LAB5

Interrupts using msp430 microcontroller and interrupt based Knight rider

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."

[illegible]

Student Signature: ______________

Submitted to:

Engr. Faheem Jan

Month Day, Year (23 03, 2025)

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Interrupt:

• Anexternal event that interrupt the microcontroller that a device need its service

• Whenever a device needs its service, the device notify the microcontroller by sending it an interrupt signal

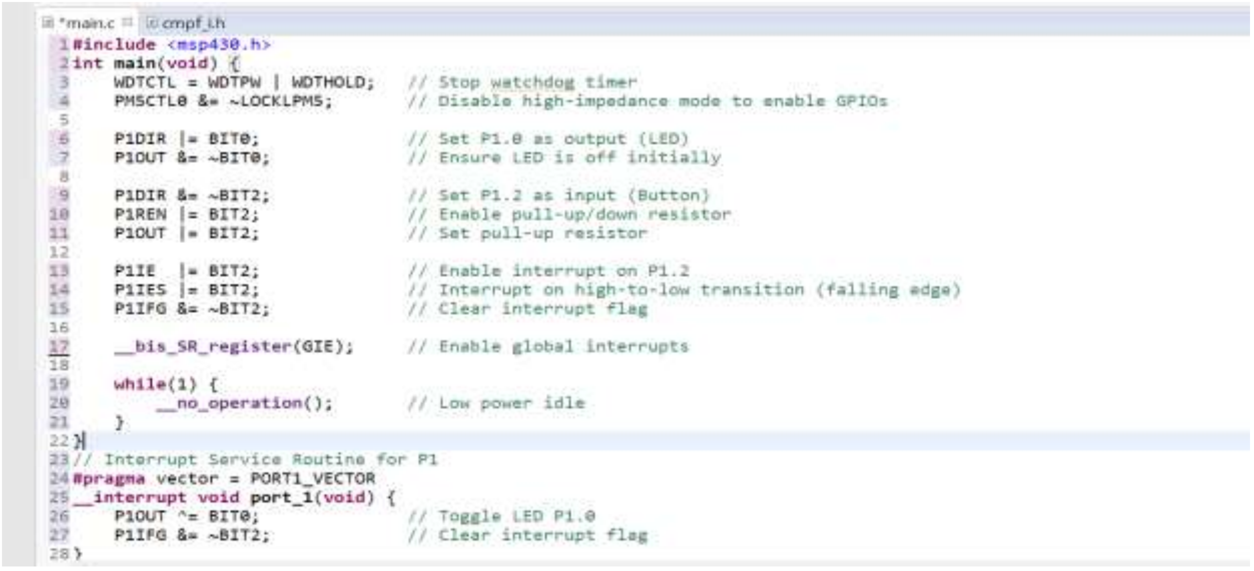
• Uponreceiving The microcontroller interrupt whatever it is doing and service the device

• The program associated with interrupt is called ISR(Interrupt Service Routine)

TASKS:

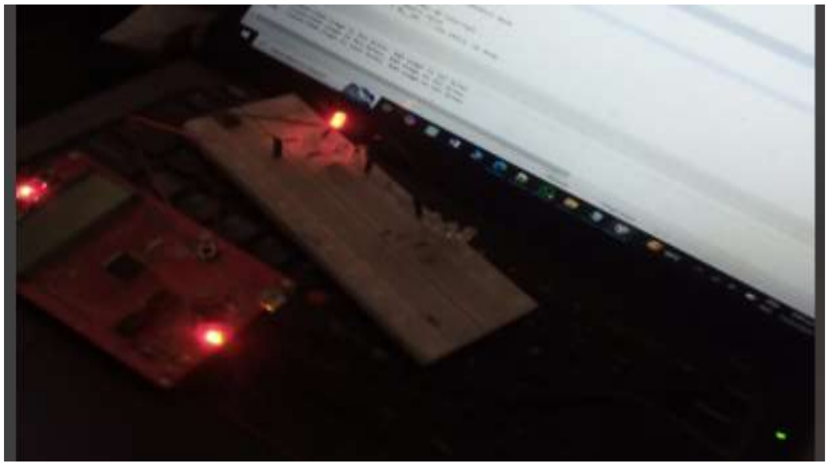
TASK 01:

Write Msp430 program which toggle LED P1.0 when an interrupt occur at P1.2



```c
#include <msp430.h>
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       // Disable high-impedance mode to enable GPIOs

    P1DIR |= BIT0;              // Set P1.0 as output (LED)
    P1OUT &= ~BIT0;             // Ensure LED is off initially

    P1DIR &= ~BIT2;             // Set P1.2 as input (Button)
    P1REN |= BIT2;              // Enable pull-up/down resistor
    P1OUT |= BIT2;              // Set pull-up resistor

    P1IE  |= BIT2;              // Enable interrupt on P1.2
    P1IES |= BIT2;              // Interrupt on high-to-low transition (falling edge)
    P1IFG &= ~BIT2;             // Clear interrupt flag

    __bis_SR_register(GIE);     // Enable global interrupts

    while(1) {
        __no_operation();       // Low power idle
    }
}
// Interrupt Service Routine for P1
#pragma vector = PORT1_VECTOR
__interrupt void port_1(void) {
    P1OUT ^= BIT0;              // Toggle LED P1.0
    P1IFG &= ~BIT2;             // Clear interrupt flag
}
```

Output:



Conclusion:

This MSP430 program sets up P1.2 as a button input and P1.0 as an LED output. When the button is pressed, an interrupt is triggered, toggling the LED state.

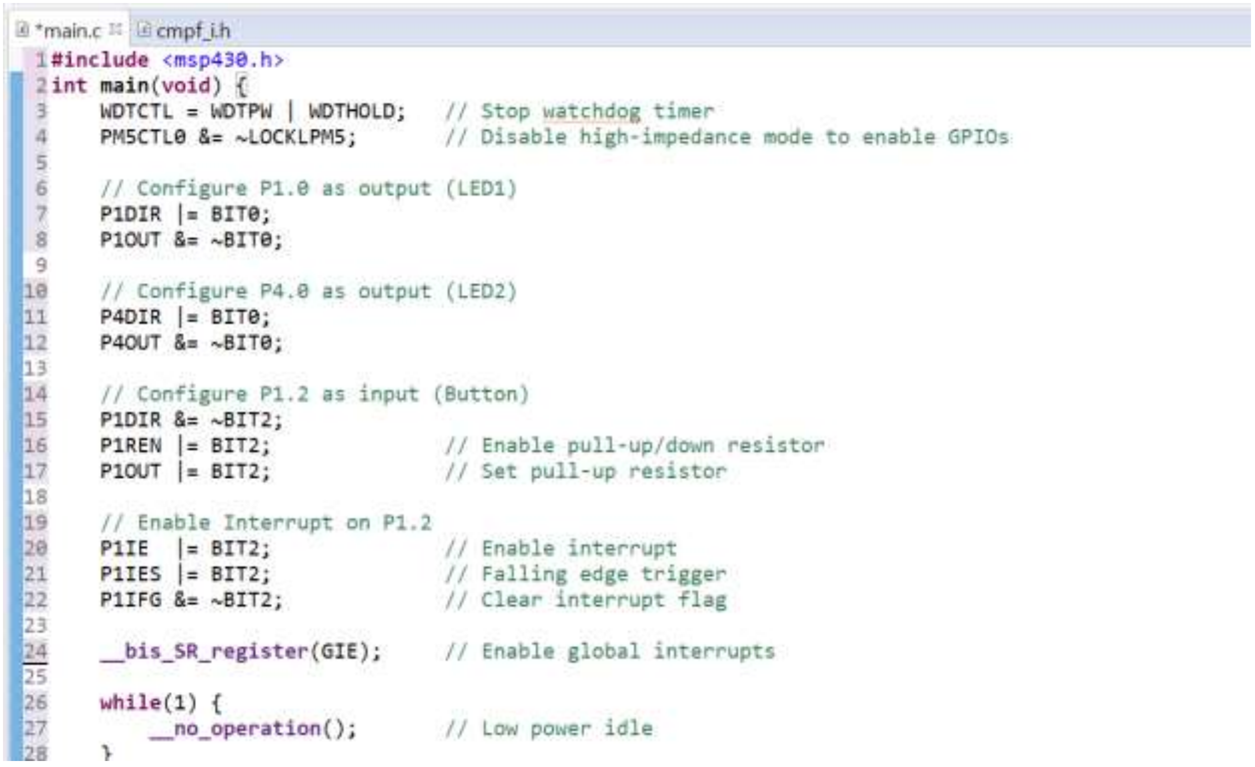
1. Stops the watchdog timer and unlocks GPIOs.
2. Configures the LED (P1.0) as output and ensures it's initially off.
3. Sets up P1.2 as an input with a pull-up resistor.
4. Enables interrupts on P1.2, triggering on a button press (falling edge).
5. Interrupt Service Routine (ISR) toggles the LED whenever the button is pressed.

