Lab-2 Group -4 report

(210020050,210020007)

Aim:

PART 0:

- From the previous exercise (LED Blink), check that the delay generated is equal to the expected value based on clock speed and number of instructions/cycles.
- Change the declaration of the loop counter variable to register int x to see how the assembly changes.
- Look at the SP and PC values and compare them with the memory map in the datasheet. Which types of memory are these registers addressing?

PART 1:

- Read the state of one of the switches (either SW1 or SW2) connected to the GPIO pins of the Tiva chip.
- Whenever the button is pressed, the LED should light up RED.
- Whenever the button is not pressed, the LED should be off.

PART 2:

- Read the state of one of the switches (either SW1 or SW2) connected to the GPIO pins of the Tiva chip.
- Upon each button being pressed, the LED color should change sequentially (Red -> Green -> Blue -> Red -> Green...)
- Make sure to implement some debouncing so that each button press causes exactly one color change.

Observation:

Part 0:

```
// LAB2-PART 0
#if Q0
  register uint current = 0;
  while (1)
  {
    GPIO_PORTF_DATA_R = 0x02;
    for (current = 0; current < COUNTER; current++)
      continue;
    GPIO_PORTF_DATA_R = 0x00;
    for (current = 0; current < COUNTER; current++)
      continue;
    current++;
  }
#endif
```

1. We observed when we change counter's qualifier to register we found out it is taking only 7 cycles for load and store, It is being stored directly in register.
2. We observed code was working as expected, It only turns on when the button is clicked

Part 1:

```
    const uint PortF_Red_Light = 1 << 1;
// LAB-2 PART-1
#if Q1
    uint clicked = 0;
    while (1)
    {
        clicked = GPIO_PORTF_DATA_R & 0x10;
        if (clicked == 0x0)
            GPIO_PORTF_DATA_R |= PortF_Red_Light;
        else
            GPIO_PORTF_DATA_R &= ~PortF_Red_Light;
    }
#endif
```

1. We didn't want to change whole value of PORT_F instead, we used mask to only change 2nd bit for red color
2. We observed code was working as expected, It only turns on when the button is clicked

Part 2:

```
void Delay(uint32_t count)
{
    volatile uint32_t i;
    for (i = 0; i < count; i++)
        continue;
}
// LAB2-PART 2
#if Q2
    uint current = 0;
    uint prev = 0, clicked = 0;
    while (1)
```

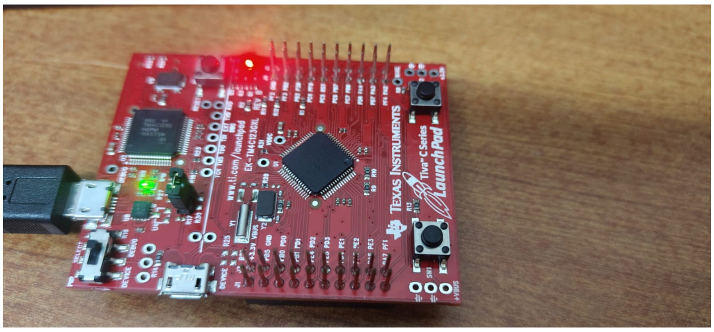
```
  {
    clicked = GPIO_PORTF_DATA_R & 0x10;
    if (prev == 0x10 & clicked == 0x0)
      current = (current + 1) % 3;
    Delay(100); /* debouncing */

    if (current % 3 == 0)
    {
      GPIO_PORTF_DATA_R = 0x02;
    }
    if (current % 3 == 1)
    {
      GPIO_PORTF_DATA_R = 0x04;
    }
    if (current % 3 == 2)
    {
      GPIO_PORTF_DATA_R = 0x08;
    }
    prev = clicked;
  }
#endif
```

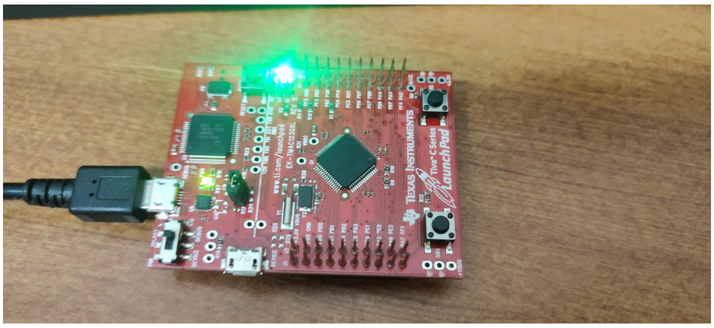
1. We waited for some time after checking for button click we thought wasting 100 cycles might be good for debouncing
2. We observed code was working as expected, It only turns on when the button is clicked

