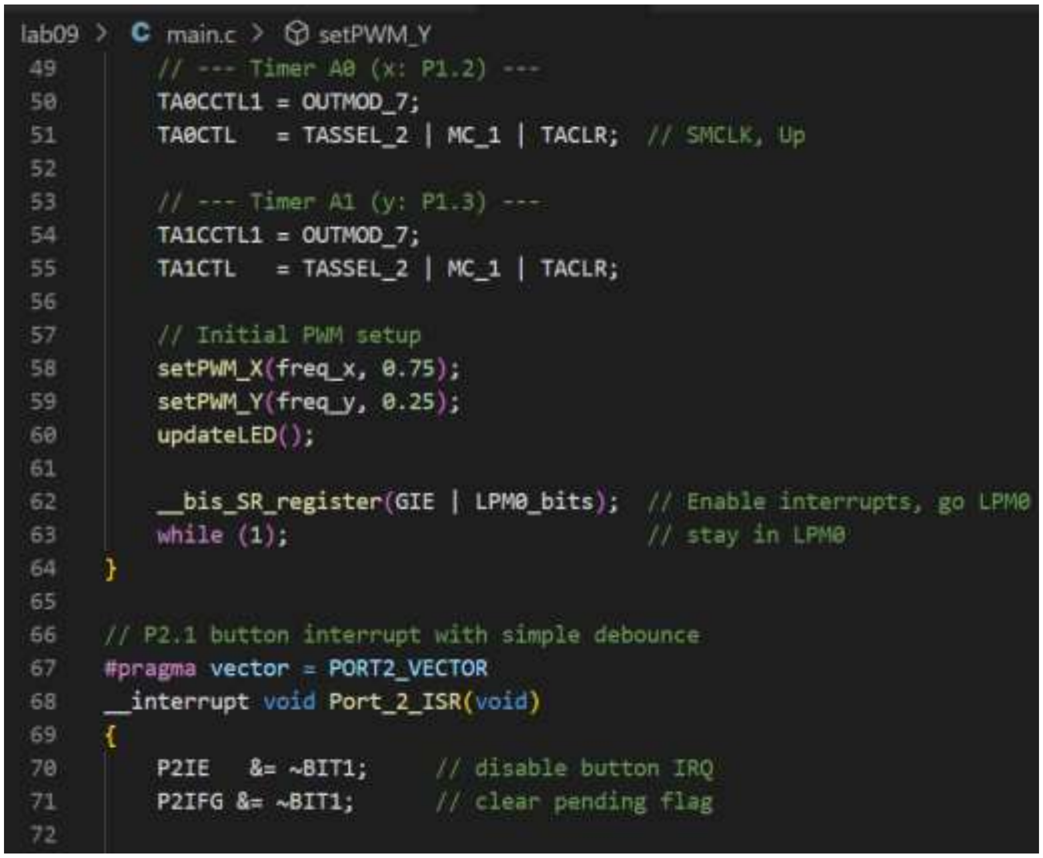
lab09 > main.c > setPWM_Y

```c
#include <msp430.h>

volatile unsigned int freq_x = 2000;  // x = 2 kHz
volatile unsigned int freq_y = 1000;  // y = 1 kHz

// Update PWM for x (P1.2 using TA0.1)
void setPWM_X(unsigned int freq, float duty) {
    unsigned int period = 1000000 / freq;  // SMCLK = 1 MHz
    TA0CCR0 = period - 1;
    TA0CCR1 = (unsigned int)(period * duty);
}

// Update PWM for y (P1.3 using TA1.1)
void setPWM_Y(unsigned int freq, float duty) {
    unsigned int period = 1000000 / freq;
    TA1CCR0 = period - 1;
    TA1CCR1 = (unsigned int)(period * duty);
}

// Update LED if x < y
void updateLED(void) {
    if (freq_x < freq_y)
        P2OUT |= BIT2;  // Turn ON LED
    else
```



