Faculty of Engineering & Technology
Electrical & Computer Engineering Department

COMPUTER DESIGN LABORATORY

ENCS4110

Report #1

Experiment #6+7
GPIO Interface & External Interrupts

Prepared by:

Islam Jihad - 1191375

Instructor: Dr. Abualseoud Hanani

Assistant: Eng. Raha Zabadi

Section: 1

Date: 14/May/2022

Abstract

in these two experiments we learn to how to use the chip TM4C123G and how to program it using assembly and C language in addition to use some functionality's and to modify it to work on edge or level or to work on rising edge or falling edge or which method to use in interrupts (Polling based method / Interrupt based method) and to configure the Bush buttons to work on pull up resistance or pull-down resistance.

Table of Contents

Theory

TM4C123G LaunchPad Introduction

The TM4C123G belongs to Texas Instruments' line of high-performance 32-bit ARM cortex M4 microcontrollers with a wide range of peripherals. The Tiva LaunchPad includes a built-in processor with a floating-point unit that runs at up to 80MHz (FPU). In addition, the Cortex-M4F processor supports the capability of tail chaining A layered vector interrupt controller is also included (NVIC). The troubleshooting JTAG and SWD (serial wire debugger) are the programming and debugging interfaces used.

TM4C123G LaunchPad Features

The TM4C123G can be used in a wide range of situations. It has a number of connectivity peripherals that can be used to link a variety of electronic devices, including sensors and actuators like IR sensors and motors. The TM4C123G is a Thumb2 16/32-bit code that uses 26% less memory and is 25% faster than pure 32-bit code. It also features a configurable clocking mechanism and access to a real-time clock via the hibernation module.

GPIO (General Purpose Input/Output)

It features 0-43 input-output pins for common use. On the advanced high-performance bus, each GPIO can be utilized as an external edge or leveltriggered interrupt, commence ADC sampling, and adjust toggling rate to up to CPU clock speed. In the input arrangement, each input pin has a 5V tolerance voltage. A weak pull-up, pull-down, and open drain are available on each GPIO pin.

Each of the aforementioned I/O ports has a number of registers connected with it, each having its own memory map location. The aforementioned addresses are base addresses, which means that the registers connected with that port are contained within them.

Port Name	Lower Address	Upper Address
GPIO port A	0x40004000	0x40004FFF
GPIO port B	0x40005000	0x40005FFF
GPIO port C	0x40006000	0x40006FFF
GPIO port D	0x40007000	0x40007FFF
GPIO port E	0x40024000	0x40024FFF
GPIO port F	0x40025000	0x40025FFF

Table 1 ports

On Board Push Buttons

On the LaunchPad, there are two onboard switches (push buttons) that are coupled internally with the GPIO pins: a toggle switch for power and another push button for resetting or restarting the program execution that is already loaded on the board. As seen in the diagram below:

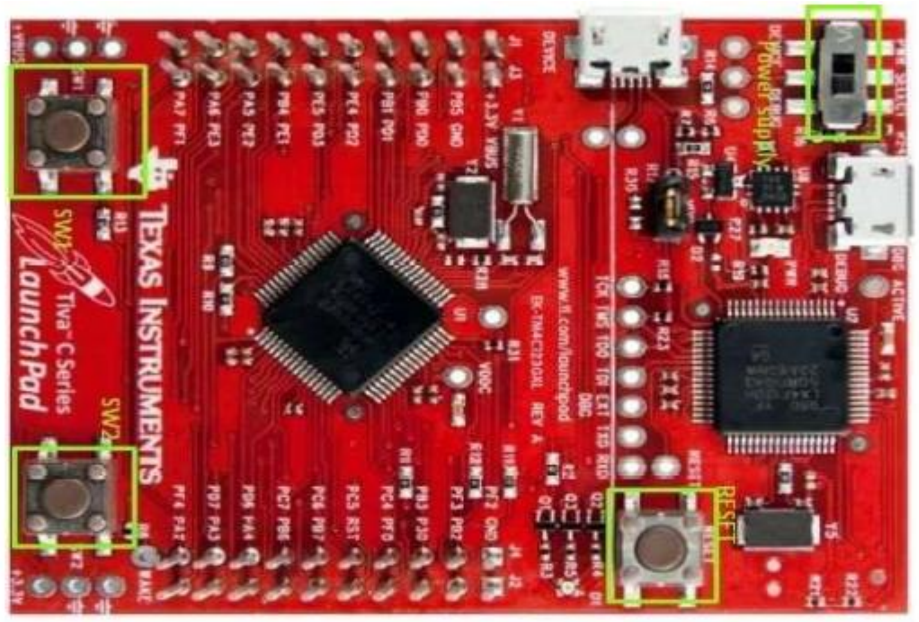


Figure 1 buttons places

The PF0 GPIO pin is linked to the SW1 push-button switch, while the PF4 GPIO pin is connected to the SW2 push-button switch.

Onboard LEDs

One RGB LED is included inside the TIVA LaunchPad. It is connected to the GPIO pins' F port internally, and when enabled, the LED displays the color of the enabled pin. Furthermore, there is a green power LED onboard that, when "on," indicates that the board is switched on. In the diagram below, both LEDs are highlighted.

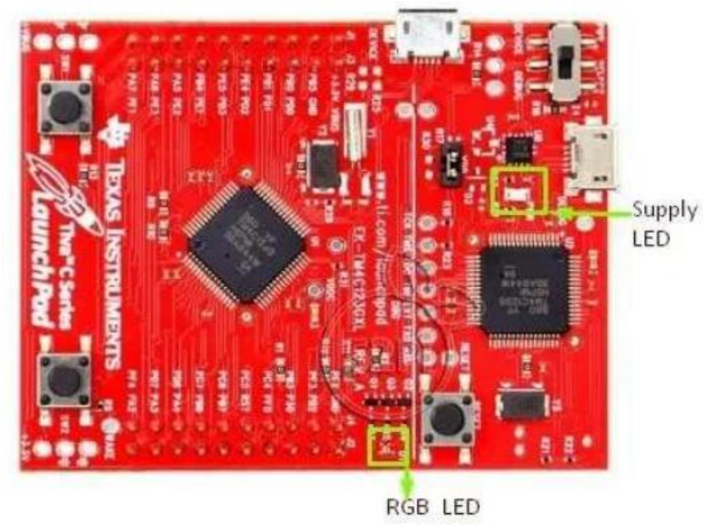


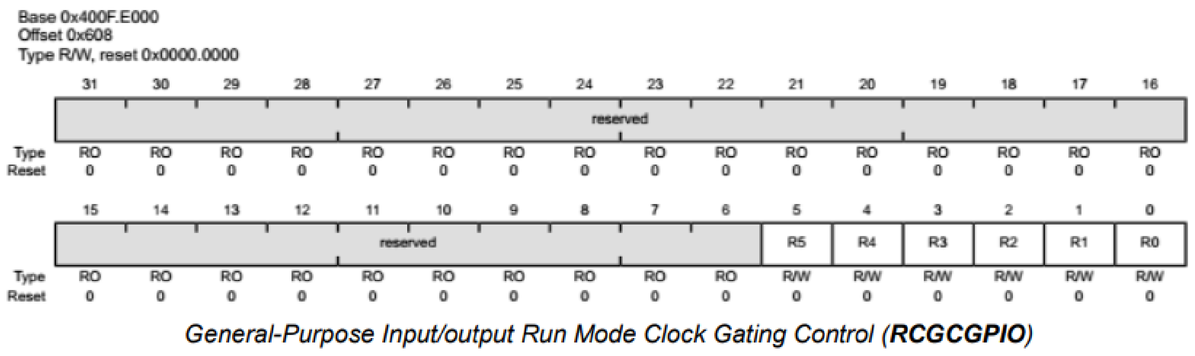
Figure 2 lid places

Activating the Clock

In Run mode, the RCGCGPIO register gives software the ability to activate and disable GPIO modules. When a module is activated, it receives a clock and access to the module. When Access attempts to module registers produce a bus when the clock is deactivated to conserve power. fault. The figure below depicts this registration. The clock for GPIO port F may be activated by asserting the RCGGPIO register's 6th bit We may now set any bit (i.e., make it 1) in a particular register in one of three methods. For instance, We may use the following code to set the 6th bit of the RCGGPIO register:

RCGGPIO = (16); / direct assign: all other pins are set to 0.

Case 2: RCGGPIO |= 0x20; / direct assign: no effect on other pins RCGGPIO |= (16); /binary – Case 3 OR and assign: no effect on other pins



Base 0x400F.E000
Offset 0x608
Type R/W, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

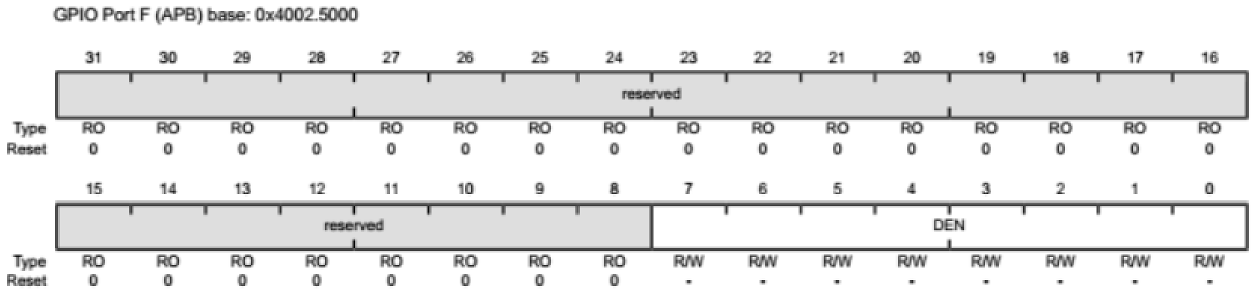
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

*General-Purpose Input/output Run Mode Clock Gating Control (**RCGCGPIO**)*

Figure 3 General-Purpose Input/output Run Mode Clock Gating Control (RCGCGPIO)

Configuring the Pin as Output

After activating the clocks, any relevant pins must be configured. A single pin (PF3) must be setup as an output in this situation. To utilize the pin as a digital input or output, the relevant bit in the GPIODEN register must be set, and then the corresponding pin must be configured as an output by setting a bit in the GPIODIR register.