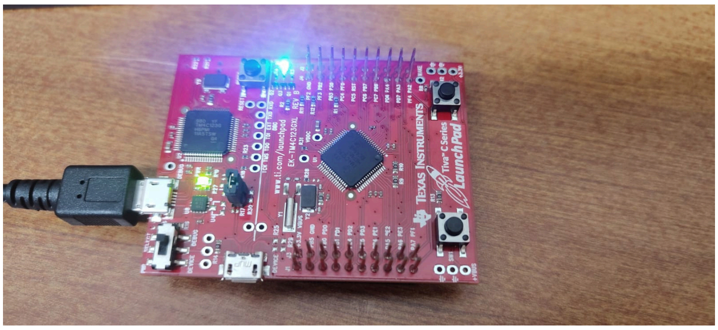
Conclusion:

1. We learned about register keyword significance and how SP and PC address memory from disassembly.
2. Also how to read to know how to read switch value and use it to switch in code.
3. We also understood limitations of physical switch and the importance of debouncing code

Images:







Complete Code:

```
#include <stdint.h>

#include <stdbool.h>

#include "tm4c123gh6pm.h"


#define COUNTER 800000


// Set these macros to 1 to enable the corresponding lab part

#define Q1 1

#define Q2 0

#define Q0 0
```

```c
void GPIO_Init(void)
{
    SYSCTL_RCGCGPIO_R |= 0x20;          // Enable clock to GPIOF
    GPIO_PORTF_LOCK_R = 0x4C4F434B;    // Unlock GPIO Port F
    GPIO_PORTF_CR_R = 0x1F;            // Allow changes to PF4-PF0
    GPIO_PORTF_DEN_R = 0x1E;           // Enable digital functionality for PF1-PF4
    GPIO_PORTF_DIR_R |= 0x0E;          // Set PF1-PF3 as output (LEDs)
    GPIO_PORTF_DIR_R &= ~0x10;         // Set PF4 as input (SW1)
    GPIO_PORTF_PUR_R |= 0x10;          // Enable pull-up resistor on PF4
}

void Delay(uint32_t count)
{
    volatile uint32_t i;
    for (i = 0; i < count; i++);
}

int main(void)
{
    GPIO_Init();

    // MASK VALUE
    const int PortF_Red_Light = 1 << 1;

    // LAB-2 PART-1: Toggle Red LED based on switch press
#if Q1
    while (1)
    {
        int clicked = GPIO_PORTF_DATA_R & 0x10;
        if (clicked == 0x00)
```

```
            GPIO_PORTF_DATA_R |= PortF_Red_Light;  // Turn on Red LED
        else
            GPIO_PORTF_DATA_R &= ~PortF_Red_Light; // Turn off Red LED
    }
#endif

    // LAB2-PART 2: Cycle through Red, Blue, Green LEDs on button press
#if Q2
    int current = 0;
    int prev = 0, clicked = 0;

    while (1)
    {
        clicked = GPIO_PORTF_DATA_R & 0x10;

        if (prev == 0x10 && clicked == 0x00)  // Detect falling edge (button press)
        {
            current = (current + 1) % 3;  // Increment and wrap around current state
        }

        Delay(1000);  // Debouncing delay

        // Clear PF1-PF3 (turn off all LEDs)
        GPIO_PORTF_DATA_R &= ~0x0E;

        // Set the appropriate LED based on current state
        switch (current)
        {
        case 0:
            GPIO_PORTF_DATA_R |= 0x02;  // Red LED
            break;
```

```
        case 1:
            GPIO_PORTF_DATA_R |= 0x04;  // Blue LED
            break;
        case 2:
            GPIO_PORTF_DATA_R |= 0x08;  // Green LED
            break;
        }

        prev = clicked;
    }
#endif

    // LAB2-PART 0: Toggle Red LED with a delay
#if Q0
    while (1)
    {
        GPIO_PORTF_DATA_R = 0x02;  // Turn on Red LED
        Delay(COUNTER);

        GPIO_PORTF_DATA_R = 0x00;  // Turn off Red LED
        Delay(COUNTER);
    }
#endif

    return 0;
}
```