This is a basic interrupt-based button press LED toggle program for low-power embedded systems.

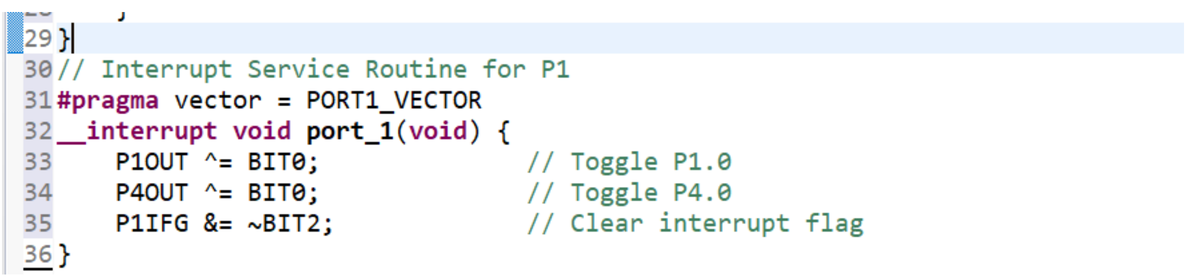
Task 02:

Toggle LED p1.0 and P4.0 when an interrupt occurs at P1.2

CODE:

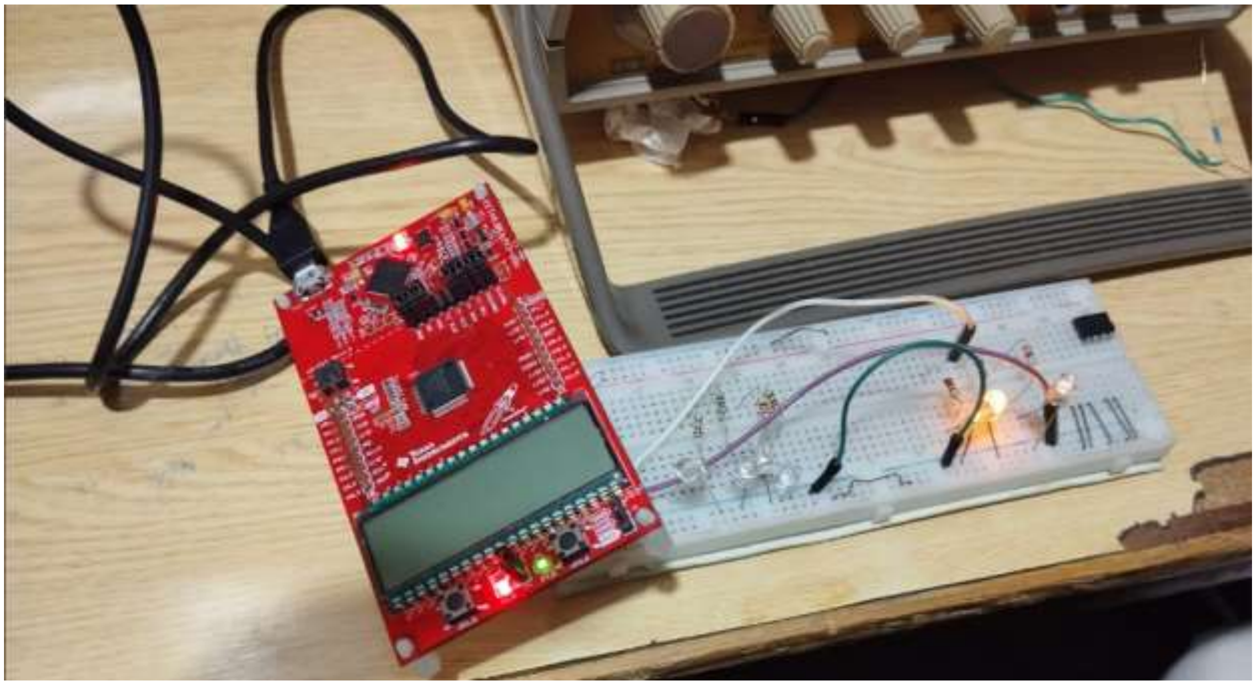


```
#include <msp430.h>
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       // Disable high-impedance mode to enable GPIOs

    // Configure P1.0 as output (LED1)
    P1DIR |= BIT0;
    P1OUT &= ~BIT0;

    // Configure P4.0 as output (LED2)
    P4DIR |= BIT0;
    P4OUT &= ~BIT0;

    // Configure P1.2 as input (Button)
    P1DIR &= ~BIT2;
    P1REN |= BIT2;              // Enable pull-up/down resistor
    P1OUT |= BIT2;              // Set pull-up resistor

    // Enable Interrupt on P1.2
    P1IE  |= BIT2;              // Enable interrupt
    P1IES |= BIT2;              // Falling edge trigger
    P1IFG &= ~BIT2;             // Clear interrupt flag

    __bis_SR_register(GIE);     // Enable global interrupts

    while(1) {
        __no_operation();       // Low power idle
    }
```



```
    }
}
// Interrupt Service Routine for P1
#pragma vector = PORT1_VECTOR
__interrupt void port_1(void) {
    P1OUT ^= BIT0;              // Toggle P1.0
    P4OUT ^= BIT0;              // Toggle P4.0
    P1IFG &= ~BIT2;             // Clear interrupt flag
}
```

