# CPE 325: Intro to Embedded Computer System

## Lab03

## LEDs and Switches

**Submitted by**: Gianna Foti

**Date of Experiment**:_____2 September 2024_____

**Report Deadline**:_____9 September 2024_____

**Lab Section:** _____CPE353-02_____

**Demonstration Deadline**: _____9 September 2024_____

# Introduction

In this lab, we were tasked with coding a program to interface SW1 and SW2 as inputs and LED1 and LED2 as outputs. Initially, LED1 is turned on, and LED2 is off at the start of the program. The primary objective was to detect the state of the switches and control the LEDs based on specific behaviors when the switches are held down. Depending on which switch is pressed, the program should make LED1 and LED2 blink at predetermined frequencies. Furthermore, the lab introduces important embedded system concepts such as handling switch debouncing and generating precise delays to achieve the required LED blink rates. There was an additional bonus objective when both switches were pressed that I completed and implemented into program 1.

# Theory Topics

1. Debouncing
    a. When a microcontroller's switches are pressed or released, they can produce a "bouncing" effect, where the signal rapidly toggles between "on" and "off" before stabilizing. This occurs because mechanical contacts in the switch physically bounce for a few milliseconds, causing unintended transitions and resulting in multiple erroneous readings of a single press or release. To prevent this, a software delay is introduced to allow the signal to settle before it's processed. Typically, a delay of 10 to 20 milliseconds is sufficient to filter out these rapid fluctuations and ensure that only a single, reliable input is detected.
2. Software Delay
    a. A software delay is a technique used to introduce a controlled pause in the execution of a program, achieved by instructing the microcontroller to idle for a specific number of clock cycles. By introducing a pause in execution, the system can achieve proper timing synchronization before executing the next instruction. In my lab, I utilized software

delays to create timing for both debouncing the switches and controlling the frequency of the LED blinking.

b. In this lab, software delays were crucial for ensuring that:
    1. **Debouncing**: The program waited for around 20 milliseconds after detecting a switch press to allow the bouncing effect to settle, preventing false inputs.
    2. **Blinking Frequencies**: The delays helped generate the correct blink rates for the LEDs, such as 7 Hz for LED2 when SW1 is pressed and 9 Hz for LED1 when SW2 is pressed. These blink frequencies were calculated based on the required timing and the system clock speed.

# Results & Observation

## Part 1:

### Program Description:

This program controls two LEDs (Red and Green) on the MSP430 board using two input switches, S1 and S2. Initially, the Red LED is turned on, and the Green LED is off. When S1 is pressed, the Red LED turns off, and the Green LED blinks at a frequency of 7 Hz. If S2 is pressed, the Green LED turns on, and the Red LED blinks at 9 Hz. For the bonus, if both switches are pressed simultaneously, the Red and Green LEDs alternate blinking at a frequency of 3 Hz. When neither switch is pressed, the system reverts to its default state, with the Red LED on and the Green LED off. Debouncing is implemented to avoid false triggering caused by mechanical switch bounce, and delays are carefully calculated to achieve the precise blink rates for each LED state. The calculations will be detailed in the next section of this report.

Process:

1. I configured the Red LED and Green LED's as outputs by setting the corresponding but in the direction register.
2. I then did input configuration (S1 and S2).
    a. switch S1 (P2.1) is configured as an input with a pull-up resistor

b. switch S2 (P1.1) is configured as an input with a pull-up resistor:

