Embedded Systems Lab 5 Report

Name 1: Suhas Reddy
Roll No 1: 210020030
Name 2: Ansh vivek
Roll No 2: 210020002

Aim

The goal of this experiment was to write a program that continuously monitors a GPIO input pin (SW1) and toggles the RED LED when the switch is pressed. The LED should turn on or off with each press of SW1. Additionally, we were required to use Git for version control to track the changes made to the code files.

Theory

In this experiment, we used GPIO interrupts to detect when the SW1 button was pressed and toggle the RED LED accordingly. SW1 is an active-low switch, meaning it sends a low signal when pressed. We monitored the falling edge of the signal to detect button presses. Using interrupts ensures that the button press is detected immediately without continuously checking the switch's state in the main loop. We also committed changes to the project files using Git.

Main Components

- **GPIO Interrupt:** An interrupt is triggered when SW1 is pressed. The interrupt service routine (ISR) toggles the state of the RED LED and clears the interrupt flag.
- **SW1 (GPIO Input):** SW1 is configured to detect a falling edge, which happens when the button is pressed. The input is connected to a GPIO pin and triggers the interrupt.
- **Startup File:** The GPIO interrupt handler function is implemented in the `tm4c123gh6pm_startup_ccs.c` file. This allows the RED LED to toggle each time the button is pressed.

Code

```
#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"

void GPIOPortF_Handler(void);

// Interrupt handler for GPIO Port F
void GPIOPortF_Handler(void) {
    // Check if interrupt occurred on PF4 (Switch)
    if (GPIO_PORTF_RIS_R & (1 << 4)) {
        GPIO_PORTF_ICR_R |= (1 << 4); // Clear the interrupt
            flag
        // Toggle the LED on PF1
        GPIO_PORTF_DATA_R ^= 0x02;  // Toggle LED on PF1 (
           red LED)
    }
}

int main(void) {
    SYSCTL_RCGC2_R |= 0x00000020; /* enable clock to GPIOF
       */
    GPIO_PORTF_LOCK_R = 0x4C4F434B; /* unlock commit
       register */
    GPIO_PORTF_CR_R = 0x1F; /* make PORTF0 configurable */
    GPIO_PORTF_DEN_R = 0x1E; /* set PORTF pins 4 pin */
    GPIO_PORTF_DIR_R = 0x0E; /* set PORTF4 pin as input user
        switch pin */
    GPIO_PORTF_PUR_R = 0x10; /* PORTF4 is pulled up */

    // Set PF4 to trigger interrupt on falling edge (button
       press 1 to 0 transition)
    GPIO_PORTF_IS_R &= ~(1 << 4);     // Edge-sensitive
    GPIO_PORTF_IBE_R &= ~(1 << 4);    // Single edge
    GPIO_PORTF_IEV_R &= ~(1 << 4);    // Falling edge

    // Clear any prior interrupt on PF4
    GPIO_PORTF_ICR_R |= (1 << 4);

    // Enable interrupt on PF4
    GPIO_PORTF_IM_R |= (1 << 4);

    // Enable the interrupt in NVIC (IRQ30 for GPIO Port F)
    NVIC_EN0_R |= (1 << 30);
```

```
    // Initial state: turn off the LED
    GPIO_PORTF_DATA_R &= 0x02;
    //debouncing
    int temp = 0;
           while(temp < 100000)
               temp ++;
    while(1) {
        // Main loop can remain empty as the interrupt will
            handle the logic
        }
}
```

Results

The program successfully toggled the RED LED when SW1 was pressed. The interrupt was detected, and the LED changed its state as expected. Some minor issues with switch bouncing were observed, leading to occasional missed toggles. Git was used to track changes, making it easier to manage the project.

Conclusion

We successfully implemented a GPIO interrupt to toggle the RED LED with each press of the SW1 switch. The experiment demonstrated how interrupts can be used for efficient event handling in embedded systems. Using Git for file management was also beneficial for version control and code organization.