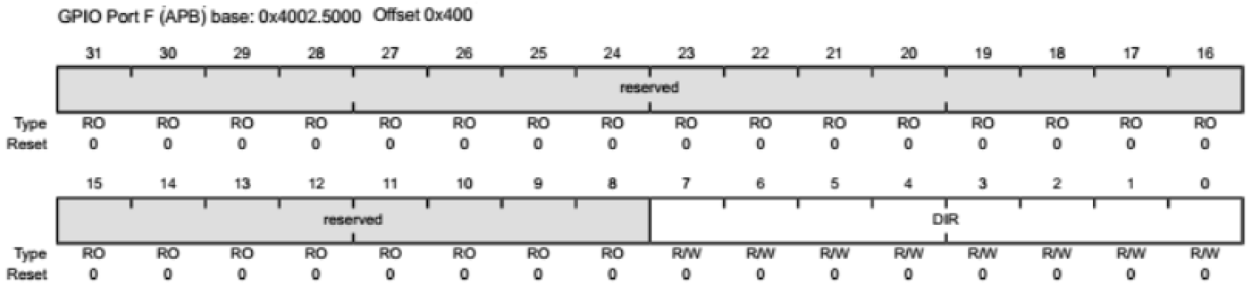


GPIO Port F (APB) base: 0x4002.5000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DEN							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

Figure 4 Configuring the Pin as Output 1



GPIO Port F (APB) base: 0x4002.5000 Offset 0x400

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5 Configuring the Pin as Output 2

How to use Switch as a digital Input?

To input any parameters to digital systems, mechanical switches are widely utilized. Digital inputs may be used to connect the switches to a microcontroller. One of the two techniques for implementing the software program for switch interfacing is as follows.

→Method based on polling

→Method based on interruptions

This course will cover polling-based switch interfacing. Before we go any further, it's crucial to understand the physical behavior of switches, and then we'll talk about switch bouncing, which is one of the most fundamental aspects of that behavior.

Switch Bouncing

Contact bouncing occurs in electrical switches that employ mechanical contacts to shut or open a circuit. Asynchronous switch inputs are not electrically clean. When a hardware switch is pushed, the switch's mechanical contact, which completes the electrical connection, begins to bounce. The program will interpret a single switch push as many presses due to the bouncing effect.

The program will get perplexed as to how many times the switch has been pushed. This issue has both software and hardware solutions. A basic RC filter is used as the hardware solution to this problem. The resistor and capacitor values are selected such that the input is recorded after the bouncing time has ended. The image below depicts the bouncing effect of a switch.

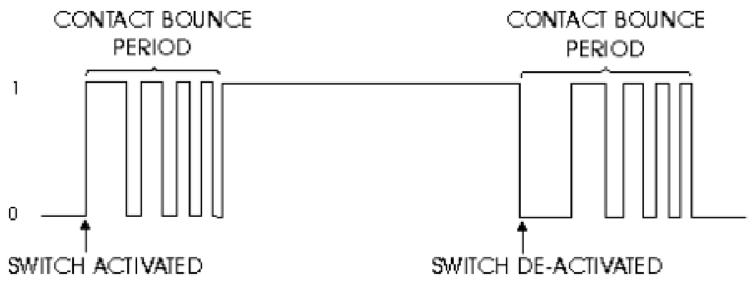


Figure 6 Switch Bouncing

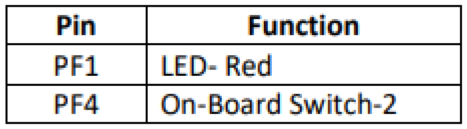
The bouncing effect of the switch is seen in detail in the diagram above. The switch is initially in the off (0) position. When the switch is turned on, it will bounce numerous times, as shown in the diagram, before getting to a constant ON (1) state. When the switch is turned off, the same thing happens. When dealing exclusively with hardware, this issue may not be a concern, but when working with the TIVA LaunchPad's GPIO pins, one push may be read as numerous presses, and the output may not be as expected or necessary.

This is one of the most crucial considerations when dealing with switches. If we want to use the board's built-in switch, we must make the matching pin an input pin.

The pin will read data from the switch and operate the board's built-in LED, which is set as an output, based on the data collected from the switch.

Controlling an LED with a push button using Tiva Launchpad

Let's start with a basic example of utilizing switch-one, which is attached to the PF4 pin of PORTF, to operate an LED connected to the PF1 pin. The LED will light on anytime a user hits the push button linked to the PF0 pin of the TM4C123G6PM microcontroller. Furthermore, the LED shuts off as soon as the user releases the push button.



Pin	Function
PF1	LED- Red
PF4	On-Board Switch-2

GPIO Pins as Digital Input Registers Configuration

Instead of building our own register definition file, we will utilize the register definition header file supplied in Keil, which provides TM4C123G6PM microcontroller general purpose and peripheral register definitions. In the last experiment, we demonstrated how direct pointers dereferencing is used to change the register values of microcontroller peripherals using register memory addresses.

The memory locations of all peripheral registers are listed in the TM4C123G6PM.h header file. As a result, rather of developing our own header file, we may utilize this one. You should, however, be aware of how microcontroller peripheral registers are accessed via pointers and direct memory dereferencing.

The following four steps are the main configuration steps of switch initialization:

1. Enabling the clock

2. Enabling the data register for pin0 or pin4

3. Enabling the direction register as GPIO input register

4. Enabling the PAD for digital operation and enabling the corresponding pull up register

GPIO Interrupts

In embedded systems, general-purpose input-output pins are essential. External components may be easily integrated with microcontrollers via GPIO pins. Microcontrollers utilize input pins to receive data from the outside world, while output pins are used to display data or operate devices such as motors.

Why do we need to use TM4C123 GPIO Interrupts?

We saw an example of controlling an onboard LED of Tiva LaunchPad utilizing onboard switches, such as SW1 (PF0) and SW2 (PF1), in the previous lesson on controlling an LED with a push button using TM4C123 Tiva C LaunchPad (PF4). In that lesson, the TM4C123 microcontroller polls the PF0 and PF4 bits of the PORTF of the TM4C123G microcontroller to verify the condition of the push button. However, one of the major disadvantages of the polling approach is that the microcontroller will have to verify the state of input switches after each consecutive execution of the code or monitor continually (polling method). To synchronize external physical devices with microcontrollers, external or GPIO interrupts are employed.

Instead of continually monitoring the status of input switches, a GPIO pin configured as a digital input may be set to generate an interrupt whenever the state of the switch changes. Interrupt triggers may be activated by falling edges, rising edges, or both falling and rising edges, and they can be level triggered.

In conclusion, using external GPIO interrupts makes the embedded system event driven, responsive, and efficient in terms of microcontroller processing time and resources.

TM4C123GH6PM Microcontroller GPIO Interrupts

PORTA, PORTB, PORTC, PORD, PORTE, and PORTF are six GPIO ports on the TM4C123GH6PM microcontroller.

Each GPIO port's pin may be set as an external interrupt source. We'll learn how to use the PF0 and PF4 pins as an external interrupt source in this lesson. The technique for configuring additional GPIO interrupts, on the other hand, will stay the same.

Two onboard switches, SW1 and SW2, are attached to GPIO pins PF0 and PF4 on the TM4C123 Tiva C LaunchPad. Examples of GPIO interrupt programming will be shown using these input switches.