3. Then I had the program enter an infinite loop to continuously monitor the state of the switches and control the LEDs accordingly.
    a. Switch S1 Pressed:
        i. When Switch S1 is pressed (S1_PRESSED), the program checks that S2 is not pressed.
        ii. A debounce delay is introduced to filter out mechanical noise from the switch: for (i = 2000; i > 0; i--);.
        iii. After confirming that S1 is still pressed:
            1. A 7 Hz delay is applied: __delay_cycles(71428.5714).
            2. The Red LED is turned off: P1OUT &= ~REDLED.
            3. The Green LED is toggled (switched between on and off): P4OUT ^= GREENLED.
    b. Switch S2 Pressed:
        i. If S2 is pressed (S2_PRESSED) and S1 is not pressed, the program applies a debounce delay.
        ii. After debounce, a 9 Hz delay is applied: __delay_cycles(55555.55556).
        iii. The Green LED is turned on: P4OUT |= GREENLED.
        iv. The Red LED is toggled: P1OUT ^= REDLED.
    c. Both S1 and S2 Pressed:
        i. When both switches are pressed, a debounce delay is introduced.
        ii. The LEDs alternate their states at a 3 Hz frequency:
            1. The Red LED is turned on, and the Green LED is turned off, followed by a delay: __delay_cycles(166666.6667).
            2. Then the Red LED is turned off, and the Green LED is turned on: __delay_cycles(166666.6667).
    d. Neither S1 Nor S2 Pressed:
        i. When neither switch is pressed, the program resets the LEDs to the default state:
            1. The Red LED remains on: P1OUT |= REDLED.
            2. The Green LED remains off: P4OUT &= ~GREENLED