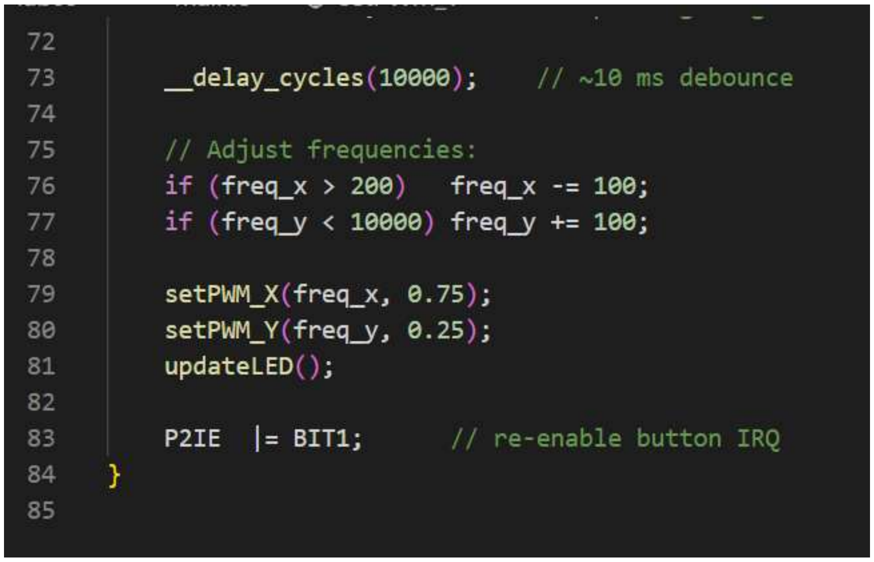
lab09 > main.c > setPWM_Y

```c
        P2OUT &= ~BIT2; // Turn OFF LED
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;     // stop watchdog
    PM5CTL0 &= ~LOCKLPM5;

    // --- PWM OUTPUTS on P1.2 and P1.3 ---
    P1DIR |= BIT2 | BIT3;
    P1SEL0 |= BIT2 | BIT3;

    // --- LED OUTPUT on P2.2 ---
    P2DIR |= BIT2;
    P2OUT &= ~BIT2;

    // --- BUTTON INPUT P2.1 w/ pull-up + interrupt ---
    P2DIR &= ~BIT1;
    P2REN |= BIT1;
    P2OUT |= BIT1;
    P2IES |= BIT1;     // falling edge
    P2IFG &= ~BIT1;
    P2IE  |= BIT1;

```



lab09 > main.c > setPWM_Y

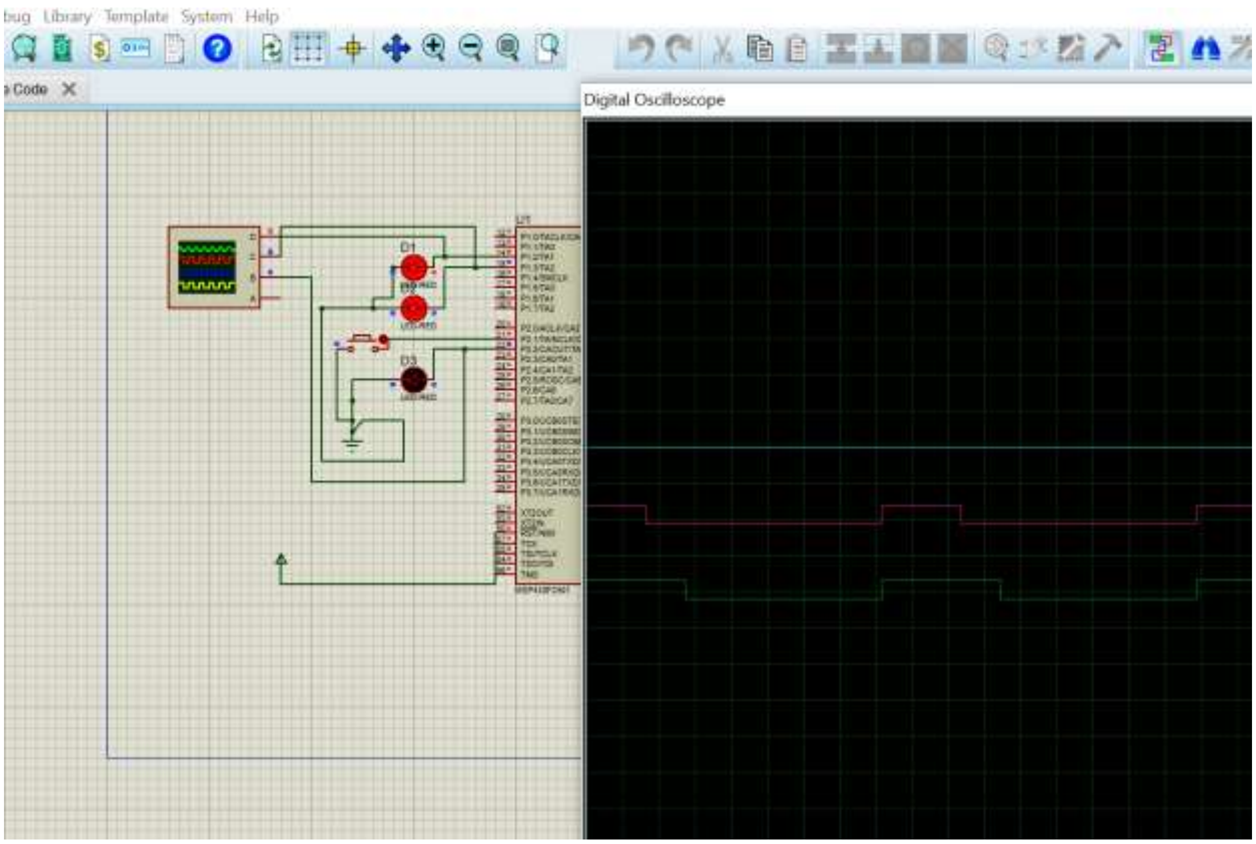
```c
    // --- Timer A0 (x: P1.2) ---
    TA0CCTL1 = OUTMOD_7;
    TA0CTL   = TASSEL_2 | MC_1 | TACLR;  // SMCLK, Up

    // --- Timer A1 (y: P1.3) ---
    TA1CCTL1 = OUTMOD_7;
    TA1CTL   = TASSEL_2 | MC_1 | TACLR;

    // Initial PWM setup
    setPWM_X(freq_x, 0.75);
    setPWM_Y(freq_y, 0.25);
    updateLED();

    __bis_SR_register(GIE | LPM0_bits);  // Enable interrupts, go LPM0
    while (1);                           // stay in LPM0
}

// P2.1 button interrupt with simple debounce
#pragma vector = PORT2_VECTOR
__interrupt void Port_2_ISR(void)
{
    P2IE  &= ~BIT1;     // disable button IRQ
    P2IFG &= ~BIT1;     // clear pending flag
```



