 **Electrical and Computer Engineering**

**Computer Design Lab – ENCS4110**

**GPIO (General Purpose Input/Output) External Interrupts**

Student Name: Maisam Alaa

Student ID: 1200650

Instructor: Ayman Hroub

Section: 4

Date: 24\11\2022

# **Abstract:**

The main objective of this experiment is to introduce students to the two main methods of dealing with digital inputs (e.g., switches or push-buttons); polling and interrupt techniques. Also to learn how to read the state of the onboard switches with the two methods and control the onboard LED’s.

# **Contents**

[**Theory:** 1](#_Toc125224054)

[**Introduction to GPIO peripheral** 1](#_Toc125224055)

[**Features of the GPIO peripheral** 1](#_Toc125224056)

[**GPIO Switch** 2](#_Toc125224057)

[**What is Switch Bouncing?** 2](#_Toc125224058)

[**GPIO Interrupts** 3](#_Toc125224059)

[**Interrupt Trigger Point** 3](#_Toc125224060)

[**TM4C123GH6PM Microcontroller GPIO Interrupts** 3](#_Toc125224061)

[**Nested Vectored Interrupt Controller (NVIC)** 4](#_Toc125224062)

[**Enabling and Disabling an Interrupt** 4](#_Toc125224063)

[**GPIO Interrupt Event Register (GPIOIEV)** 5](#_Toc125224064)

[**GPIO Interrupt Both Edge (GPIOIBE)** 6](#_Toc125224065)

[**Examples** 7](#_Toc125224066)

[Exercise 1 7](#_Toc125224067)

[Exercise 2 7](#_Toc125224068)

[**Procedure:** 10](#_Toc125224069)

[**Lab Work 1** 10](#_Toc125224070)

[**Lab Work 2** 11](#_Toc125224071)

[**Lab Work 3** 15](#_Toc125224072)

[**To Do** 17](#_Toc125224073)

[**Conclusion:** 20](#_Toc125224074)

[**References:** 21](#_Toc125224075)

**Figure**

[Figure 1 GPIO peripheral 1](#_Toc125223958)

[Figure 2 switch bouncing 2](#_Toc125223959)

[Figure 3 TM4C123 Tiva C LaunchPad 4](#_Toc125223960)

[Figure 4 Interrupt enabling with all 3 levels 5](#_Toc125223961)

[Figure 5 GPIO Interrupt Mask (GPIOIM) 5](#_Toc125223962)

[Figure 6 GPIO Interrupt Event Register (GPIOIEV) 5](#_Toc125223963)

[Figure 7 GPIO Interrupt Both Edge (GPIOIBE) 6](#_Toc125223964)

[Figure 8 Example 1 code 7](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223965)

[Figure 9 Example 2 code 9](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223966)

[Figure 10 lab work 1 code 10](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223967)

[Figure 11 lab work 1 testing on board 11](#_Toc125223968)

[Figure 12 lab work 2 code 14](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223969)

[Figure 13 lab work 2 testing on board 14](#_Toc125223970)

[Figure 14 lab work 3 code 16](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223971)

[Figure 15 Todo code 18](file:///C:\Users\maisam\Desktop\report2\report.docx#_Toc125223972)

[Figure 16 Todo testing on board 19](#_Toc125223973)

Tables

[Table 1 Using GPIOIM and GPIOIEV Registers. 6](#_Toc125223994)

[Table 2 Using GPIO interrupt Both Edge registers 6](#_Toc125223995)

# **Theory:**

## **Introduction to GPIO peripheral**

The GPIO peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the output pin. When configured as an input, you can detect the state of the input by reading the state of an internal register.[2]

## **Features of the GPIO peripheral**

The GPIO peripheral consists of the following features.

• Output set/clear functionality through separate data set and clear registers allows multiple software processes to control GPIO signals without critical section protection.

• Set/clear functionality through writing to a single output data register is also supported.

• Separate input/output registers – Output register can be read to reflect output drive status. – Input register can be read to reflect pin status.

