EEL 4742C: Embedded Systems
Name: Hamzah Ullah
Lab 1: Flashing the LEDs

Introduction:

As the first lab in the class, lab 1 purpose is to introduce us to the Code Composure Studio program and the correct menu options required to start a C code project. Furthermore, it also introduces us to the correct masking procedure and the difference between mapping the direction by an equivalence (Data = 0x01) which is incorrect, versus using AND and OR equivalence (Data |= 0x01, Data &= 0x01). It also stresses the importance of using the '#define' code to create constants for the bit (#define redLED BIT0) based off the mapping on the board.

The bulk of the code is given to us to blink the redLED, and from there, we use,manipulate and add to the code to complete the rest of the lab, which include giving a delay, blinking the green and red LED in synchrony, asynchrony and at the same time.

Part 1: Documentation

Part 1 entails the different parts of the documentation given, and the materials being used in the lab from here on out. The parts are as followed:

MSP430FR6989 Launchpad Board with documentation:

- MSP430FR6xx Family users Guide
- FR6xx Chip Data Sheet
- Launchpad Board User's Guide

Booster Pack:

- Booster Pack User's Guide
- Various data sheets for each component of Booster pack

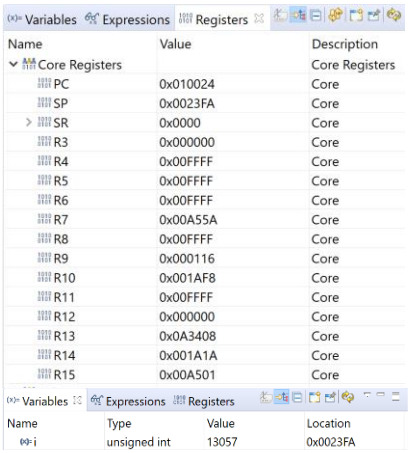
The booster pack is mentioned, but not used in the current lab.

Part 2: Flashing the LEDs

This is the first introduction to coding. Given the code in the book, we are to create a new Empty Project in .c and copy paste the code from the lab in the book to the file. A reminder is given to correct the inverse and xor symbol would need to be manually corrected. The following code is used to **blink the red LED:**

```c
#include <msp430fr6989.h>
#define redLED BIT0 //Red LED at P1.0

void main(void)
{
    volatile unsigned int i;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
                          //high-on impedance mode
    P1DIR |= redLED;    //Direct pin as output
    P1OUT &= ~redLED;   //Turn LED off

    for(;;)
    {
        //Delay Loop
        for(i=0; i<20000; i++){}

        P1OUT ^= redLED; //Toggle the LED
    }


}
```

Registers and Variables:



Changing the values of he for loop on line 16 increases and decreases the rate at which the red LED blinks. The lower the number, the faster it blinks (ex. For(i=0;i<10000;i++) is faster than For(i=0;i<40000;i++)).

Questions:

Run in Debug mode and press reset, does it work? NO

Run in normal mode and test the reset, does it work? YES it pauses the LED from blinking.

Disconnect the board and plug it back in, does the code resume running? YES, code preserved in flash.

Part 3: Setting a Long period:

This part is to show the limitation of the 16 bit architecture in setting a delay. The maximum delay is 65536 using the volatile unsigned int. This part of the lab tells us there is 2 ways to create a delay longer than the max. The lab requires us to create a delay of 120000 using both methods. The first way is using a nested delay. The way it would work using the default 20000 delay and nesting it within a loop that would iterate 6 times till the LED blinks at the end of the call. (20000 * 6 = 120000). The second way is to use a new variable; volatile uint32_5 i;. This makes the integer up to 32 bits long up to 4.2 million long as opposed to 65000 in 16 bit (2^{32} vs. 2^{16}). Below is both methods for a long delay.

Nested Loop

```
1 #include <msp430fr6989.h>
2 #define redLED BIT0 //Red LED at P1.0
3
4 void main(void)
5 {
6     volatile unsigned int i;
7     int j;
8     WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
9     PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
10                          //high-on impedance mode
11    P1DIR |= redLED;    //Direct pin as output
12    P1OUT &= ~redLED;   //Turn LED off
13
14    for(;;)
15    {
16        //Delay Loop
17       for(j=0; j<6; j++){
18        for(i=0; i<20000; i++){}
19
20       //Toggle the LED
21       }
22       P1OUT ^= redLED;
23    }
24 }
```

volatile uint32_t

```
1 #include <msp430fr6989.h>
2 #include <stdint.h>
3 #define redLED BIT0 //Red LED at P1.0
4
5 void main(void)
6 {
7  //  volatile unsigned int i;
8    volatile uint32_t i;
9    int j;
10   WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
11   PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
12                         //high-on impedance mode
13   P1DIR |= redLED;    //Direct pin as output
14   P1OUT &= ~redLED;   //Turn LED off
15
16   for(;;)
17   {
18       //Delay Loop
19       for(i=0; i<120000; i++){}
20      //Toggle the LED
21      P1OUT ^= redLED;
22   }
23 }
24
```

Part 4: Flashing two LEDs

This part of the lab requires us to code and demonstrate the green LED to on, and flash them both in asynchronous and at the same time. We are to determine the mask used to define the green LED and the direction , and initialize it the opposite the red LED and toggle the lights using the XOR operations. The bit required is #define greenLED BIT7 and the direction is P9DIR and P9OUT for the output. Below are the required output Code.