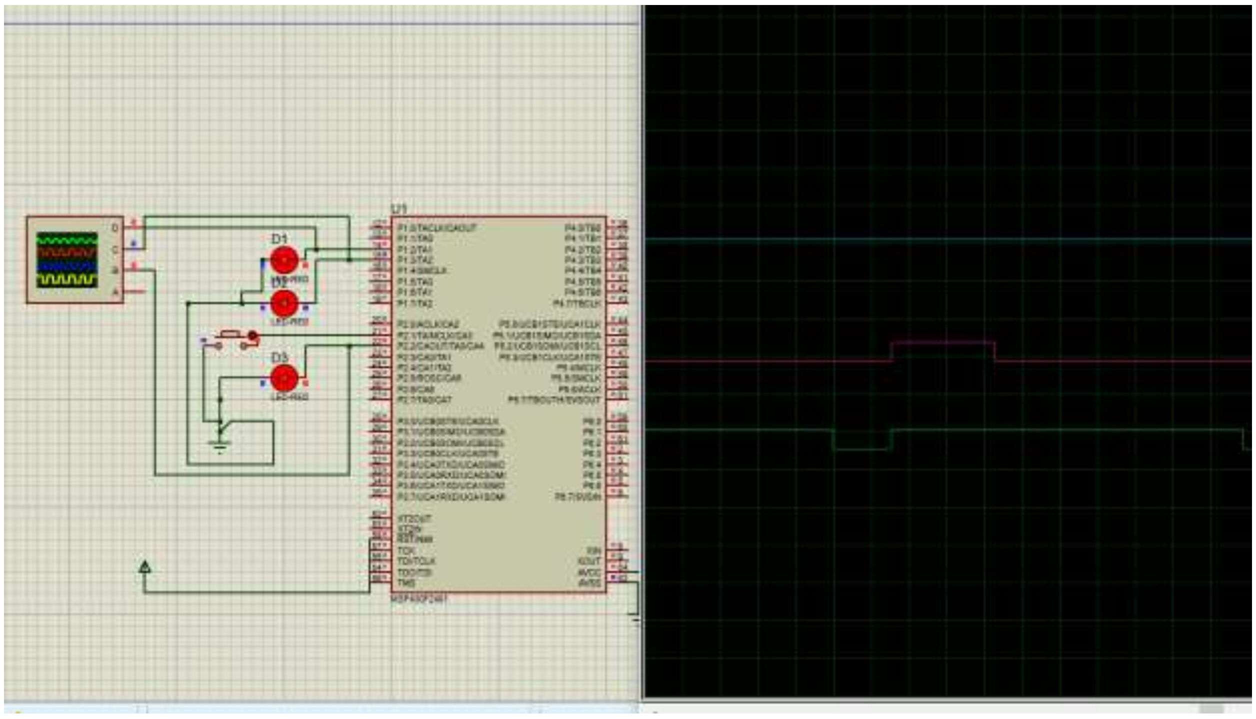
```c
    __delay_cycles(10000);    // ~10 ms debounce

    // Adjust frequencies:
    if (freq_x > 200)   freq_x -= 100;
    if (freq_y < 10000) freq_y += 100;

    setPWM_X(freq_x, 0.75);
    setPWM_Y(freq_y, 0.25);
    updateLED();

    P2IE  |= BIT1;      // re-enable button IRQ
}
```

Output:

Before crossing of x and y



After crossing of x and y


U1
D1
D2
D3
LED-RED
D
C
B
A
TCK
TMS