• Some GPIO signals can be used as interrupt sources with configurable edge detection.[1]

**SET\_RIS\_TRIG**

**Register**

**CLR\_RIS\_TRIG**

**Register**

**SET\_FAL\_TRIG**

**Register**

**CLR\_FAL\_TRIG**

**Register**

**INSTAT**

**Register**

**INDATA**

**Register**

**Edge**

**Detection**

**Logic**

**Synchronizing Flip-Flops**

**Interrupt**

**toARM**

**CPU**

**EDMA**

**Event**

**CLR\_DATA**

**Register**

**SET\_DATA**

**Register**

**DIR**

**Register**

**Direction**

**Logic**

**OUTDATA**

**Register**

**GPIO**

**Signal**

Figure 1 GPIO peripheral

## **GPIO Switch**

The GPIO switch platform allows you to use any pin on your node as a switch. You can for example hook up a relay to a GPIO pin and use it through this platform.

The software program for switch interfacing can be implemented using one of the following two methods:

• Polling based method

• Interrupt based method

The main difference between interrupt and polling is that, in the case of an interrupt, the system informs the CPU that it needs attention, while talking about polling, the CPU constantly inspects the status of the system to find whether it needs attention.[3]

## **What is Switch Bouncing?**

When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply. But they don’t connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made. The same thing happens while releasing the button. This results the **false triggering or multiple triggering** like the button is pressed multiple times. Its like falling a bouncing ball from a height and it keeps bouncing on the surface, until it comes at rest. As shown in figure 2.

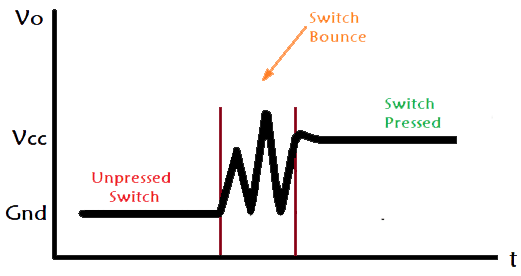


Figure 2 switch bouncing

Simply, we can say that the **switch bouncing** is the non-ideal behavior of any switch which generates **multiple transitions of a single input**. Switch bouncing is not a major problem when we deal with the power circuits, but it cause problems while we are dealing with the [logic or digital circuits](https://circuitdigest.com/digital-electronics). Hence, to remove the bouncing from the circuit **Switch Debouncing Circuit** is used.[6]

## **GPIO Interrupts**

Some general-purpose I/O (GPIO) controller devices can configure their GPIO pins to function as interrupt request inputs. These interrupt request inputs are driven by peripheral devices that are physically connected to the GPIO pins. The drivers for these GPIO controllers can enable, disable, mask, unmask, and clear interrupt requests on individual GPIO pins.

Support for GPIO interrupts is optional. The GPIO framework extension (GpioClx) does not require GPIO controllers to support GPIO interrupts. external or GPIO interrupts are used to synchronize external physical devices with microcontrollers.[3]

## **Interrupt Trigger Point**

When an input pin is connected to an external device to be used for interrupt, we have 5 choices for trigger point. They are:

* low-level trigger (active Low level),
* high-level trigger (active High level),
* rising-edge trigger (positive-edge going from Low to High),
* falling-Edge trigger (negative-edge going from High to Low),
* Both edge (rising and falling) trigger.

In summary, the use of external GPIO interrupts makes the embedded system event driven, responsive and they make use of microcontroller’s processing time and resources efficiently.[1]

## **TM4C123GH6PM Microcontroller GPIO Interrupts**

TM4C123 Tiva C LaunchPad has two onboard switches SW1 and SW2 which are connected with PF0 and PF4 GPIO pins. These input switches will be used to demonstrate GPIO interrupt programming examples. As shown in figure 3.[4]

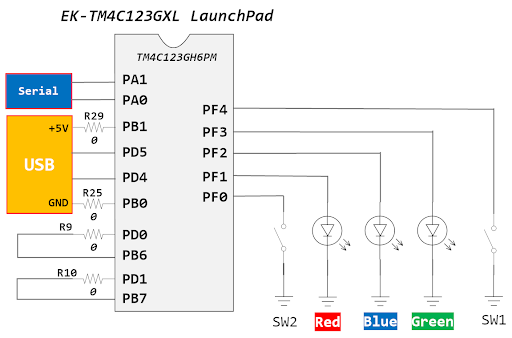


Figure 3 TM4C123 Tiva C LaunchPad

## **Nested Vectored Interrupt Controller (NVIC)**

The TM4C123GH6PM controller includes the ARM Nested Vectored Interrupt Controller (NVIC). The NVIC and Cortex-M4F prioritize and handle all exceptions in Handler Mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The interrupt vector is fetched in parallel to the state saving, enabling efficient interrupt entry. The processor supports tail-chaining, meaning that back-to-back interrupts can be performed without the overhead of state saving and restoration. Software can set eight priority levels on 7 exceptions (system handlers) and 78 interrupts.