Output:



Conclusion:

This MSP430 program controls two LEDs (P1.0 and P4.0) using a button (P1.2) with an interrupt-based approach.

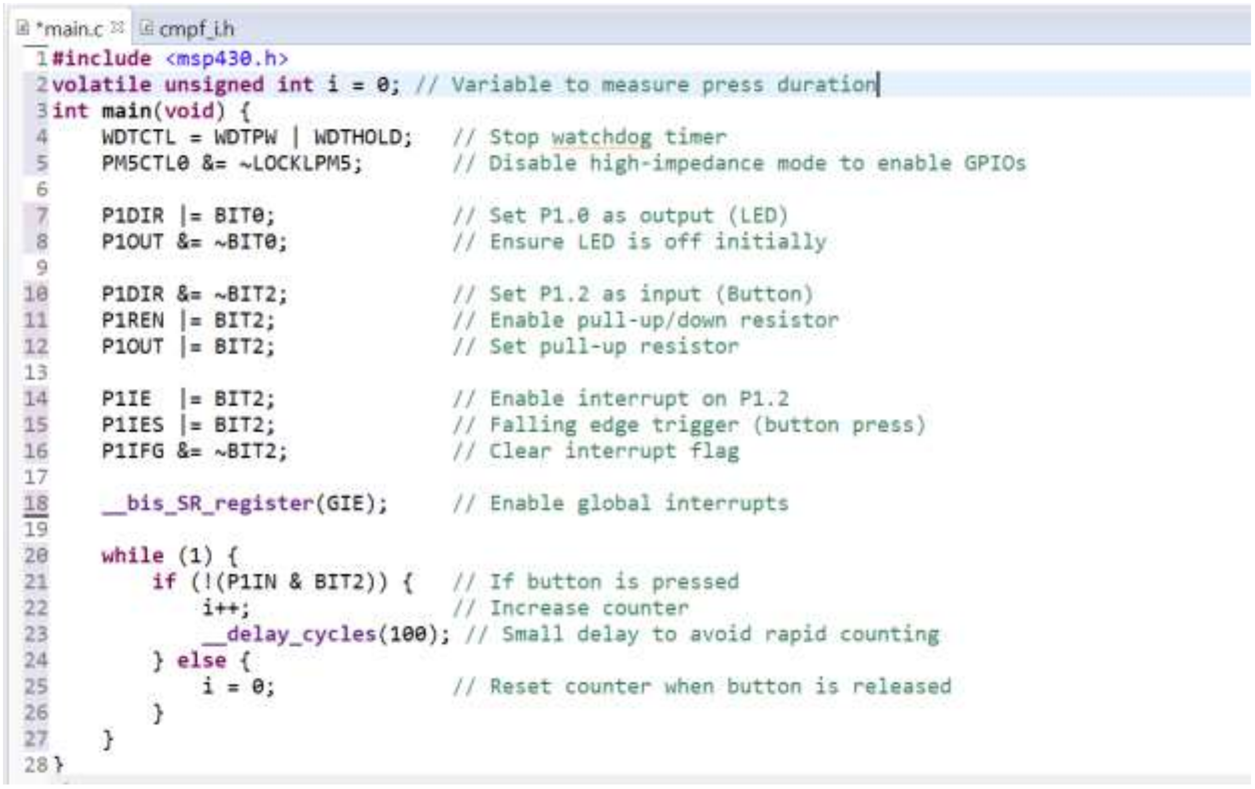
1. Stops the watchdog timer and unlocks GPIOs.
2. Configures P1.0 (LED1) and P4.0 (LED2) as outputs, initially turning them off.
3. Sets up P1.2 as an input with a pull-up resistor for the button.
4. Enables an interrupt on P1.2, triggering on a button press (falling edge).
5. The ISR toggles both LEDs whenever the button is pressed.