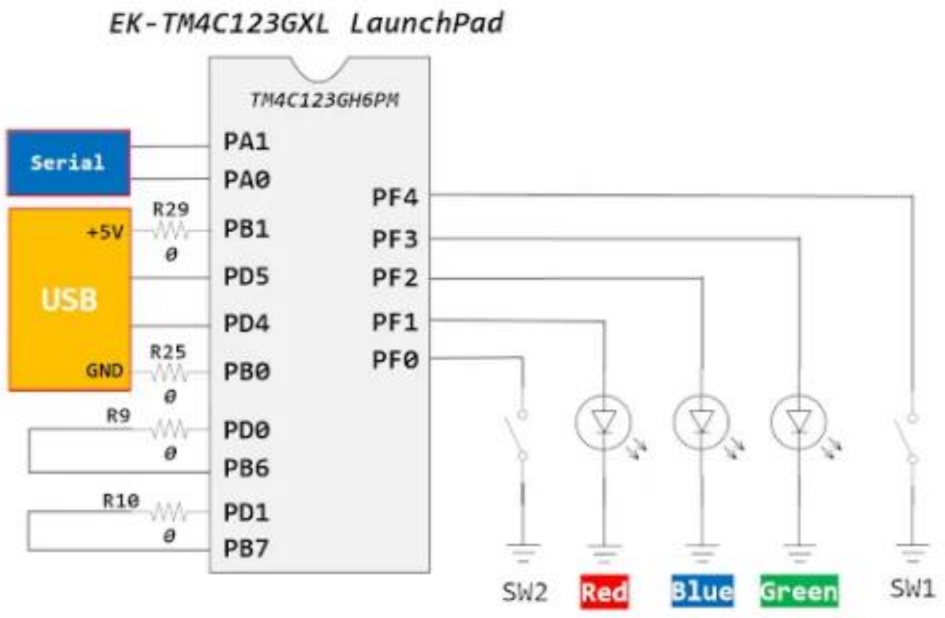


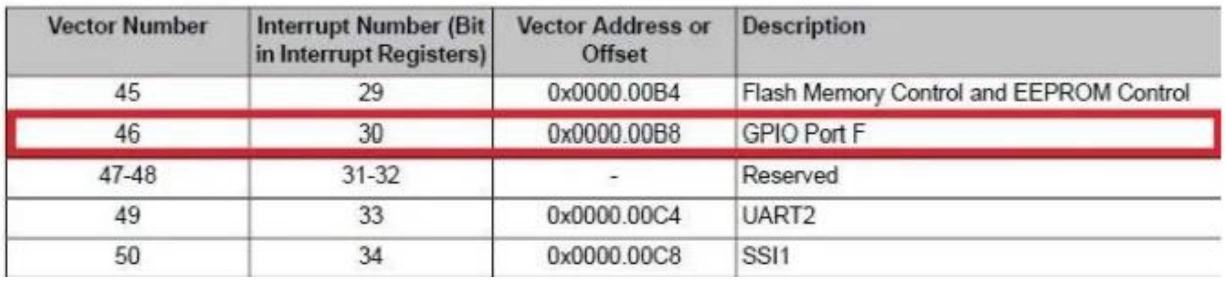
Figure 7 TM4C123GH6PM launch-pad

Find GPO Interrupt Number

The Nested Vectored Interrupt Controller (NVIC) on the TM4C123 microcontroller controls all interrupt requests given by the CPU (exceptions) or peripherals (IRQs). The TM4C123GH6PM microcontroller supports 76 peripheral interrupts (some of which are reserved), each with its own number. This interrupt number is specified in the TM4C123GH6PM startup and header files.

Each exception or peripheral interrupt is assigned a number by NVIC. The unique number allocated to each exception and peripheral interrupt can be found in table 2.9 of the TM4C123GH6PM MCU datasheet.

The interrupt number of GPIO PORTF is 30, as seen in the second column of the diagram below.



Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
45	29	0x0000.00B4	Flash Memory Control and EEPROM Control
46	30	0x0000.00B8	GPIO Port F
47-48	31-32	-	Reserved
49	33	0x0000.00C4	UART2
50	34	0x0000.00C8	SSI1

GPIO Interrupt Edge or Level Triggered Setting (GPIOIS)

external GPIO interrupts of TM4C123G microcontroller can be configured in four modes:

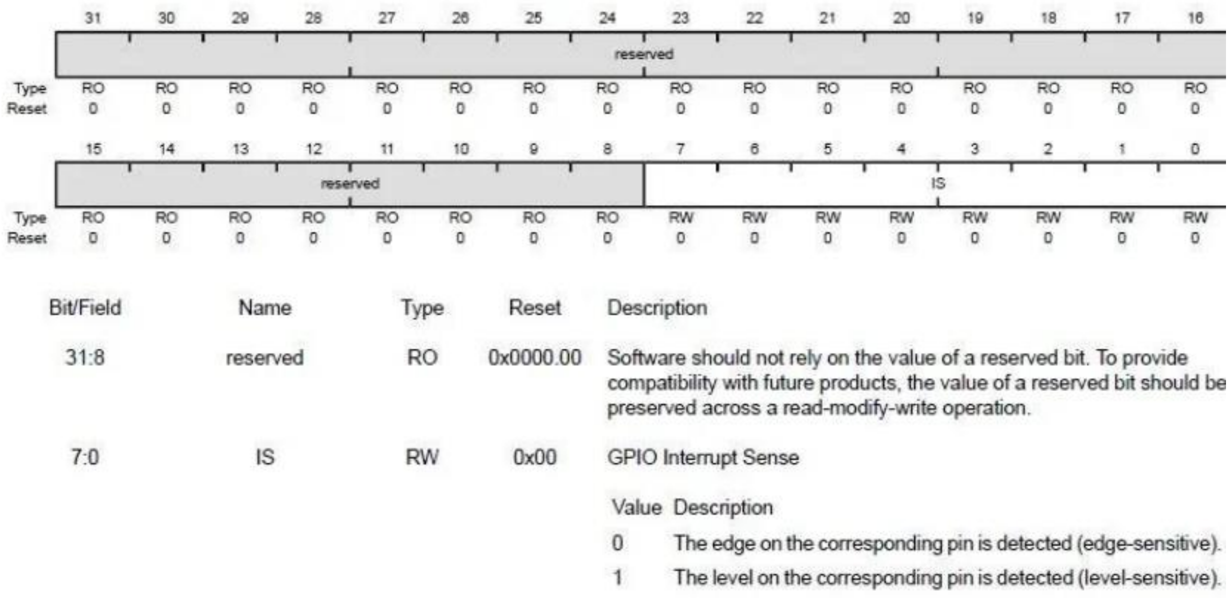
1-Positive edge triggered

2-Negative edge triggered

3-Positive Level (active high)

4-Negative Level (active low)

The GPIO interrupt sense register is used to set whether a pin is level or edge triggered. Setting a bit in the GPIOIS register to detect levels configures the associated pin, whereas removing a bit configures the corresponding pin to detect edges.


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
reserved
Type RO RO RO RO RO RO RO RO RO RO RO RO RO RO RO RO
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
reserved
IS
Type RO RO RO RO RO RO RO RO RW RW RW RW RW RW RW RW
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	IS	RW	0x00	GPIO Interrupt Sense

Value	Description
0	The edge on the corresponding pin is detected (edge-sensitive).
1	The level on the corresponding pin is detected (level-sensitive).

Figure 8 GPO Interrupt Number

Differentiating which GPIO pin causes Interrupt

As previously stated, each GPIO port has just one interrupt service procedure. GPIOF Handler is one of the interrupt handler functions in PORTF (). The same GPIOF Handler() interrupt service code is used for all PORTF pin interrupts. The difficulty now is how to tell which of the PORTF pins causes the GPIOF Handler() function to run.

The TM4C123GH6PM microcontroller includes a GPIO masked interrupt status register, so that's quite simple (GPIOMIS).

This register keeps track of the state of each pin's interrupt. The first eight bits of this register correspond to each GPIO interrupt state PIN0 through PIN7.

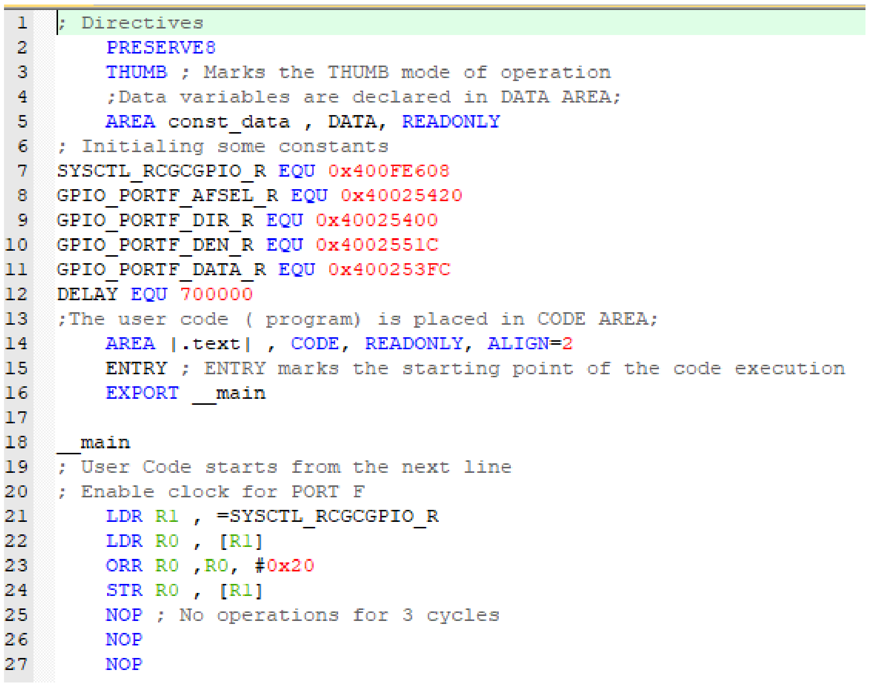
If the PF0 pin causes the PORTF interrupt service routine to be called, the 0th bit of the GPIOMIS register will be 1, and if the PF4 pin causes the interrupt service routine to be called, the 4th bit of the GPIOMIS register will be set automatically. As a result, we can determine which pin generates this specific interrupt by examining the value of each bit of the GPIOMIS register within the PORTF interrupt handler code.

Procedure and Discussion

Exp6 example #1

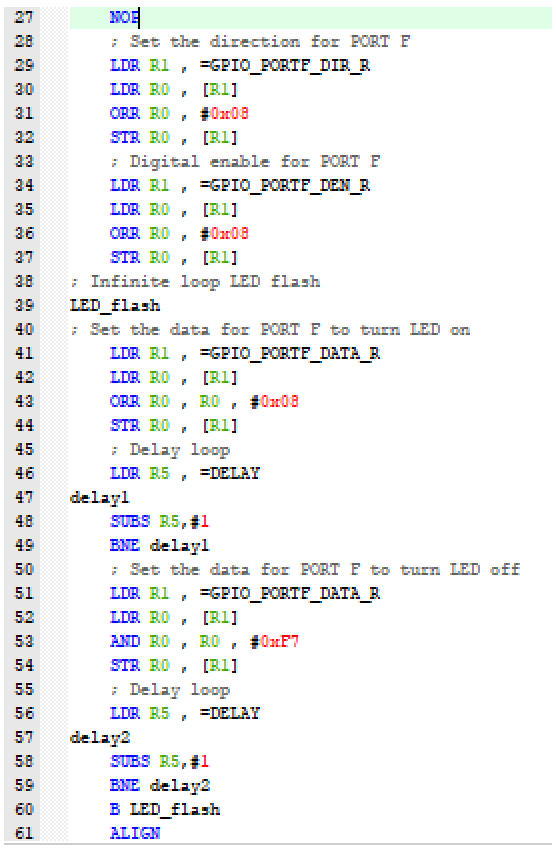
Here I Enabled the GPIO port that is used for the onboard LED.