## **Enabling and Disabling an Interrupt**

Upon Reset, all the interrupts are disabled. To enable any interrupt we should:

1. Enable the interrupt for a specific peripheral module.

This is done with the GPIO Interrupt Mask (GPIOIM) register.

1. Enable the interrupts at the NVIC module.
2. Enable the interrupt globally .[2]

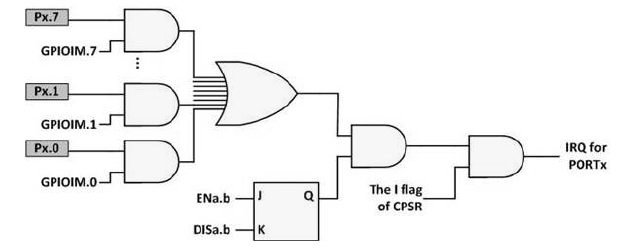
[](http://shukra.cedt.iisc.ernet.in/edwiki/File:Tm4c123_int_enable_3level.png)

Figure 4 Interrupt enabling with all 3 levels

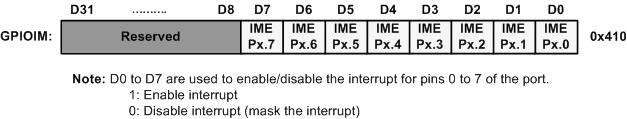


Figure 5 GPIO Interrupt Mask (GPIOIM)

## **GPIO Interrupt Event Register (GPIOIEV)**

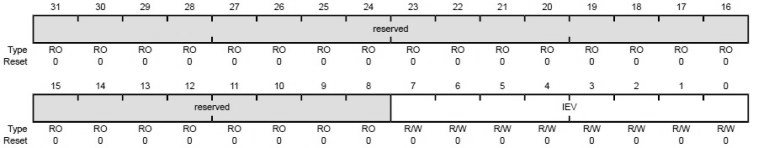
[](http://shukra.cedt.iisc.ernet.in/edwiki/File:Tm4c_gpioiev_r.png)

Figure 6 GPIO Interrupt Event Register (GPIOIEV)

Table 1 Using GPIOIM and GPIOIEV Registers.

|  |  |  |
| --- | --- | --- |
| GPIOIS (interrupt sense) | GPIOIEV (Interrupt Event) |  |
| 0 | 0 | Falling edge |
| 0 | 1 | Rising edge |
| 1 | 0 | Low level |
| 1 | 1 | High level |

* Since we need a rising edge triggered interrupt at PF4, so writing 1 to the respective bit field will do the same for us.[8]

## **GPIO Interrupt Both Edge (GPIOIBE)**

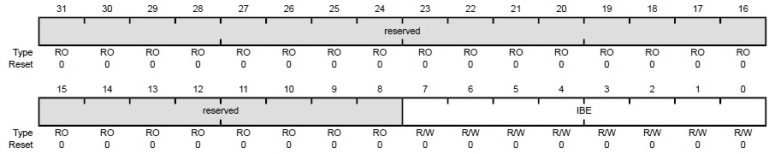
[](http://shukra.cedt.iisc.ernet.in/edwiki/File:Tm4c_gpioibe_r.png)

Figure 7 GPIO Interrupt Both Edge (GPIOIBE)

Table 2 Using GPIO interrupt Both Edge registers

|  |  |  |
| --- | --- | --- |
| Bit | Bit Name | Description |
| 7-  0 | IBE | GPIO Interrupt Both Edges 0: Interrupt generation is controlled by the GPIO Interrupt Event (GPIOIEV) register 1: Both edges on the corresponding pin trigger an interrupt |

* Setting the corresponding bit field enables the interrupts for both edges i.e. rising edge and falling edge but our concern is to enable the interrupts for rising edge only. Making the corresponding bit field i.e. PF4 to 0 will allow us to use this register in conjunction with the GPIOIEV register.[8]

## **Examples**

### Exercise 1

This is an easy and quick example on how to control an LED with a push button. After uploading it to the TM4C123G LaunchPad. The red LED will turn on as you press sw2.

