Assignment 5 Report

EE615 : Embedded Systems Lab

Atharv Gade (210020004)
Yash Meshram (210020053)
22 September 2024

Aim:

Write a program where the main loop monitors the state of a GPIO input pin (SW1) and turns on RED LED whenever SW1 is pressed. With each press, we were supposed to toggle the LED. We were also supposed to use git to manage changes in the files.

Theory:

In this experiment, we were to toggle the RED LED with each press of the SW1 switch. We were also supposed to use git for file management.

To implement the code we used GPIO interrupts. Whenever the switch is pressed, we generate an interrupt which is handled by the GIPO interrupt handler. For the interrupt service routine, we toggle the state of the RED LED and clear the interrupt when the state is changed. This is done so that whenever the SW1 is pressed, we could be able to detect interrupts. As SW1 is an active low switch, we monitor it at the falling edge of the clock.

Code Explanation:

1. GPIO Interrupt:

- An interrupt is generated whenever the SW1 switch is pressed, which is handled by the GPIO interrupt service routine.
- The interrupt service routine toggles the state of the RED LED and clears the interrupt to allow for future detections.

2. GPIO Input (SW1):

- SW1 is an active-low switch, and its state is monitored at the falling edge of the clock.
- The input from SW1 is used to trigger an interrupt whenever a press is detected.

3. Startup CCS file:

- The code for the GPIO_Interrupt_Handler is changed inside the tm4c123gh6pm_startup_ccs.c file to implement the code.

Code:

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

void GPIOPortF_Handler(void);

volatile int x=0;

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 &= 0xD;
    int debounce = 0;
           while(debounce < 1000){debounce ++;} //for debouncing
    while(1) {
        // Main loop can remain empty as the interrupt will handle the
logic
    }
}

// Interrupt handler for GPIO Port F
void GPIOPortF_Handler(void) {
    x++;

    // 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)
    }
}
```

Result:

We observed that the interrupt handler successfully identified the switch interrupt and toggled the red LED as expected. However, the common issue of switch bouncing was noticed, causing occasional inaccuracies in toggling the LED.

We used github to commit changes to our project.

Conclusion:

We demonstrated the ability to use the interrupt handler to toggle the LED on and off with each switch press. The use of git as a file management system was also understood.