And enable the GPIO pin for the LED (PF3), Set the direction as output and enable the GPIO pin for digital, it will go in an infinite loop to flash the light till R5 register =1,



```
; Directives
    PRESERVE8
    THUMB ; Marks the THUMB mode of operation
    ;Data variables are declared in DATA AREA;
    AREA const_data , DATA, READONLY
; Initialing some constants
SYSCTL_RCGCGPIO_R EQU 0x400FE608
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_DIR_R EQU 0x40025400
GPIO_PORTF_DEN_R EQU 0x4002551C
GPIO_PORTF_DATA_R EQU 0x400253FC
DELAY EQU 700000
;The user code ( program) is placed in CODE AREA;
    AREA |.text| , CODE, READONLY, ALIGN=2
    ENTRY ; ENTRY marks the starting point of the code execution
    EXPORT __main

__main
; User Code starts from the next line
; Enable clock for PORT F
    LDR R1 , =SYSCTL_RCGCGPIO_R
    LDR R0 , [R1]
    ORR R0 ,R0, #0x20
    STR R0 , [R1]
    NOP ; No operations for 3 cycles
    NOP
    NOP
```

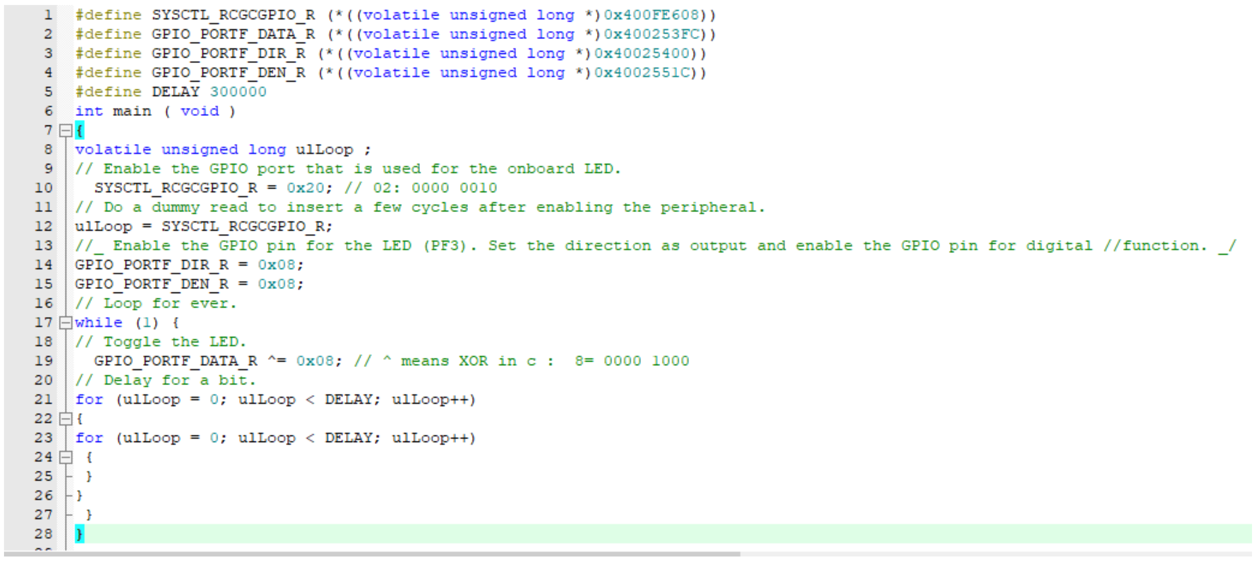
Figure 9 Exp6 example #1 pic1



```
    NOP
    ; Set the direction for PORT F
    LDR R1 , =GPIO_PORTF_DIR_R
    LDR R0 , [R1]
    ORR R0 , #0x08
    STR R0 , [R1]
    ; Digital enable for PORT F
    LDR R1 , =GPIO_PORTF_DEN_R
    LDR R0 , [R1]
    ORR R0 , #0x08
    STR R0 , [R1]
; Infinite loop LED flash
LED_flash
; Set the data for PORT F to turn LED on
    LDR R1 , =GPIO_PORTF_DATA_R
    LDR R0 , [R1]
    ORR R0 , R0 , #0x08
    STR R0 , [R1]
    ; Delay loop
    LDR R5 , =DELAY
delay1
    SUBS R5,#1
    BNE delay1
    ; Set the data for PORT F to turn LED off
    LDR R1 , =GPIO_PORTF_DATA_R
    LDR R0 , [R1]
    AND R0 , R0 , #0xF7
    STR R0 , [R1]
    ; Delay loop
    LDR R5 , =DELAY
delay2
    SUBS R5,#1
    BNE delay2
    B LED_flash
    ALIGN
```

Figure 10 Exp6 example #1 pic2

Exp6 example #2



```
#define SYSCTL_RCGCGPIO_R (*((volatile unsigned long *)0x400FE608))
#define GPIO_PORTF_DATA_R (*((volatile unsigned long *)0x400253FC))
#define GPIO_PORTF_DIR_R (*((volatile unsigned long *)0x40025400))
#define GPIO_PORTF_DEN_R (*((volatile unsigned long *)0x4002551C))
#define DELAY 300000
int main ( void )
{
volatile unsigned long ulLoop ;
// Enable the GPIO port that is used for the onboard LED.
  SYSCTL_RCGCGPIO_R = 0x20; // 02: 0000 0010
// Do a dummy read to insert a few cycles after enabling the peripheral.
ulLoop = SYSCTL_RCGCGPIO_R;
//_ Enable the GPIO pin for the LED (PF3). Set the direction as output and enable the GPIO pin for digital //function. _/
GPIO_PORTF_DIR_R = 0x08;
GPIO_PORTF_DEN_R = 0x08;
// Loop for ever.
while (1) {
// Toggle the LED.
  GPIO_PORTF_DATA_R ^= 0x08; // ^ means XOR in c :  8= 0000 1000
// Delay for a bit.
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
 {
 }
}
 }
}
```

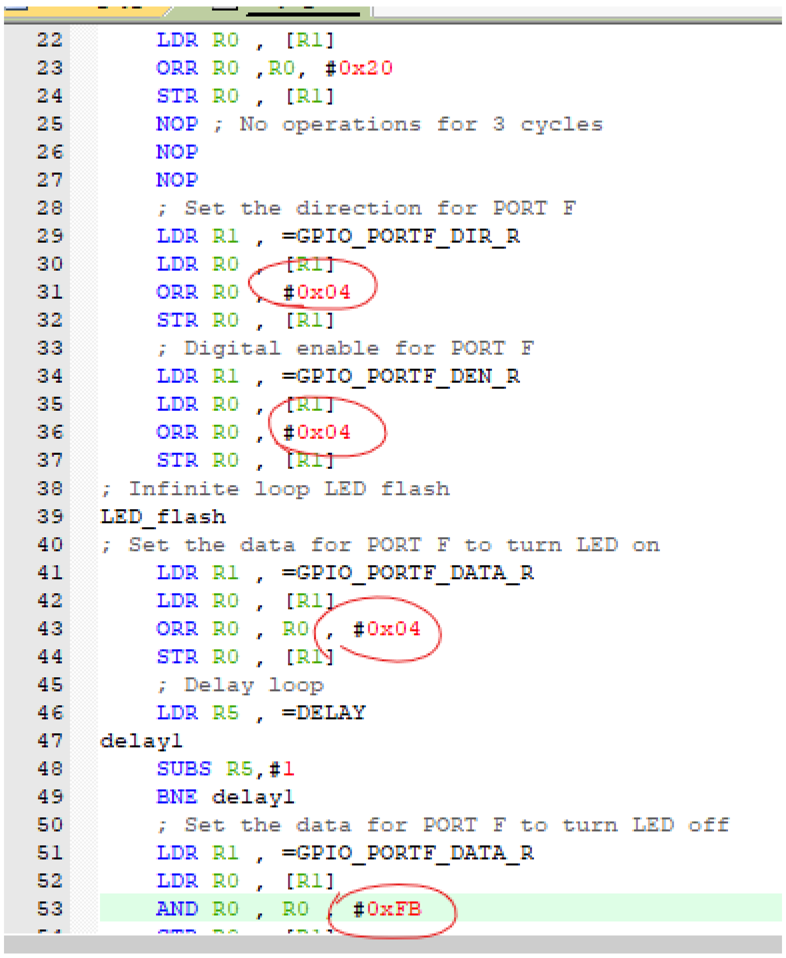
Figure 11 Exp6 example #2 pic1

As seen it will work as the same of the previous assembly code but it's written in C language, the 2 for loops are to make a delay so that the human eye can notice the flashs.

Exp6 code #1

Figure 12 Exp6 code #1 pic1

All the needed action to change the light from red to blue is to change the hexadecimal number from 08 to 04 For the led from enable and etc. and the last line I used FB with and gate to set the blue led to zero so it turn off.