This allows low-power operation and ensures LEDs change state only when the button is pressed.

Task 03:

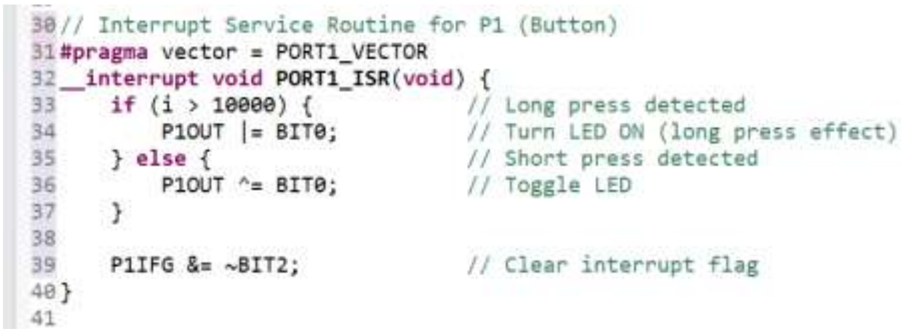
Detect short or long press of button if button is pressed for longer time the LED should remain ON if the button is pressed for a shorter time it should toggle.

CODE:



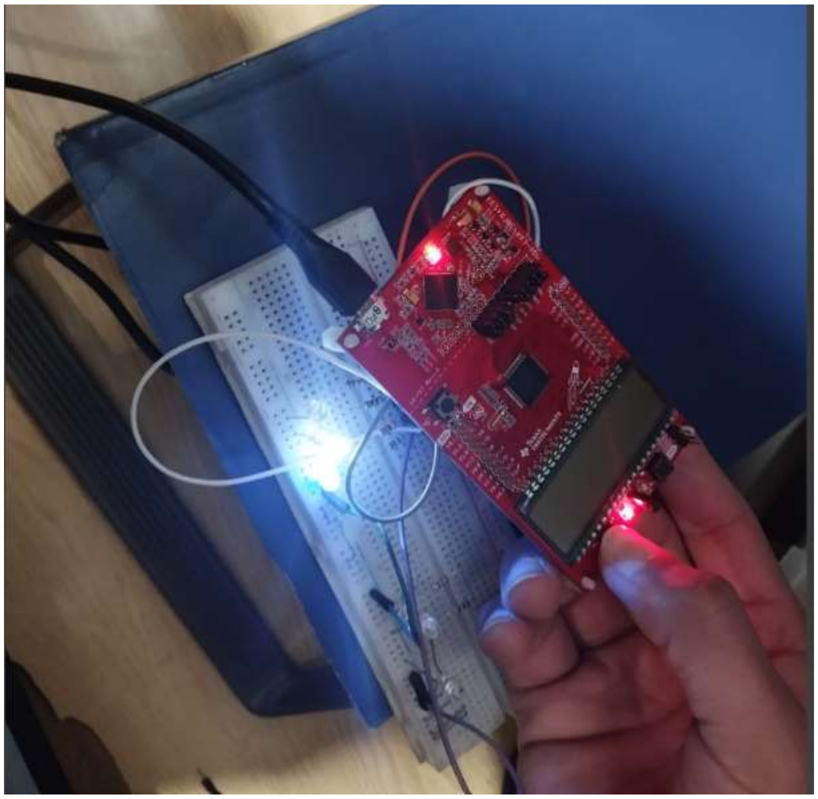
*main.c ☒ cmpf_i.h

```c
#include <msp430.h>
volatile unsigned int i = 0; // Variable to measure press duration
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       // Disable high-impedance mode to enable GPIOs

    P1DIR |= BIT0;              // Set P1.0 as output (LED)
    P1OUT &= ~BIT0;             // Ensure LED is off initially

    P1DIR &= ~BIT2;             // Set P1.2 as input (Button)
    P1REN |= BIT2;              // Enable pull-up/down resistor
    P1OUT |= BIT2;              // Set pull-up resistor

    P1IE  |= BIT2;              // Enable interrupt on P1.2
    P1IES |= BIT2;              // Falling edge trigger (button press)
    P1IFG &= ~BIT2;             // Clear interrupt flag

    __bis_SR_register(GIE);     // Enable global interrupts

    while (1) {
        if (!(P1IN & BIT2)) {   // If button is pressed
            i++;                // Increase counter
            __delay_cycles(100); // Small delay to avoid rapid counting
        } else {
            i = 0;              // Reset counter when button is released
        }
    }
}
```



```c
// Interrupt Service Routine for P1 (Button)
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void) {
    if (i > 10000) {            // Long press detected
        P1OUT |= BIT0;          // Turn LED ON (long press effect)
    } else {                    // Short press detected
        P1OUT ^= BIT0;          // Toggle LED
    }

    P1IFG &= ~BIT2;             // Clear interrupt flag
}
```