Synchronous:

```c
#include <msp430FR6989.h>
#include <stdint.h>
#define redLED BIT0 //Red LED at P1.0
#define greenLED BIT7 //Greed LED at P9.7

void main(void)
{
    volatile unsigned int i;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
                          //high-on impedance mode
    P1DIR |= redLED;    //Direct pin as output
    P1OUT &= ~redLED;   //Turn LED off

    P9DIR |= greenLED;    //Direct pin as output
    P9OUT &= ~greenLED;    //LED on

    for(;;)
    {
        //Delay Loop
        for(i=0; i<40000; i++){}
        P1OUT ^= redLED; //Red LED blink
        P9OUT ^= greenLED;
    }
}
```

Asynchronous

```c
#include <msp430fr6989.h>
#include <stdint.h>
#define redLED BIT0 //Red LED at P1.0
#define greenLED BIT7 //Greed LED at P9.7
void main(void)
{
    volatile unsigned int i;
//   volatile uint32_t i;
//   int j;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
                          //high-on impedance mode
    P1DIR |= redLED;    //Direct pin as output
    P1OUT &= ~redLED;   //Turn LED off

    P9DIR |= greenLED;    //Direct pin as output
    P9OUT &= greenLED;    //LED on

    for(;;)
    {
        //Delay Loop
        for(i=0; i<20000; i++){}
       //Toggle the LED
       P1OUT ^= redLED;
       P9OUT ^= greenLED;
    }
}
```

Part 5: Flashing two LEDs at different Rates:

The best way for implementation is to nest an LED in a loop to blink twice before the second led turns on once which is outside of the loop. Below is the code below for the red LED blinking twice as fast as the green LED.

Blinking at different rates

```c
#include <msp430FR6989.h>
#include <stdint.h>
#define redLED BIT0 //Red LED at P1.0
#define greenLED BIT7 //Greed LED at P9.7

void main(void)
{
    volatile unsigned int i;
    int j;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; //Disable GPIO power-on default
                          //high-on impedance mode
    P1DIR |= redLED;    //Direct pin as output
    P1OUT &= ~redLED;   //Turn LED off

    P9DIR |= greenLED;    //Direct pin as output
    P9OUT &= greenLED;    //LED on

    for(;;)
    {
        //Delay Loop
        for(j=0; j<2;j++ ){
        for(i=0; i<40000; i++){}
        P1OUT ^= redLED; //Red LED blink
        }
        P9OUT ^= greenLED;
    }
}
```

QUESTIONS:

1. **In this lab, we used a delay loop to create a small delay; what is its effect on the battery life if the device is battery operated? Is it a good way of generating a delay?**

   The effect on the battery life is that it will keep draining the battery even if it is doing nothing. The delay is telling the system to wait for the next blink, as opposed to idling or 'going to sleep'. It is good in terms of convenience in coding and implementation, but will continue to drain the battery so it is not best for everyday use or a low power system.

2. **The MSP430 CPU runs on a clock that can be slowed down or sped up; what happens to the delay generated by the loop if the clock driving the CPU speeds up or slows down?**

   The delay would change if you change the speed of the clock. As the clock increases, the delay would decrease because the clock would reach the defined delay threshold much sooner than a slower clock. If you slow the clock down the delay would take longer to implement because the distance between the clock trigger is larger.

3. **How does the code run in debug mode? Is the microcontroller running as an independent computer?**

   The MCU runs dependent on the CPU, allowing the user to manipulate the device right from their CPU, allowing you to view critical information such as variables, step through the debug mode to fix errors, and stopping and starting the code. The microcontroller is not an independent computer.

4. **How does the code run in normal mode? Is the microcontroller running as an independent computer?**

   In normal mode, the computer has no control over the MCU. The buttons become usable as manufactured intention (unless coded) and whatever the last code that was implemented in the chip will continue to run unless we reenter debug mode or reprogram it through the program. The microcontroller is running as an independent computer.

5. **In which mode does the reset button work?**

   The reset button works in Normal Mode.

6. **What is the data type uint16_t? What about int16_t? Are these standard C syntax?**

   The types are unsigned integer 16 bit and integer 16 bit. These are standard C syntax.