

5 September 2024

Lab 5 Report

Group 9

Ashis Pradhan

Gurvir Singh Bhatti

210020003

210020012

Aim:

We were tasked to write a program where the press of a switch is detected by a GPIO interrupt and on each button press the interrupt handler should toggle the state of RED LED. Also we had to submit the git repository of the Lab submission.

Theory:

The switch 1 press is directly linked to the GPIO port F interrupt. We could easily program the interrupt for toggling the red LED on and off. We also had to separately enable the interrupt for GPIO port F in the corresponding registers. Below are the code segments where we made sure that the interrupt is handled appropriately:

The function-

```
49
50 // Interrupt handler for GPIO Port F
51 void GPIOPortF_Handler(void) {
52     // Check if interrupt occurred on PF4 (Switch)
53     if (GPIO_PORTF_RIS_R & (1 << 4)) {
54         // Toggle the LED on PF1
55         GPIO_PORTF_DATA_R ^= (1 << 1); // Toggle LED on PF1 (red LED)
56
57         GPIO_PORTF_ICR_R |= (1 << 4); // Clear the interrupt flag
58     }
59 }
60 }
61
```

The startup interrupt vectors-

```

29 // Forward declaration of the default fault handlers.
30 //
31 //*****
32 void ResetISR(void);
33 static void NmiISR(void);
34 static void FaultISR(void);
35 extern void GPIOPortF_Handler(void);|
36 static void IntDefaultHandler(void);
37
38 //*****
39 //
40 // External declaration for the reset handler that is to be called when the
41 // processor is started
42 //
43 //*****
44 extern void __c_int00(void);

```

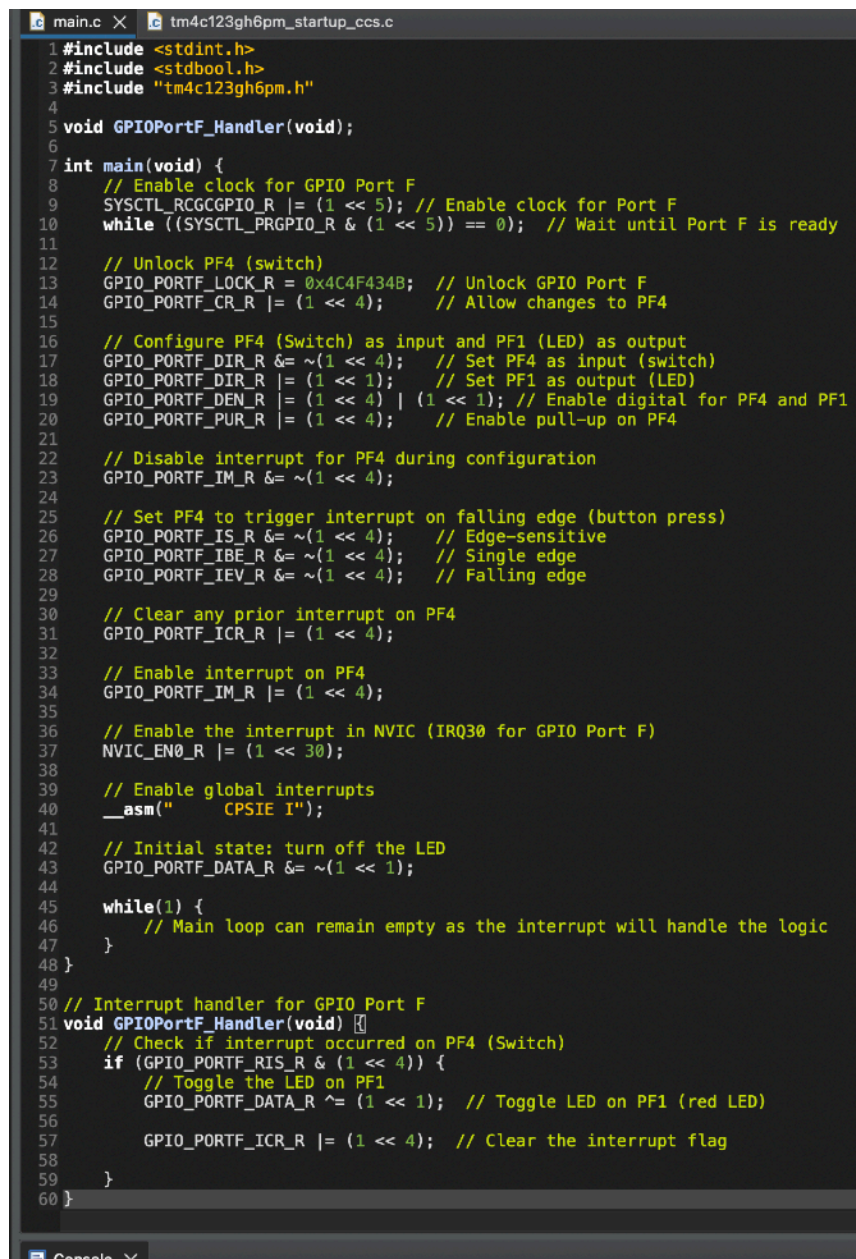
The enabling of interrupts in the corresponding registers-

```

7 int main(void) {
8     // Enable clock for GPIO Port F
9     SYSTCL_RCGCGPIO_R |= (1 << 5); // Enable clock for Port F
10    while ((SYSTCL_PRGPIO_R & (1 << 5)) == 0); // Wait until Port F is ready
11
12    // Unlock PF4 (switch)
13    GPIO_PORTF_LOCK_R = 0x4C4F434B; // Unlock GPIO Port F
14    GPIO_PORTF_CR_R |= (1 << 4); // Allow changes to PF4
15
16    // Configure PF4 (Switch) as input and PF1 (LED) as output
17    GPIO_PORTF_DIR_R &= ~(1 << 4); // Set PF4 as input (switch)
18    GPIO_PORTF_DIR_R |= (1 << 1); // Set PF1 as output (LED)
19    GPIO_PORTF_DEN_R |= (1 << 4) | (1 << 1); // Enable digital for PF4 and PF1
20    GPIO_PORTF_PUR_R |= (1 << 4); // Enable pull-up on PF4
21
22    // Disable interrupt for PF4 during configuration
23    GPIO_PORTF_IM_R &= ~(1 << 4);
24
25    // Set PF4 to trigger interrupt on falling edge (button press)
26    GPIO_PORTF_IS_R &= ~(1 << 4); // Edge-sensitive
27    GPIO_PORTF_IBE_R &= ~(1 << 4); // Single edge
28    GPIO_PORTF_IEV_R &= ~(1 << 4); // Falling edge
29
30    // Clear any prior interrupt on PF4
31    GPIO_PORTF_ICR_R |= (1 << 4);
32
33    // Enable interrupt on PF4
34    GPIO_PORTF_IM_R |= (1 << 4);
35
36    // Enable the interrupt in NVIC (IRQ30 for GPIO Port F)
37    NVIC_EN0_R |= (1 << 30);
38
39    // Enable global interrupts
40    __asm(" CPSIE I");
41
42    // Initial state: turn off the LED
43    GPIO_PORTF_DATA_R &= ~(1 << 1);
44