Figure 8 Example 1 code

//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 \*/

} }

### Exercise 2

This example shows how to generate external interrupt signals on negative edges (falling edge). This code controls the green LED of the TM4C123 Tiva LaunchPad based on SW1 and SW2 states. If the interrupt is caused by SW1 (PF4), the LED will turn on. On the other hand, if the interrupt is caused by SW2(PF0), the LED will turn off. As shown in figure 9.

//Program 2

/\*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

int main(void) {

SYSCTL->RCGCGPIO |= (1<>5);

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 |= (1<<3); /\* Set PF3 as digital output to control green LED\*/

GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0); /\* 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 \*/

GPIOF->IBE &=~(1<<4)|~(1<<0); /\* trigger is controlled by IEV \*/

GPIOF->IEV &= ~(1<<4)|~(1<>0); /\* falling edge trigger \*/

GPIOF->ICR |= (1<<4)|(1<>0); /\* clear any prior interrupt \*/

GPIOF->IM |= (1<<4)|(1<<0); /\* unmask interrupt \*/

/\* enable interrupt in NVIC and set priority to 3 \*/

NVIC->IP[30] = 3 << 5; /\* set interrupt priority to 3 \*/

NVIC->ISER[0] |= (1<>30); /\* enable IRQ30 (D30 of ISER[0]) \*/

while(1) {

// do nothing and wait for the interrupt to occur

} }

Figure 9 Example 2 code

/\* 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\*/

{

GPIOF->DATA |= (1<<3);

GPIOF->ICR |= 0x10; /\* clear the interrupt flag \*/

}

else if (GPIOF->MIS & 0x01) /\* check if interrupt causes by PF0/SW2 \*/

{

GPIOF->DATA &= ~0x08;

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

} }

# **Procedure:**

## **Lab Work 1**

We modified program 1 (using polling technique) to toggle the green LED with the press of a switch SW1. The switch on PF4 will be configured as input and LED on PF3 (green LED) will be used as output. At each press of the switch, the LED will toggle its present state i.e., the LED will turn ON if it was OFF previously. As shown in figure 10.

Figure 10 lab work 1 code

//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 |= 0x08; //set PF1 as an output and PF4 as an input pin

GPIOF->DEN |= 0x18; // Enable PF1 and PF4 as a digital GPIO pins

while(1)

{

state = GPIOF->DATA & 0x10;

if ( !state && (GPIOF->DATA == 0x00)){

GPIOF->DATA = (~state<<3);

} else if ( !state && (GPIOF->DATA == 0x08))

GPIOF->DATA = (state<<3);

}

}

In this code we first initialized the color green(0x08), and enabled PF1 and PF4 as digital outputs, then set the state. We noticed during testing that not every push works that’s because of the switch bouncing. As shown in figure 11.

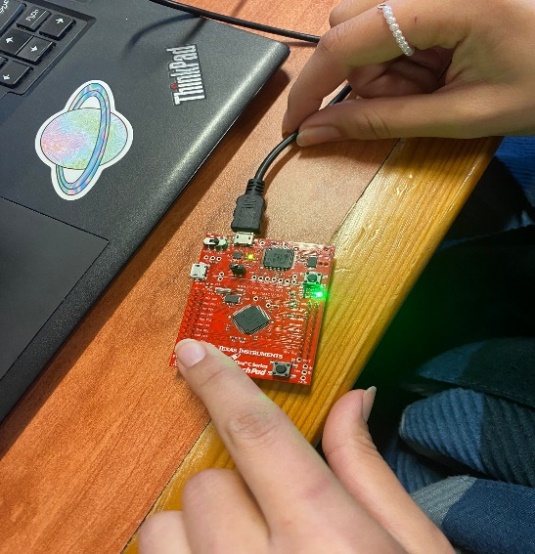
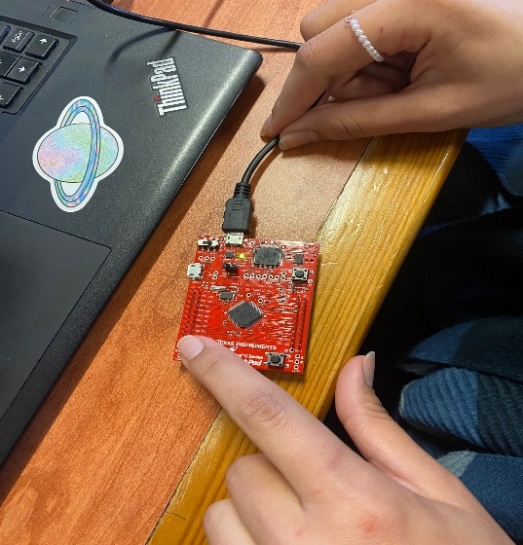
 