Output:



Conclusion:

This MSP430 program controls an LED (P1.0) based on button press duration (P1.2) using an interrupt-based approach and a counter.

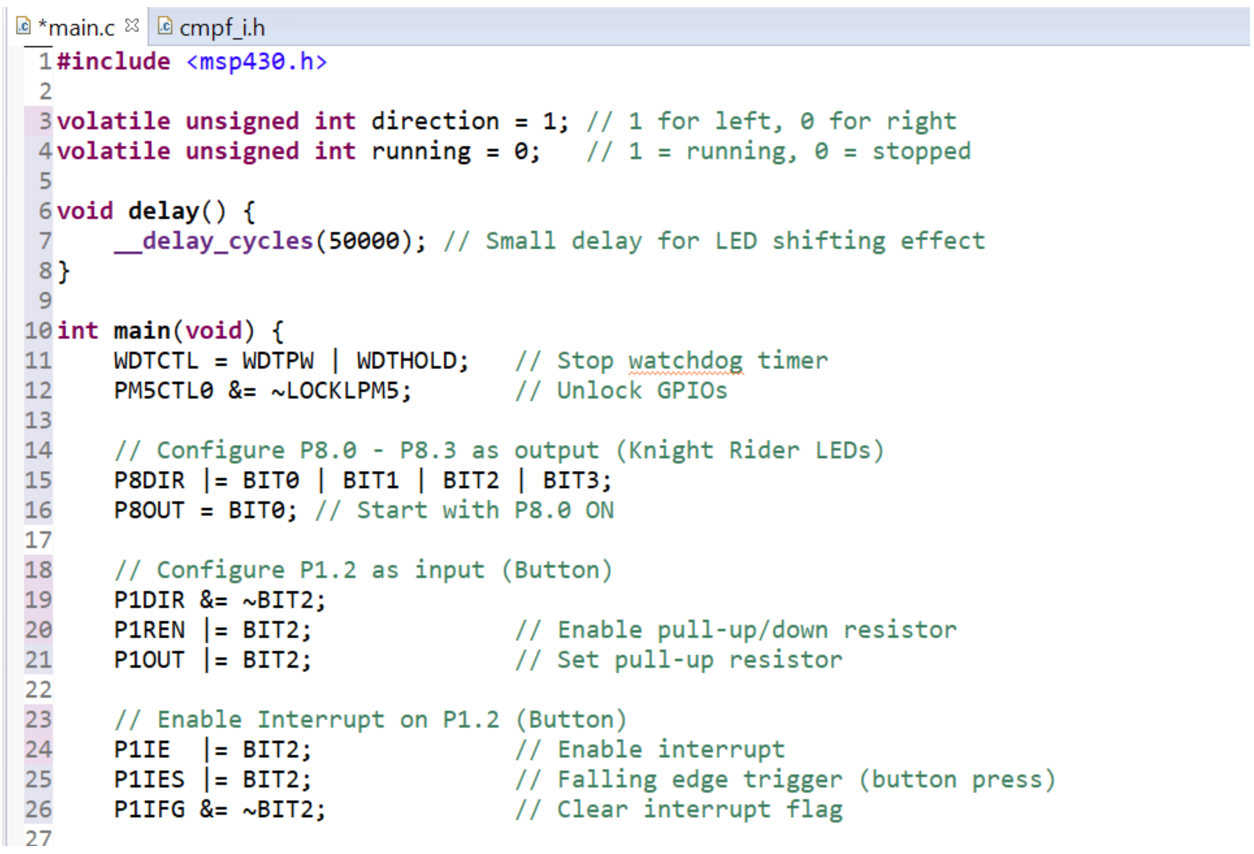
1. Stops watchdog timer and unlocks GPIOs.
2. Configures P1.0 as an output (LED) and P1.2 as an input (Button) with a pull-up resistor.
3. Enables an interrupt on P1.2 (falling edge trigger).
4. In the main loop, a counter (i) tracks button press duration.
5. In the ISR:
    - If pressed for a long time (i > 10000), the LED stays ON.
    - If pressed briefly, the LED toggles.