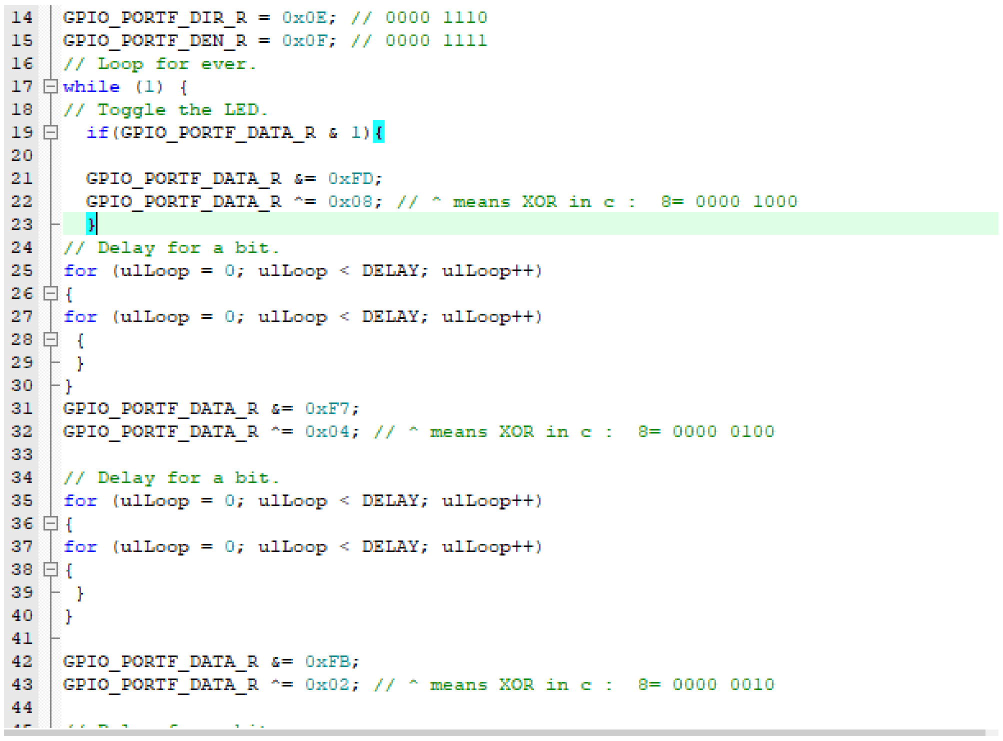


```
    LDR R0 , [R1]
    ORR R0 ,R0, #0x20
    STR R0 , [R1]
    NOP ; No operations for 3 cycles
    NOP
    NOP
    ; Set the direction for PORT F
    LDR R1 , =GPIO_PORTF_DIR_R
    LDR R0 , [R1]
    ORR R0 , #0x04
    STR R0 , [R1]
    ; Digital enable for PORT F
    LDR R1 , =GPIO_PORTF_DEN_R
    LDR R0 , [R1]
    ORR R0 , #0x04
    STR R0 , [R1]
; Infinite loop LED flash
LED_flash
; Set the data for PORT F to turn LED on
    LDR R1 , =GPIO_PORTF_DATA_R
    LDR R0 , [R1]
    ORR R0 , R0 , #0x04
    STR R0 , [R1]
    ; Delay loop
    LDR R5 , =DELAY
delay1
    SUBS R5,#1
    BNE delay1
    ; Set the data for PORT F to turn LED off
    LDR R1 , =GPIO_PORTF_DATA_R
    LDR R0 , [R1]
    AND R0 , R0 , #0xFB
```

And to change the delay all is needed to the DELAY variable only, if the number is greater it will be slower and the opposite is true too.

Exp6 code #2

I enabled the 3 led ports and the push button, then configure the lids as output and the push button as input, then in every led inside the loop I turned the previous led off by using AND gate and turn on the led I wanted but put 1 on the data line

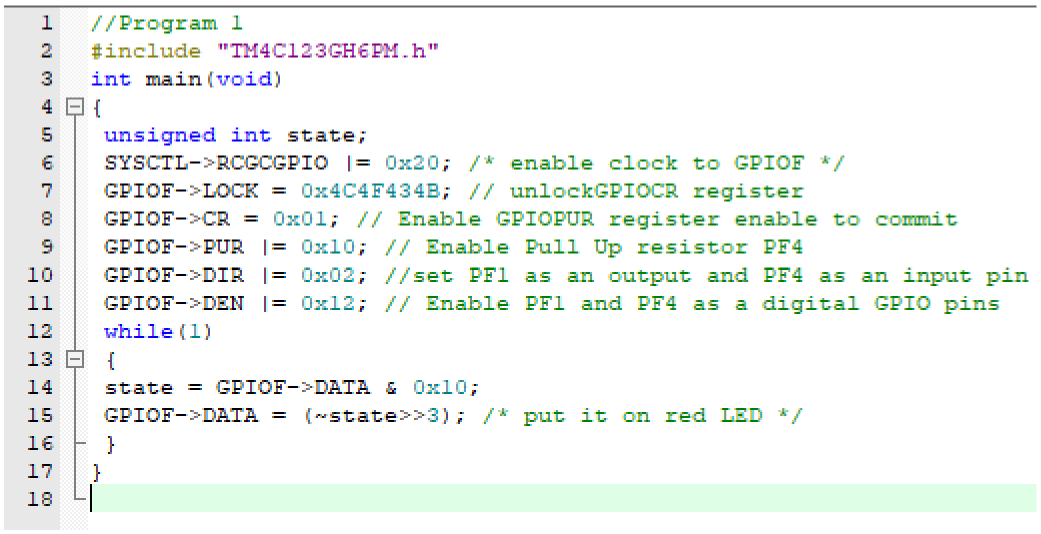


```
GPIO_PORTF_DIR_R = 0x0E; // 0000 1110
GPIO_PORTF_DEN_R = 0x0F; // 0000 1111
// Loop for ever.
while (1) {
// Toggle the LED.
  if(GPIO_PORTF_DATA_R & 1){

  GPIO_PORTF_DATA_R &= 0xFD;
  GPIO_PORTF_DATA_R ^= 0x08; // ^ means XOR in c :   8= 0000 1000
  }
// Delay for a bit.
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
 {
 }
}
GPIO_PORTF_DATA_R &= 0xF7;
GPIO_PORTF_DATA_R ^= 0x04; // ^ means XOR in c :   8= 0000 0100

// Delay for a bit.
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
 }
}

GPIO_PORTF_DATA_R &= 0xFB;
GPIO_PORTF_DATA_R ^= 0x02; // ^ means XOR in c :   8= 0000 0010

```

Figure 13 Exp6 code #2 pic1

Exp6 program #1

This code enable the clock for F port And unlock the P0 to configure it then enable it then look it again, next I enable the pull up resistor for button 4, Then I sit button 4 as in input and button one as output, then I enabled them both, and inside the while loop I made the red lid flash on and off depending on button 4.



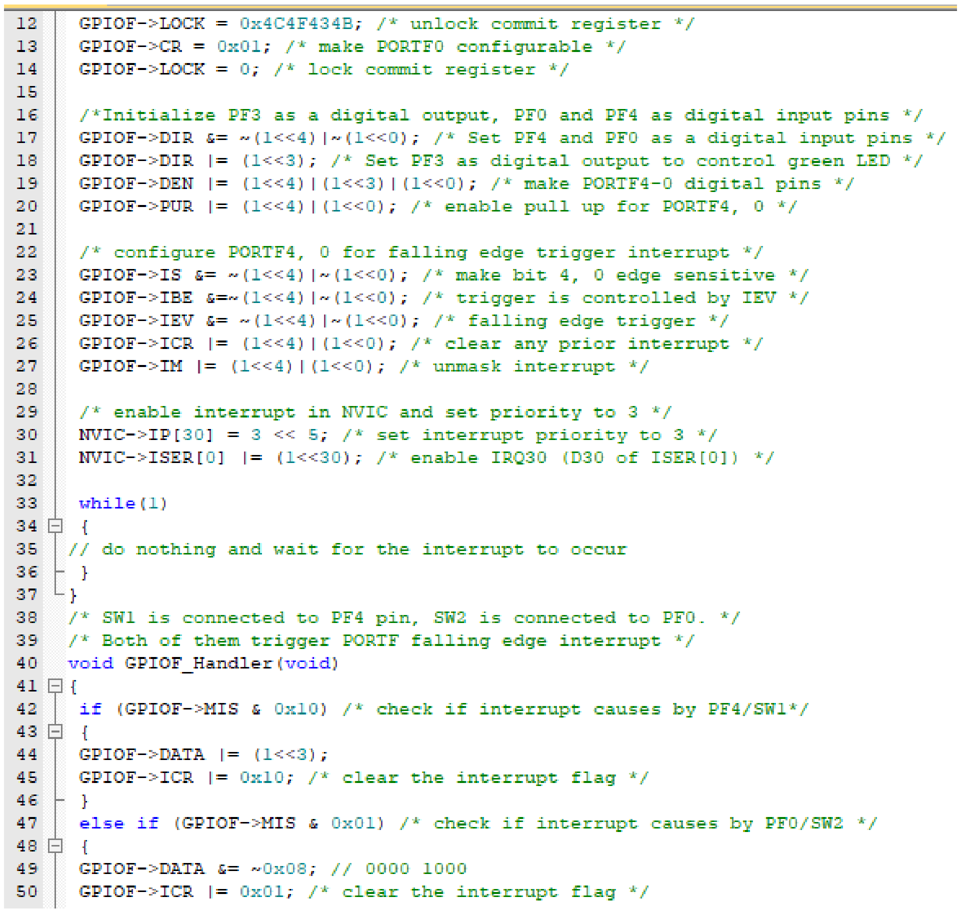
```
//Program 1
#include "TM4C123GH6PM.h"
int main(void)
{
 unsigned int state;
 SYSCTL->RCGCGPIO |= 0x20; /* enable clock to GPIOF */
 GPIOF->LOCK = 0x4C4F434B; // unlockGPIOCR register
 GPIOF->CR = 0x01; // Enable GPIOPUR register enable to commit
 GPIOF->PUR |= 0x10; // Enable Pull Up resistor PF4
 GPIOF->DIR |= 0x02; //set PF1 as an output and PF4 as an input pin
 GPIOF->DEN |= 0x12; // Enable PF1 and PF4 as a digital GPIO pins
 while(1)
 {
 state = GPIOF->DATA & 0x10;
 GPIOF->DATA = (~state>>3); /* put it on red LED */
 }
}
```

Figure 14 Exp6 program #1 pic1

Exp6 program #2

Here next each line there is a comment for it such as put button one and four as input registers and green button as output register, In line theory and theory 1 I sit the priority of interrupts to Level 3 so that it can be executed before the code.