```

After this we uploaded the files in a GitHub repository for submission.

Code:


```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "tm4c123gh6pm.h"
4
5 void GPIOPortF_Handler(void);
6
7 int main(void) {
8     // Enable clock for GPIO Port F
9     SYSCTL_RCGCGPIO_R |= (1 << 5); // Enable clock for Port F
10    while ((SYSCTL_PRGPIO_R & (1 << 5)) == 0); // Wait until Port F is ready
11
12    // Unlock PF4 (switch)
13    GPIO_PORTF_LOCK_R = 0x4C4F434B; // Unlock GPIO Port F
14    GPIO_PORTF_CR_R |= (1 << 4); // Allow changes to PF4
15
16    // Configure PF4 (Switch) as input and PF1 (LED) as output
17    GPIO_PORTF_DIR_R &= ~(1 << 4); // Set PF4 as input (switch)
18    GPIO_PORTF_DIR_R |= (1 << 1); // Set PF1 as output (LED)
19    GPIO_PORTF_DEN_R |= (1 << 4) | (1 << 1); // Enable digital for PF4 and PF1
20    GPIO_PORTF_PUR_R |= (1 << 4); // Enable pull-up on PF4
21
22    // Disable interrupt for PF4 during configuration
23    GPIO_PORTF_IM_R &= ~(1 << 4);
24
25    // Set PF4 to trigger interrupt on falling edge (button press)
26    GPIO_PORTF_IS_R &= ~(1 << 4); // Edge-sensitive
27    GPIO_PORTF_IBE_R &= ~(1 << 4); // Single edge
28    GPIO_PORTF_IEV_R &= ~(1 << 4); // Falling edge
29
30    // Clear any prior interrupt on PF4
31    GPIO_PORTF_ICR_R |= (1 << 4);
32
33    // Enable interrupt on PF4
34    GPIO_PORTF_IM_R |= (1 << 4);
35
36    // Enable the interrupt in NVIC (IRQ30 for GPIO Port F)
37    NVIC_EN0_R |= (1 << 30);
38
39    // Enable global interrupts
40    __asm(" CPSIE I");
41
42    // Initial state: turn off the LED
43    GPIO_PORTF_DATA_R &= ~(1 << 1);
44
45    while(1) {
46        // Main loop can remain empty as the interrupt will handle the logic
47    }
48 }
49
50 // Interrupt handler for GPIO Port F
51 void GPIOPortF_Handler(void) {
52     // Check if interrupt occurred on PF4 (Switch)
53     if (GPIO_PORTF_RIS_R & (1 << 4)) {
54         // Toggle the LED on PF1
55         GPIO_PORTF_DATA_R ^= (1 << 1); // Toggle LED on PF1 (red LED)
56
57         GPIO_PORTF_ICR_R |= (1 << 4); // Clear the interrupt flag
58     }
59 }
60 }

```

-:Please find the attached startup file for the interrupt handling and better understanding of the implementation:-

Output:

We observed that the interrupt handler was able to correctly determine the switch interrupt and appropriately toggle the red LED. However, we noticed the usual

bouncing issue of the switch, where we notice that sometimes the switch does not accurately toggle the LED.

Conclusion:

We demonstrated that we can utilise the interrupt handler to toggle the LED on and off on a switch press.