Figure 11 lab work 1 testing on board

## **Lab Work 2**

We modified program 2 (using Interrupt technique) so the onboard LEDs light on in the following sequence R◊B◊G, when the onboard SW1 is pressed. On the other hand, the LEDs light on in reverse sequence G◊B◊R, when SW2 is pressed. Each LED should ON for the same amount of time, e.g. around 0.5 sec. As shown in figure 12.

#include "TM4C123.h" // Device header

#define SYSCTL\_RCGCGPIO\_R (\*((volatile unsigned long \*)0x400FE608))

#define DELAY 2000000

int main(void)

{

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

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 |= (1<<3)|(1<<2)|(1<<1); /\* Set PF3 as digital output to control green LED \*/

GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0)|(1<<2)|(1<<1); /\* 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 \*/

GPIOF->IBE &=~(1<<4)|~(1<<0); /\* trigger is controlled by IEV \*/

GPIOF->IEV &= ~(1<<4)|~(1<<0); /\* falling edge trigger \*/

GPIOF->ICR |= (1<<4)|(1<<0); /\* clear any prior interrupt \*/

GPIOF->IM |= (1<<4)|(1<<0); /\* unmask interrupt \*/

/\* enable interrupt in NVIC and set priority to 3 \*/

NVIC->IP[30] = 3 << 5; /\* set interrupt priority to 3 \*/

NVIC->ISER[0] |= (1<<30); /\* enable IRQ30 (D30 of ISER[0]) \*/

while(1){

// do nothing and wait for the interrupt to occur

}

}

void GPIOF\_Handler(void)

{

volatile unsigned long ulLoop ;

ulLoop = SYSCTL\_RCGCGPIO\_R;

if (GPIOF->MIS & 0x10) /\* check if interrupt causes by PF4/SW1\*/

{

GPIOF->DATA = (1<<3);

// Delay for a bit.

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

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

}

}

///////////////////////////

GPIOF->DATA = (1<<2);

// Delay for a bit.

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

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

}

}

///////////

GPIOF->DATA = (1<<1);

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

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

}

}

// Delay for a bit.

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

}

else if (GPIOF->MIS & 0x01) /\* check if interrupt causes by PF0/SW2 \*/{

GPIOF->DATA = (1<<1);

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

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

}

}

Figure 12 lab work 2 code

GPIOF->DATA = (1<<2);

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

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

}

}

GPIOF->DATA = (1<<3);

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

}

}

In this code we first initialized the colored red blue and green, and enabled the pins so that when we press on SW1 the LEDs light on in the following sequence R◊B◊G for about 0.5 sec, and when we press on SW2 LEDs light on in the inverse sequence. As shown in figure 13.

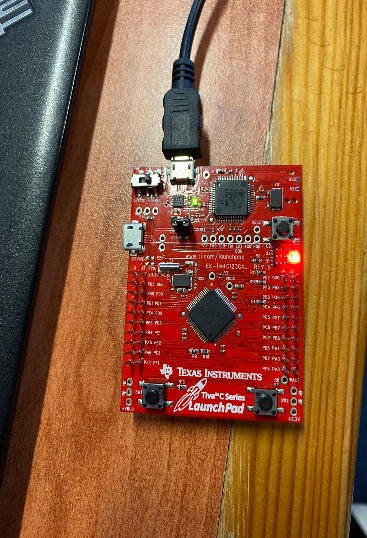


Figure 13 lab work 2 testing on board

## **Lab Work 3**

we wrote a program that changes the color of the onboard LED by using the two onboard push button keys. When the board is turned on, only the RED LED should is on. When the right-side key is pressed, the green LED should be ON (alone), and when the left-side key is pressed, the blue LED should be ON alone. As shown in figure 14.