The code here works on interrupt method but the previous code works on bolling based method, the robbed method wouldn't make overhead on the processor so it will execute the code only if an interrupt is called, so in the function online 40 it says if an interrupt came from button one then turn the red lid and the other part of the if statement says if button two is pushed then turn off the red lid. before finishing each statement we have to clear the interrupt flag that's because the next interrupt can affect the code so if the flag isn't cleared then the code won't work perfectly.

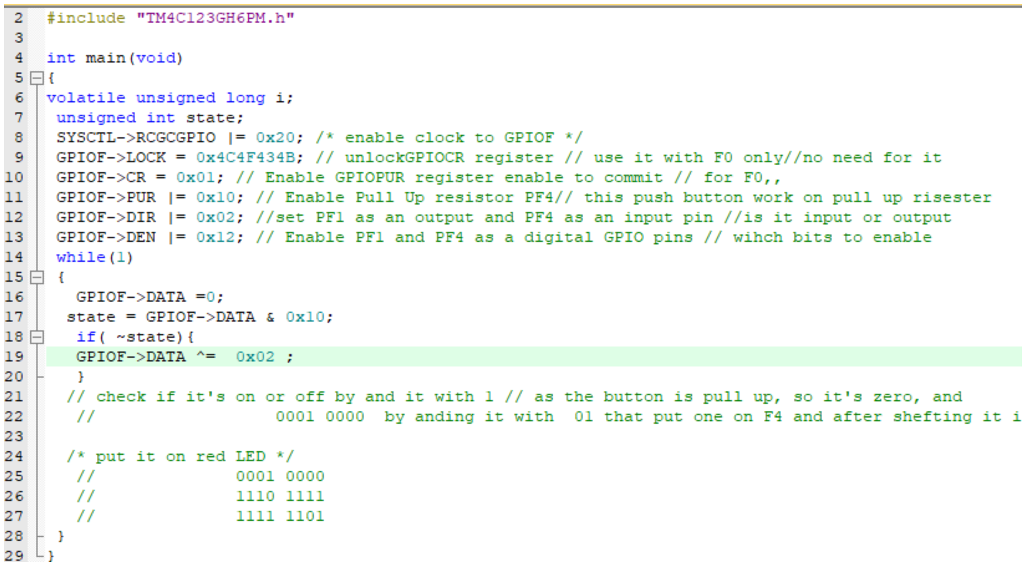


```
12    GPIOF->LOCK = 0x4C4F434B; /* unlock commit register */
13    GPIOF->CR = 0x01; /* make PORTF0 configurable */
14    GPIOF->LOCK = 0; /* lock commit register */
15
16    /*Initialize PF3 as a digital output, PF0 and PF4 as digital input pins */
17    GPIOF->DIR &= ~(1<<4)|~(1<<0); /* Set PF4 and PF0 as a digital input pins */
18    GPIOF->DIR |= (1<<3); /* Set PF3 as digital output to control green LED */
19    GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0); /* make PORTF4-0 digital pins */
20    GPIOF->PUR |= (1<<4)|(1<<0); /* enable pull up for PORTF4, 0 */
21
22    /* configure PORTF4, 0 for falling edge trigger interrupt */
23    GPIOF->IS &= ~(1<<4)|~(1<<0); /* make bit 4, 0 edge sensitive */
24    GPIOF->IBE &=~(1<<4)|~(1<<0); /* trigger is controlled by IEV */
25    GPIOF->IEV &= ~(1<<4)|~(1<<0); /* falling edge trigger */
26    GPIOF->ICR |= (1<<4)|(1<<0); /* clear any prior interrupt */
27    GPIOF->IM |= (1<<4)|(1<<0); /* unmask interrupt */
28
29    /* enable interrupt in NVIC and set priority to 3 */
30    NVIC->IP[30] = 3 << 5; /* set interrupt priority to 3 */
31    NVIC->ISER[0] |= (1<<30); /* enable IRQ30 (D30 of ISER[0]) */
32
33    while(1)
34    {
35   // do nothing and wait for the interrupt to occur
36    }
37   }
38   /* SW1 is connected to PF4 pin, SW2 is connected to PF0. */
39   /* Both of them trigger PORTF falling edge interrupt */
40   void GPIOF_Handler(void)
41   {
42    if (GPIOF->MIS & 0x10) /* check if interrupt causes by PF4/SW1*/
43    {
44    GPIOF->DATA |= (1<<3);
45    GPIOF->ICR |= 0x10; /* clear the interrupt flag */
46    }
47    else if (GPIOF->MIS & 0x01) /* check if interrupt causes by PF0/SW2 */
48    {
49    GPIOF->DATA &= ~0x08; // 0000 1000
50    GPIOF->ICR |= 0x01; /* clear the interrupt flag */
```

Figure 15 Exp6 program #2 pic1

Exp6 program #3

I modified program one to toggle the green lid status every time I push the button As seen all I needed is just to xor the present state with ox02,and put it inside if statement that chick if button two is clicked or not. The program here works on **(bolling based method)**

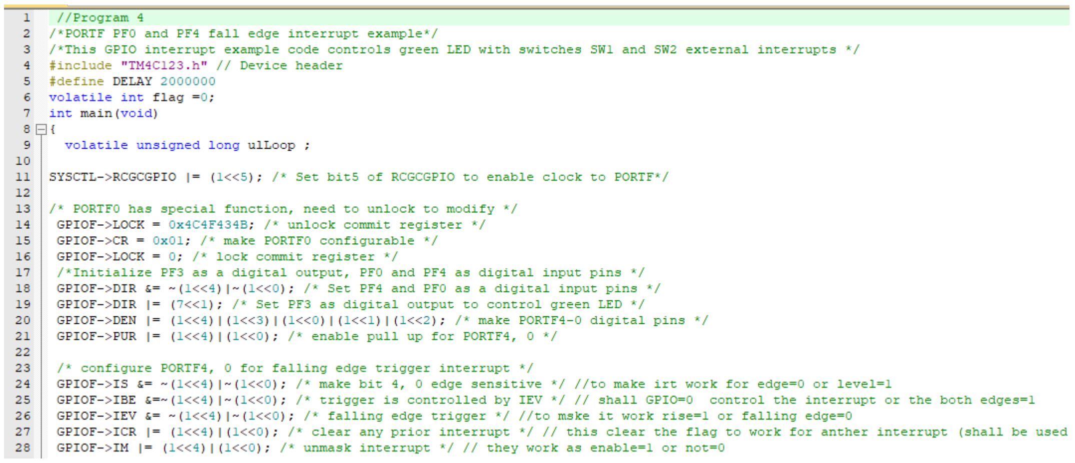


```c
#include "TM4C123GH6PM.h"

int main(void)
{
volatile unsigned long i;
 unsigned int state;
 SYSCTL->RCGCGPIO |= 0x20; /* enable clock to GPIOF */
 GPIOF->LOCK = 0x4C4F434B; // unlockGPIOCR register // use it with F0 only//no need for it
 GPIOF->CR = 0x01; // Enable GPIOPUR register enable to commit // for F0,,
 GPIOF->PUR |= 0x10; // Enable Pull Up resistor PF4// this push button work on pull up risester
 GPIOF->DIR |= 0x02; //set PF1 as an output and PF4 as an input pin //is it input or output
 GPIOF->DEN |= 0x12; // Enable PF1 and PF4 as a digital GPIO pins // wihch bits to enable
 while(1)
 {
   GPIOF->DATA =0;
  state = GPIOF->DATA & 0x10;
   if( ~state){
   GPIOF->DATA ^=  0x02 ;
   }
  // check if it's on or off by and it with 1 // as the button is pull up, so it's zero, and
   //                  0001 0000  by anding it with  01 that put one on F4 and after shefting it i

  /* put it on red LED */
   //              0001 0000
   //              1110 1111
   //              1111 1101
 }
}
```

Figure 16 Exp6 program #3 pic1

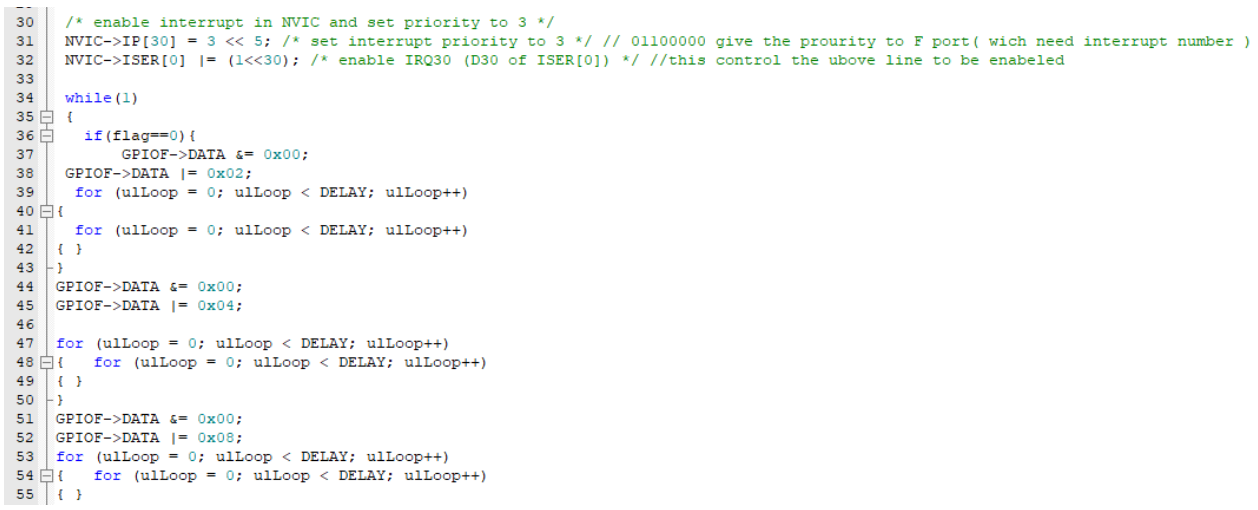
Exp6 program #4

This program will work as the following: win button one is clicked then the three lid colors will flash red green blue, and if button two is clicked then the order of the lid will be conflict two green blue red, all the comments from enable to direction etc are commented next to each line. I wrote in the interrupt function and if statement that change the flag between zero and one and the etc code is in the while loop in the main function.