This allows short and long press detection, making the LED behave differently based on press duration.

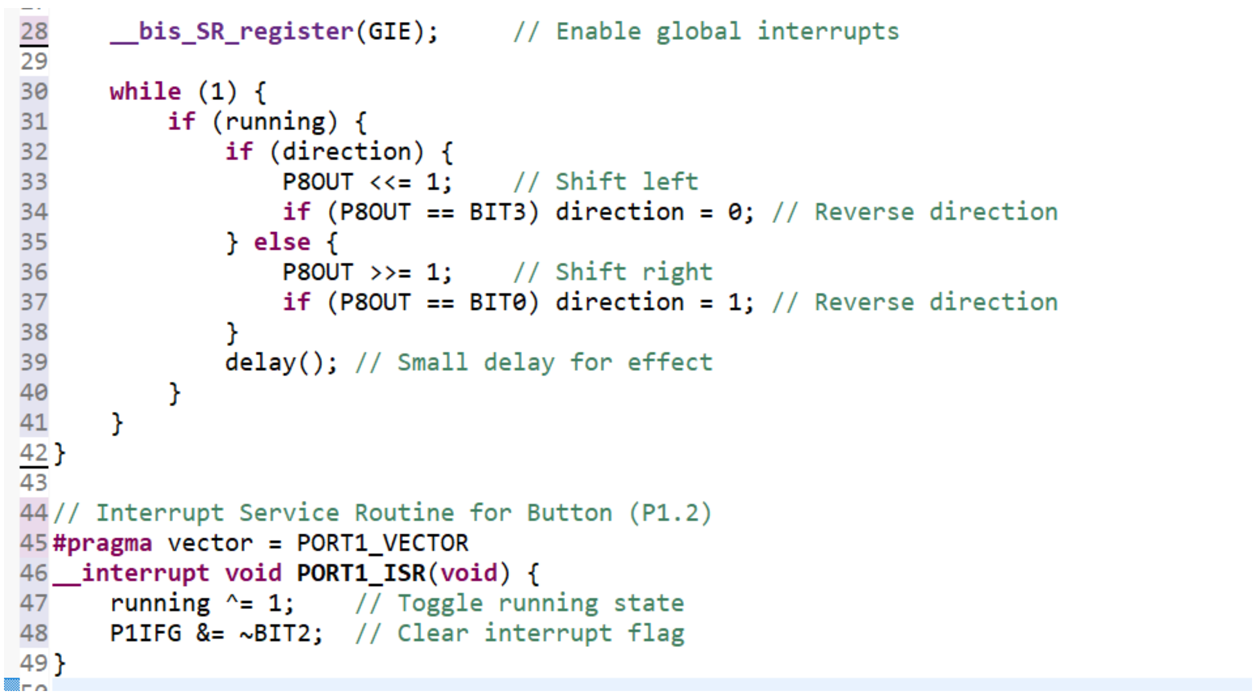
Task 04:

Write code for interrupt based knight rider.

CODE:

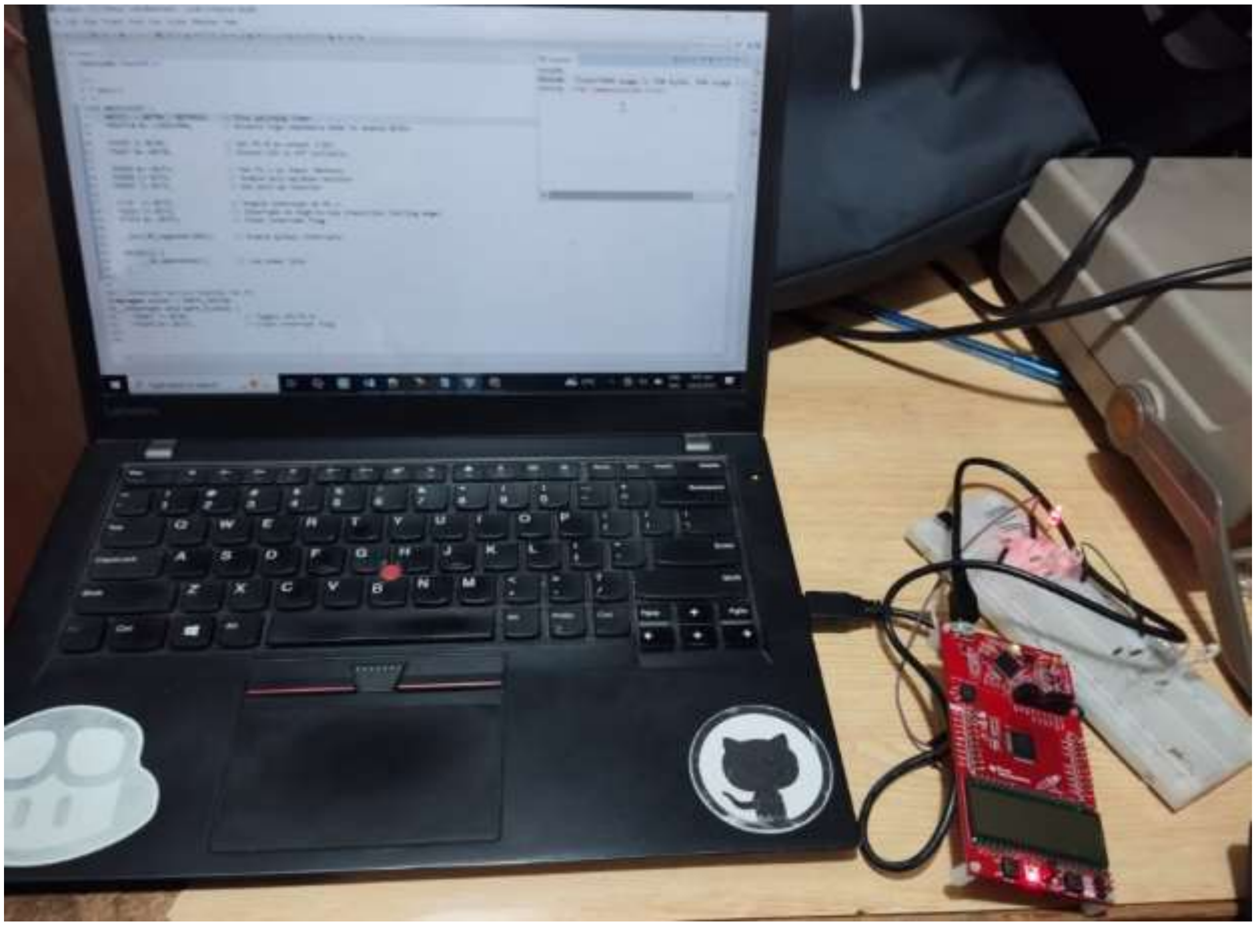


```c
#include <msp430.h>

volatile unsigned int direction = 1; // 1 for left, 0 for right
volatile unsigned int running = 0;   // 1 = running, 0 = stopped

void delay() {
    __delay_cycles(50000); // Small delay for LED shifting effect
}

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;        // Unlock GPIOs

    // Configure P8.0 - P8.3 as output (Knight Rider LEDs)
    P8DIR |= BIT0 | BIT1 | BIT2 | BIT3;
    P8OUT = BIT0; // Start with P8.0 ON

    // Configure P1.2 as input (Button)
    P1DIR &= ~BIT2;
    P1REN |= BIT2;               // Enable pull-up/down resistor
    P1OUT |= BIT2;               // Set pull-up resistor

    // Enable Interrupt on P1.2 (Button)
    P1IE  |= BIT2;               // Enable interrupt
    P1IES |= BIT2;               // Falling edge trigger (button press)
    P1IFG &= ~BIT2;              // Clear interrupt flag

```



```c
    __bis_SR_register(GIE);     // Enable global interrupts

    while (1) {
        if (running) {
            if (direction) {
                P8OUT <<= 1;    // Shift left
                if (P8OUT == BIT3) direction = 0; // Reverse direction
            } else {
                P8OUT >>= 1;    // Shift right
                if (P8OUT == BIT0) direction = 1; // Reverse direction
            }
            delay(); // Small delay for effect
        }
    }
}

// Interrupt Service Routine for Button (P1.2)
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void) {
    running ^= 1;     // Toggle running state
    P1IFG &= ~BIT2;  // Clear interrupt flag
}
```

Output:



Conclusion:

This Knight Rider LED effect code for the MSP430 uses interrupts to toggle LED movement on P8.0 - P8.3 when a button (P1.2) is pressed. The LEDs shift left and right smoothly, reversing direction at the edges. The interrupt ensures the effect can be started or stopped dynamically, making it an efficient and interactive implementation.