4. End

## Flowchart:

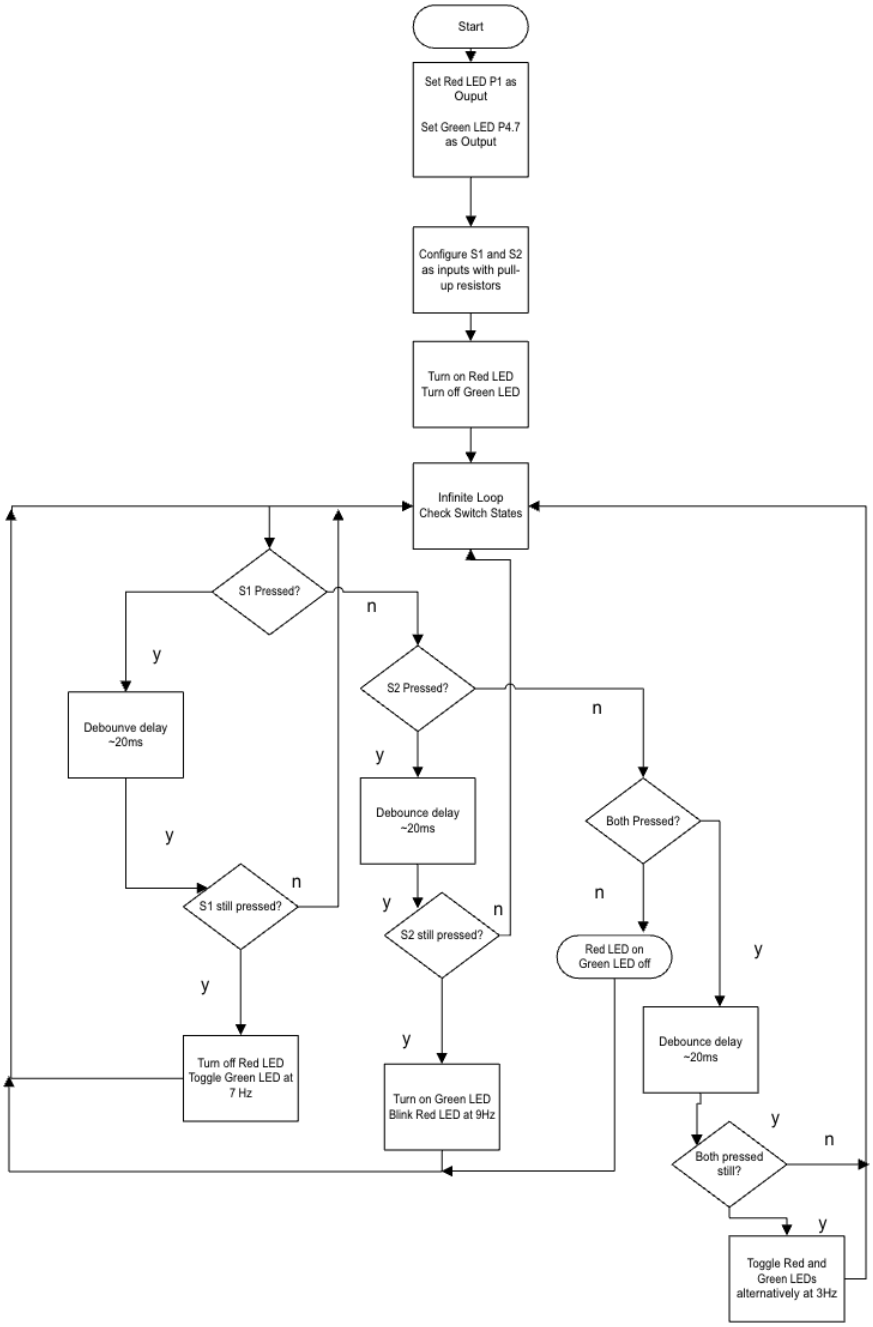


Figure 1: Program 1 and Bonus Flowchart

## Report Questions:

1. **How do you handle debouncing?**
    a. Debouncing is handled by introducing a short delay (~20ms) after detecting a switch press. This prevents false triggering due to mechanical switch bounce. In the code, after detecting a switch press, a for-loop is used to create a small delay (for (i = 2000; i > 0; i--);) before processing the switch's state. This delay allows any switch bounce to settle before the input is registered as a valid press.
2. **How do you create the required delay?**
    a. The required delays to achieve specific LED blink rates are created using the __delay_cycles() function, which pauses the execution for a specified number of clock cycles. The MSP430 operates at 12 MHz, and the number of cycles for each delay is calculated based on the desired frequency (7 Hz, 9 Hz, or 3 Hz). For example, to achieve a 7 Hz blink rate, the code uses __delay_cycles(71428.5714), which corresponds to half the period of a 7 Hz signal. The same approach is used for the 9 Hz and 3 Hz blink rates with their calculated cycle delays.
3. **How does your code handle both the switches?**
    a. The code continuously monitors the state of both switches, S1 and S2, within the main loop. Macros (S1_PRESSED and S2_PRESSED) are used to detect whether each switch is pressed (i.e., when the corresponding pin is low). Depending on the switch combination, the program performs different actions:
        i. If only S1 is pressed, the Red LED turns off, and the Green LED blinks at 7 Hz.
        ii. If only S2 is pressed, the Green LED turns on, and the Red LED blinks at 9 Hz.
        iii. If both S1 and S2 are pressed, the LEDs alternate blinking at 3 Hz.
        iv. If neither switch is pressed, the system returns to its default state (Red LED on, Green LED off).

## Part 2:

### Calculation Description:

In order to find the exact timings, we had to calculate them. In this program, precise delays are required to control the blink rates of the LEDs. The frequency of blinking is achieved using delay loops, and the delay time is calculated based on the clock frequency of the MSP430, which operates at a default frequency of 12 MHz. Timing Delays → 1/sec = Hz

1. **7 Hz for Green LED** (when S1 is pressed):
    a. 1/(x) = 7Hz →1/7 secs = 0.1428571429 secs → (1/7)/2 = 0.0714285714 secs → 71.4285714 ms → 71428.5714 µs
2. **9 Hz for Red LED** (when S2 is pressed):
    a. 1/(x) = 9Hz→1/9 secs = 0.1111111111 secs →(1/9)/2 = 0.0555555556 secs→55.55555556 ms →55555.55556 µs
3. **3 Hz for both LEDs** (when S1 and S2 are pressed):
    a. 1/(x) = 3Hz a→ 1/3 secs = 0.3333333333 secs →(1/3)/2 = 0.1666666667 secs → 166.6666667 ms → 166666.6667 µs

## Conclusion

In conclusion, I successfully configured the MSP430F5529 microcontroller to use switches S1 and S2 as inputs. I initialized the first LED (red) in the "on" state and the second LED (green) in the "off" state. The program handled different input conditions effectively, achieving the following outcomes:

1. **When Switch 1 (S1) was pressed**, LED 2 (green) blinked at 7 Hz.
2. **When Switch 2 (S2) was pressed**, LED 1 (red) blinked at 9 Hz.
3. **When both switches were pressed**, LED 1 and LED 2 alternated blinking at 3 Hz.

4. **When no switches were pressed**, the system returned to its default state, with LED 1 on and LED 2 off.