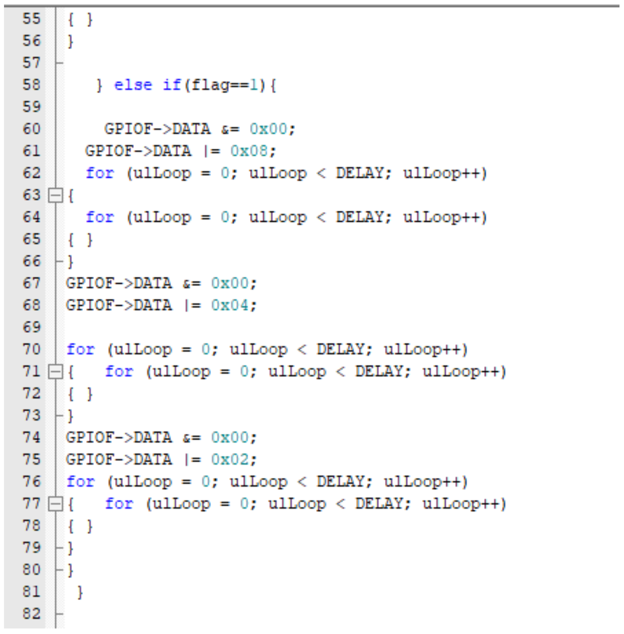
```c
 //Program 4
/*PORTF PF0 and PF4 fall edge interrupt example*/
/*This GPIO interrupt example code controls green LED with switches SW1 and SW2 external interrupts */
#include "TM4C123.h" // Device header
#define DELAY 2000000
volatile int flag =0;
int main(void)
{
  volatile unsigned long ulLoop ;

SYSCTL->RCGCGPIO |= (1<<5); /* Set bit5 of RCGCGPIO to enable clock to PORTF*/

/* PORTF0 has special function, need to unlock to modify */
 GPIOF->LOCK = 0x4C4F434B; /* unlock commit register */
 GPIOF->CR = 0x01; /* make PORTF0 configurable */
 GPIOF->LOCK = 0; /* lock commit register */
 /*Initialize PF3 as a digital output, PF0 and PF4 as digital input pins */
 GPIOF->DIR &= ~(1<<4)|~(1<<0); /* Set PF4 and PF0 as a digital input pins */
 GPIOF->DIR |= (7<<1); /* Set PF3 as digital output to control green LED */
 GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0)|(1<<1)|(1<<2); /* make PORTF4-0 digital pins */
 GPIOF->PUR |= (1<<4)|(1<<0); /* enable pull up for PORTF4, 0 */

 /* configure PORTF4, 0 for falling edge trigger interrupt */
 GPIOF->IS &= ~(1<<4)|~(1<<0); /* make bit 4, 0 edge sensitive */ //to make irt work for edge=0 or level=1
 GPIOF->IBE &=~(1<<4)|~(1<<0); /* trigger is controlled by IEV */ // shall GPIO=0  control the interrupt or the both edges=1
 GPIOF->IEV &= ~(1<<4)|~(1<<0); /* falling edge trigger */ //to mske it work rise=1 or falling edge=0
 GPIOF->ICR |= (1<<4)|(1<<0); /* clear any prior interrupt */ // this clear the flag to work for anther interrupt (shall be used
 GPIOF->IM |= (1<<4)|(1<<0); /* unmask interrupt */ // they work as enable=1 or not=0
```

Figure 17 Exp6 program #4 pic1



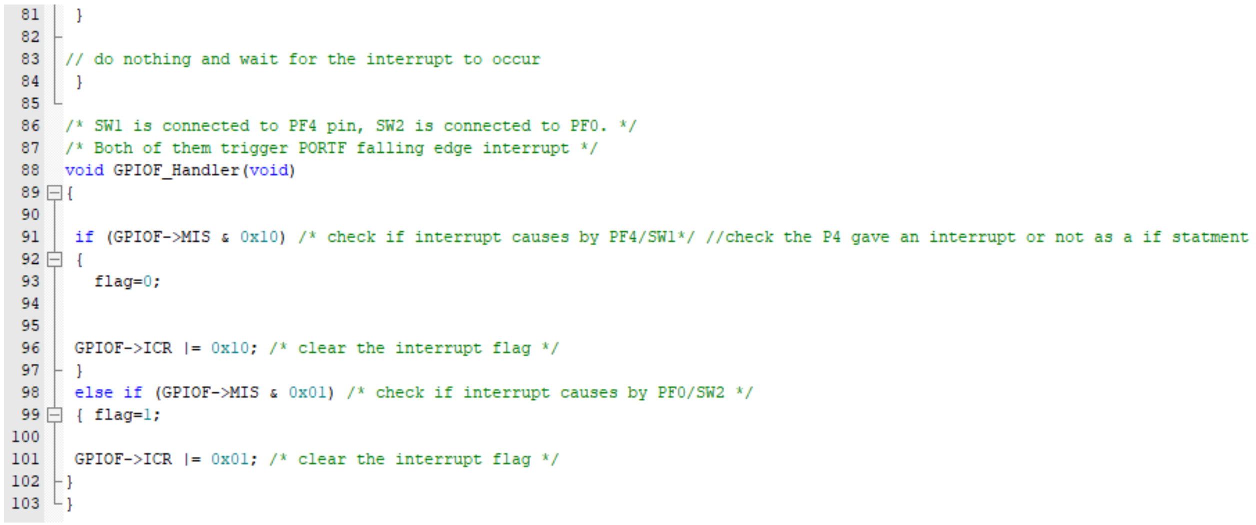
```
/* enable interrupt in NVIC and set priority to 3 */
NVIC->IP[30] = 3 << 5; /* set interrupt priority to 3 */ // 01100000 give the prourity to F port( wich need interrupt number )
NVIC->ISER[0] |= (1<<30); /* enable IRQ30 (D30 of ISER[0]) */ //this control the ubove line to be enabeled

while(1)
{
  if(flag==0){
      GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x02;
 for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
 for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
}
GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x04;

for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{   for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
}
GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x08;
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{   for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
```

Figure 18 Exp6 program #4 pic2



```
{ }
}

   } else if(flag==1){

     GPIOF->DATA &= 0x00;
   GPIOF->DATA |= 0x08;
   for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
   for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
}
GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x04;

for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{    for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
}
GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x02;
for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{    for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }
}
}
 }
```

Figure 19 Exp6 program #4 pic3

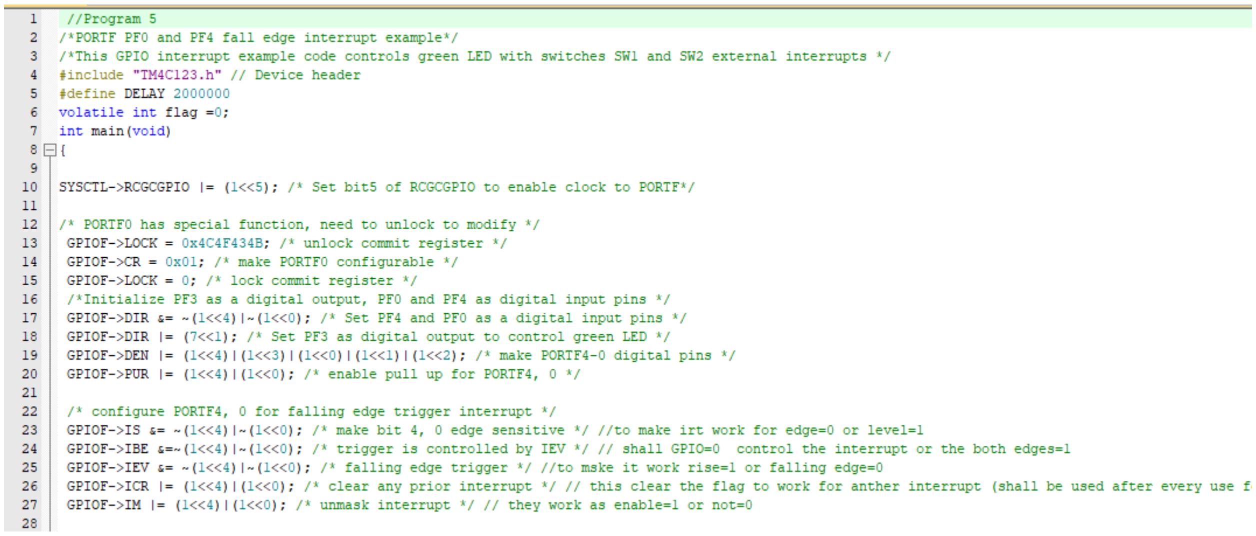


```
 }

// do nothing and wait for the interrupt to occur
 }

/* SW1 is connected to PF4 pin, SW2 is connected to PF0. */
/* Both of them trigger PORTF falling edge interrupt */
void GPIOF_Handler(void)
{

 if (GPIOF->MIS & 0x10) /* check if interrupt causes by PF4/SW1*/ //check the P4 gave an interrupt or not as a if statment
 {
   flag=0;


 GPIOF->ICR |= 0x10; /* clear the interrupt flag */
 }
 else if (GPIOF->MIS & 0x01) /* check if interrupt causes by PF0/SW2 */
 { flag=1;

 GPIOF->ICR |= 0x01; /* clear the interrupt flag */
}
}
```