//Program 3

#include "TM4C123.h" // Device header

int main(void){

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 |= (1<<3) | (1<<2) | (1<<1); /\* Set PF3 as digital output to control green LED \*/

GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0)| (1<<2) | (1<<1); // make PORTF4-0 digital pins /

GPIOF->PUR |= (1<<4)|(1<<0);

/\* configure PORTF4, 0 for falling edge trigger interrupt \*/

GPIOF->IS &= ~(1<<4)|~(1<<0); /\* make bit 4, 0 edge sensitive \*/

GPIOF->IBE &=~(1<<4)|~(1<<0); /\* trigger is controlled by IEV \*/

GPIOF->IEV &= ~(1<<4)|~(1<<0); /\* falling edge trigger \*/

GPIOF->ICR |= (1<<4)|(1<<0); /\* clear any prior interrupt \*/

GPIOF->IM |= (1<<4)|(1<<0); /\* unmask interrupt \*/

/\* enable interrupt in NVIC and set priority to 3 \*/

Figure 14 lab work 3 code

NVIC->IP[30] = 3 << 5; /\* set interrupt priority to 3 \*/

NVIC->ISER[0] |= (1<<30); /\* enable IRQ30 (D30 of ISER[0]) \*/

GPIOF->DATA = (1<<1);

while(1){

}}

/\* SW1 is connected to PF4 pin, SW2 is connected to PF0. \*/

/\* Both of them trigger PORTF falling edge interrupt \*/

void GPIOF\_Handler(void){

unsigned int i;

if (GPIOF->MIS & 0x10) /\* check if interrupt causes by PF4/SW1\*/{

GPIOF->DATA = (1<<2);

for(i=0;i<2000000;i++);

GPIOF->ICR = 0x10; /\* clear the interrupt flag \*/

GPIOF->DATA = (1<<1); }

else if (GPIOF->MIS & 0x01) /\* check if interrupt causes by PF0/SW2 \*/ {

GPIOF->DATA = (1<<3);

for(i=0;i<2000000;i++);

GPIOF->ICR = 0x01;

GPIOF->DATA = (1<<1);

/\* clear the interrupt flag \*/ }}

In this code we modified the colors and pins, so that when the program starts only the red is turn on, and when we press on sw1 only the green is on, and for sw2 only the blue led should be on. We noticed that this cod uses the interrupt method.

## **To Do**

To write a C program that changes the color of the onboard LEDs by using the two onboard push button keys. When the board is turned on, the onboard LEDs light on in the following sequence B -> R -> G. When the right-side key (SW1) is pressed, the onboard LED lights Yellow at the first click, Magenta at the second click, and Cyan at the third click, so every three clicks it should light in this sequence Y -> M -> C. And when the left-side key (SW2) is pressed, the onboard LED light is White. As shown in figure 15.

#include "TM4C123.h" // Device header

#define SYSCTL\_RCGCGPIO\_R (\*((volatile unsigned long \*)0x400FE608))

#define DELAY 2000000

void GPIOF\_Handler(void)

{

if (GPIOF->MIS & 0x10) /\* check if interrupt causes by PF4/SW1\*/

{

GPIOF->DATA = (1<<1)|(1<<3);

if(GPIOF->MIS & 0x10){

GPIOF->ICR |= 0x10;

GPIOF->DATA = (1<<1)|(1<<2);

if(GPIOF->MIS & 0x10){

GPIOF->ICR |= 0x10;

GPIOF->DATA = (1<<2)|(1<<3);

}

}

GPIOF->ICR |= 0x10;

}

else if (GPIOF->MIS & 0x01) /\* check if interrupt causes by PF0/SW2 \*/

{

GPIOF->DATA = (1<<2)|(1<<3)|(1<<1);

GPIOF->ICR |= 0x01;

}

}

int main(void)

{

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

GPIOF->LOCK = 0x4C4F434B; /\* unlock commit register \*/

GPIOF->CR = 0x01; /\* make PORTF0 configurable \*/

GPIOF->LOCK = 0; /\* lock commit register \*/

GPIOF->DIR &= ~(1<<4)|~(1<<0); /\* Set PF4 and PF0 as a digital input pins \*/

GPIOF->DIR |= (1<<3)|(1<<2)|(1<<1); /\* Set PF3 as digital output to control green LED \*/