This lab deepened my understanding of fundamental embedded systems concepts, including input/output interfacing, debouncing, and timing control. All tasks were completed without any issues, and this exercise significantly improved my understanding of embedded systems programming and peripheral control.

# Appendix

Table 1: Program 1 source code with Bonus

```
/*-------------------------------------------------------------------------------
 * File: Lab03_P1.c
 * Description: This program controls two LEDs (Red and Green) on the MSP430 board based on
 * the state of two input switches (S1 and S2). The program performs the following:
 * - If Switch 1 (S1) is pressed, the Red LED turns off and the Green LED toggles at 7 Hz.
 * - If Switch 2 (S2) is pressed, the Green LED turns on and the Red LED toggles at 9 Hz.
 * - If both switches are pressed, both LEDs toggle alternately at 3 Hz.
 * - If neither switch is pressed, the Red LED remains on and the Green LED remains off.
 * Debouncing is implemented to avoid false triggering due to mechanical switch bounce.
 * Board: 5529
 * Input: S1 (P2.1), S2 (P1.1)
 * Output: Red LED (P1.0), Green LED (P4.7)
 * Author: Gianna Foti
 * Date: September 5, 2024
 *-------------------------------------------------------------------------------*/
#include <msp430.h>
// S1 is pressed when P2.1 is low
#define S1_PRESSED (!(P2IN & BIT1))
// S2 is pressed when P1.1 is low
#define S2_PRESSED (!(P1IN & BIT1))
// Mask for P1.0 (Red LED)
#define REDLED BIT0
// Mask for P4.7 (Green LED)
#define GREENLED BIT7
void main(void)
{
   // Stop watchdog timer
   WDTCTL = WDTPW | WDTHOLD;
   // Set Red LED (P1.0) as output
   P1DIR |= REDLED;
   // Set Green LED (P4.7) as output
   P4DIR |= GREENLED;
   // Start with Red LED on and Green LED off
   P1OUT |= REDLED;
   P4OUT &= ~GREENLED;
   // Configure S1 (P2.1) as input with pull-up resistor
   P2DIR &= ~BIT1;
   P2REN |= BIT1;
   P2OUT |= BIT1;
   // Configure S2 (P1.1) as input with pull-up resistor
   P1DIR &= ~BIT1;
   P1REN |= BIT1;
   P1OUT |= BIT1;
   //initializing unsigned int to 0
   unsigned int i = 0;
   // Infinite loop
   while (1)
   {
      // S1 pressed, S2 not pressed
      if (S1_PRESSED && !S2_PRESSED)
      {
```

```
        // Debounce delay ~20ms
        for (i = 2000; i > 0; i--);
        //if s1 is pressed
        if (S1_PRESSED)
        {
            // 7Hz delay
            __delay_cycles(71428.5714);
            // Turn off Red LED
            P1OUT &= ~REDLED;
            // Toggle Green LED
            P4OUT ^= GREENLED;
        }
    }
    // S2 pressed and S1 not pressed
    else if (S2_PRESSED && !S1_PRESSED)
    {
        // Debounce delay ~20ms
        for (i = 2000; i > 0; i--);
        if (S2_PRESSED)
        {
            // 9Hz delay
            __delay_cycles(55555.55556);
            // Turn on green LED
            P4OUT |= GREENLED;
            // Toggle the red LED
            P1OUT ^= REDLED;
        }
    }
    // both S1 and S2 pressed
    else if (S1_PRESSED && S2_PRESSED)
    {
        // Debounce delay ~20ms
        for (i = 2000; i > 0; i--);
        // 3Hz delay
        __delay_cycles(166666.6667);
        // turn on red LED
        P1OUT |= REDLED;
        // turn off green LED
        P4OUT &= ~GREENLED;
        // 3Hz delay
        __delay_cycles(166666.6667);
        // turn off red LED
        P1OUT &= ~REDLED;
        // turn on green LED
        P4OUT |= GREENLED;
    }
    // neither S1 nor S2 pressed, return to default state
    else
    {
        // red LED on
        P1OUT |= REDLED;
        // green LED off
        P4OUT &= ~GREENLED;
    }
```

```
    }
}
```