Figure 20 Exp6 program #4 pic4

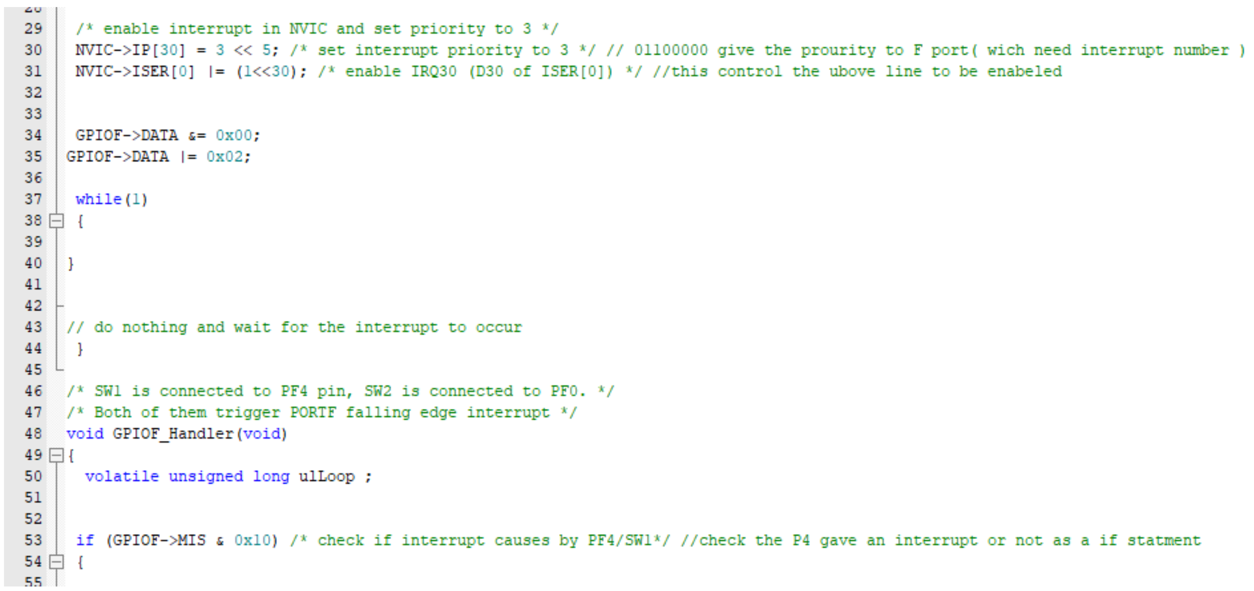
Exp6 program #5

This program is almost like the previous one the idea here is to turn on the red lid at the beginning and when the user click the right button the green lid will flash and when the user click the left button the blue lid will flash. Line 34 &35 are to turn on the red light at the beginning, the code here work on interrupt method so the while loop is empty and if one button make an interruption the functionality in the function will work which turn off the lights then turn on the light depending on which button is clicked. The nested loops are to make some delay so the flashes won't affect on each other.



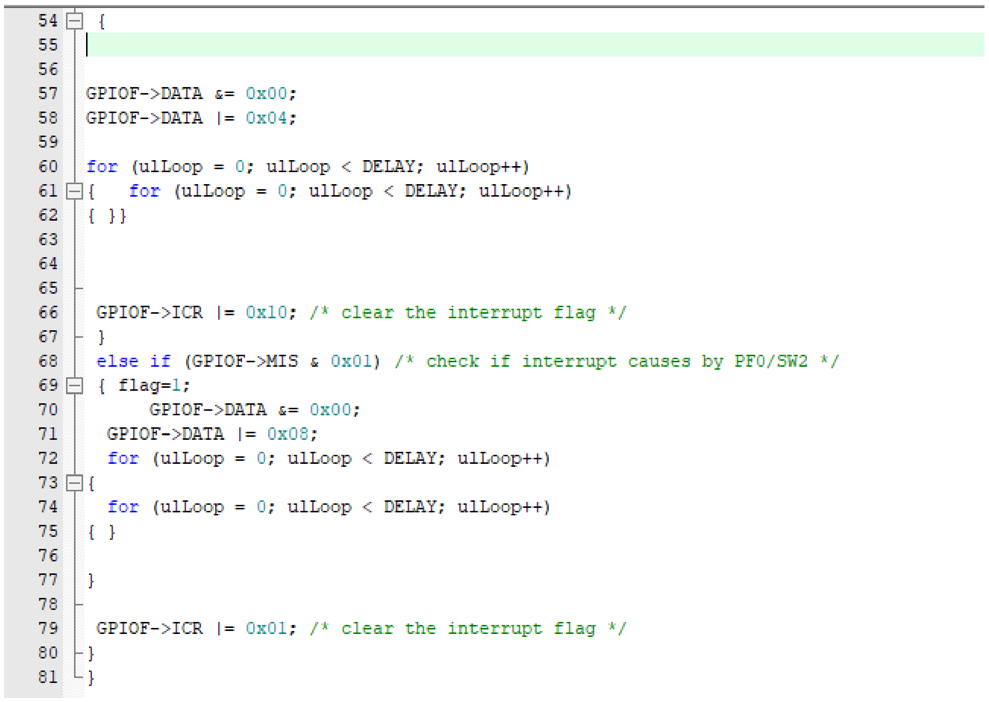
```
1   //Program 5
2  /*PORTF PF0 and PF4 fall edge interrupt example*/
3  /*This GPIO interrupt example code controls green LED with switches SW1 and SW2 external interrupts */
4  #include "TM4C123.h" // Device header
5  #define DELAY 2000000
6  volatile int flag =0;
7  int main(void)
8  {
9
10 SYSCTL->RCGCGPIO |= (1<<5); /* Set bit5 of RCGCGPIO to enable clock to PORTF*/
11
12 /* PORTF0 has special function, need to unlock to modify */
13  GPIOF->LOCK = 0x4C4F434B; /* unlock commit register */
14  GPIOF->CR = 0x01; /* make PORTF0 configurable */
15  GPIOF->LOCK = 0; /* lock commit register */
16  /*Initialize PF3 as a digital output, PF0 and PF4 as digital input pins */
17  GPIOF->DIR &= ~(1<<4)|~(1<<0); /* Set PF4 and PF0 as a digital input pins */
18  GPIOF->DIR |= (7<<1); /* Set PF3 as digital output to control green LED */
19  GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0)|(1<<1)|(1<<2); /* make PORTF4-0 digital pins */
20  GPIOF->PUR |= (1<<4)|(1<<0); /* enable pull up for PORTF4, 0 */
21
22  /* configure PORTF4, 0 for falling edge trigger interrupt */
23  GPIOF->IS &= ~(1<<4)|~(1<<0); /* make bit 4, 0 edge sensitive */ //to make irt work for edge=0 or level=1
24  GPIOF->IBE &=~(1<<4)|~(1<<0); /* trigger is controlled by IEV */ // shall GPIO=0  control the interrupt or the both edges=1
25  GPIOF->IEV &= ~(1<<4)|~(1<<0); /* falling edge trigger */ //to mske it work rise=1 or falling edge=0
26  GPIOF->ICR |= (1<<4)|(1<<0); /* clear any prior interrupt */ // this clear the flag to work for anther interrupt (shall be used after every use f
27  GPIOF->IM |= (1<<4)|(1<<0); /* unmask interrupt */ // they work as enable=1 or not=0
28
```

Figure 21 Exp6 program #5 pic1



```
 /* enable interrupt in NVIC and set priority to 3 */
 NVIC->IP[30] = 3 << 5; /* set interrupt priority to 3 */ // 01100000 give the prourity to F port( wich need interrupt number )
 NVIC->ISER[0] |= (1<<30); /* enable IRQ30 (D30 of ISER[0]) */ //this control the ubove line to be enabeled


 GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x02;

 while(1)
 {

}


// do nothing and wait for the interrupt to occur
 }

/* SW1 is connected to PF4 pin, SW2 is connected to PF0. */
/* Both of them trigger PORTF falling edge interrupt */
void GPIOF_Handler(void)
{
  volatile unsigned long ulLoop ;


 if (GPIOF->MIS & 0x10) /* check if interrupt causes by PF4/SW1*/ //check the P4 gave an interrupt or not as a if statment
 {
```

Figure 22 Exp6 program #5 pic2



```
 {


GPIOF->DATA &= 0x00;
GPIOF->DATA |= 0x04;

for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{    for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }}



 GPIOF->ICR |= 0x10; /* clear the interrupt flag */
 }
 else if (GPIOF->MIS & 0x01) /* check if interrupt causes by PF0/SW2 */
 { flag=1;
      GPIOF->DATA &= 0x00;
  GPIOF->DATA |= 0x08;
  for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{
  for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
{ }

}

 GPIOF->ICR |= 0x01; /* clear the interrupt flag */
}
}
```

Figure 23 Exp6 program #5 pic3

Conclusion

In this experiment we learn how to use the chip and its functions from output and inputs using assembly or C language and how to compile and run the code, we also learn how to use the two methods polling or interrupts, and we noticed that the interrupt method is more efficient than polling method, and how important to clear the flag after using the interrupt.

References

The lab manual