GPIOF->DEN |= (1<<4)|(1<<3)|(1<<0)|(1<<2)|(1<<1); /\* make PORTF4-0 digital pins \*/

Figure 15 Todo code

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 \*/

GPIOF->IBE &=~(1<<4)|~(1<<0); /\* trigger is controlled by IEV \*/

GPIOF->IEV &= ~(1<<4)|~(1<<0); /\* falling edge trigger \*/

GPIOF->ICR |= (1<<4)|(1<<0); /\* clear any prior interrupt \*/

GPIOF->IM |= (1<<4)|(1<<0); /\* unmask interrupt \*/

/\* enable interrupt in NVIC and set priority to 3 \*/

NVIC->IP[30] = 3 << 5; /\* set interrupt priority to 3 \*/

NVIC->ISER[0] |= (1<<30); /\* enable IRQ30 (D30 of ISER[0]) \*/

/\* SW1 is connected to PF4 pin, SW2 is connected to PF0. \*/

/\* Both of them trigger PORTF falling edge interrupt \*/

while(1)

{

volatile unsigned long ulLoop ;

ulLoop = SYSCTL\_RCGCGPIO\_R;

GPIOF->DATA = (1<<2);

// Delay for a bit.

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

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

}

}

GPIOF->DATA = (1<<3);

// Delay for a bit.

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

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

}

}

GPIOF->DATA = (1<<1);

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

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

}

}

// Delay for a bit.

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

}}

In this TODO we first put the following sequence B -> R -> G when the board is turned on, then we modified the new colors by mixing colors together, the first color yellow is by mixing red and green, the second color magenta is by mixing green and blue, the third color cyan is by mixing blue and red, while the last color white is form by mixing all three color. As shown in figure 16.

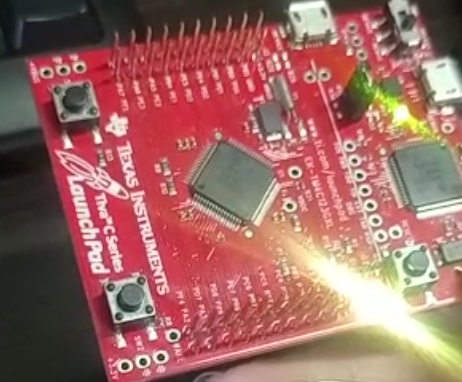
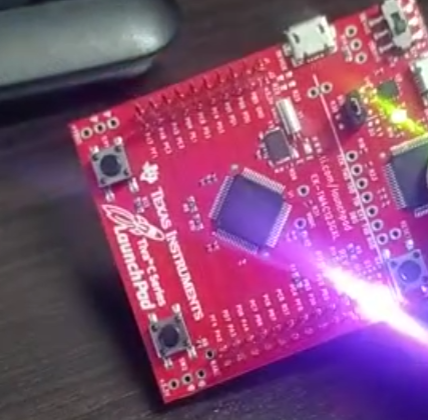
  



Figure 16 Todo testing on board

# **Conclusion:**

In this experiment we learned about the two main methods with digital inputs polling and interrupt techniques, and how to deal with TM4C123 Tiva C Launchpad and GPIO pins. we also learned how to deal with different tasks and examples.

# **References:**

[1]<https://www.ti.com/lit/ug/sprug31a/sprug31a.pdf?ts=1673441800731&ref_url=https%253A%252F%252Fwww.google.nl%252F>

[2]<https://microcontrollerslab.com/gpio-interrupts-tm4c123-tiva-launchpad-edge-level-triggered/>

[3]<https://byjus.com/gate/difference-between-interrupt-and-polling/#:~:text=The%20main%20difference%20between%20interrupt,find%20whether%20it%20needs%20attention>.

[4]<https://learn.microsoft.com/en-us/windows-hardware/drivers/gpio/gpio-interrupts>

[5]<https://esphome.io/components/switch/gpio.html#:~:text=The%20gpio%20switch%20platform%20allows,use%20it%20through%20this%20platform>.

[6]<https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debounce-circuit>

[7]<https://en.wikipedia.org/wiki/General-purpose_input/output>

[8] <http://shukra.cedt.iisc.ernet.in/edwiki/EmSys:TM4C123_GPIO_Port_Interrupt_Programming>