EEL 4742C: Embedded Systems
Name: Hamzah Ullah
Lab 2: Using the Push Buttons

Introduction:

For this lab, we will be using two features on the MSP4306989 which is the push buttons. The push buttons mapped at P1.1 and P1.2 for button 1 and 2 respectively are being used to demonstrate how we turn the buttons on, and when a input data is detected, in conjunction with activation of the button, will yield some result. The result in this lab, is the redLED and greenLED reacting based on our code. The first thing to do is the enable the buttons. To do so we are introduced to the P1REN which is like a switch to enable the buttons to be manipulated in the code. This lab entails turning on the redLED with a button, the greenLED with another button, and testing the limits by setting parameters for the buttons so they are more in control with what we want. Finally, we are to design our own code to manipulate the buttons and LEDs.

Part 1: Reading the Push Buttons

This part of the lab introduces us to how to assign a button to an LED and turn it on. To do that we need to enable the resistor required for the button on with P1REN. To do that with simply OR-ing it with the button to enable that button. Furthermore, we are to set the output for the button by OR-ing it with the P1OUT. That only enables the button but does not turn on the LED. To turn the LED on, we have to set parameters to check the status of the button, then when that status is met, turn on the LED by OR-ing it. Otherwise, the light remains off. Below is the code that shows the required missing blanks, where BUT1 outputs the redLED.

Turn on the redLED with BUT1 code:

```
#include <msp430fr6989.h>
#define redLED BIT0 //red led at p1.0

#define greenLED BIT7 //green led at p9.7
#define BUT1 BIT1 //button s1 at p1.1
#define BUT2 BIT2 //button s1 at p1.1
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;    //enable the gpio pins

    //Configure and inintialize LEDs
    P1DIR |= redLED;//Direct pin as output
    P9DIR |= greenLED;//Direct pin as output
    P1OUT &= ~redLED;//Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure buttons
    P1DIR &= ~BUT1;
    P1REN |= BUT1;
    P1OUT |= BUT1;

//  //Configure buttons
//  P1DIR &= ~BUT2;
//  P1REN |= BUT2;
//  P1OUT |= BUT2;


    //Polling the button in an infinite loop
    for(;;){
        //Fill the if-statement below... RED LED

        if((P1IN & BUT1 ) == 0)
        P1OUT |= redLED;
        else
        P1OUT &= ~redLED;

//      //Fill the if-statement below... GREEN LED
//      if((P1IN & BUT2 ) == 0)
//      P9OUT |= greenLED;
//      else
//      P9OUT &= ~greenLED;

    }
}
```

Part 2: Using Two Buttons

For this part of the lab, we are to configure both buttons and have button 1 (BUT1)control the redLED and button 2(BUT2) control the greenLED, so when they are pressed, they have independent controls of their respective LED with no interference. To do this, BUT2 has to be mapped to the output of the greenLED located at P9.7. By enabling BUT2 and initializing the greenLED, we are able to do the same thing as the red LED. Below is the code I used to demonstrate this in the lab.

Turning on the redLED with BUT1 and the greenLED with BUT2:

```
#include <msp430fr6989.h>
#define redLED BIT0 //red led at p1.0

#define greenLED BIT7 //green led at p9.7
#define BUT1 BIT1 //button s1 at p1.1
#define BUT2 BIT2 //button s1 at p1.1
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;    //enable the gpio pins

    //Configure and inintialize LEDs
    P1DIR |= redLED;//Direct pin as output
    P9DIR |= greenLED;//Direct pin as output
    P1OUT &= ~redLED;//Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure buttons
    P1DIR &= ~BUT1;
    P1REN |= BUT1;
    P1OUT |= BUT1;

    //Configure buttons
    P1DIR &= ~BUT2;
    P1REN |= BUT2;
    P1OUT |= BUT2;



    //Polling the button in an infinite loop
    for(;;){
        //Fill the if-statement below... RED LED

        if((P1IN & BUT1 ) == 0)
        P1OUT |= redLED;
        else
        P1OUT &= ~redLED;

        //Fill the if-statement below... GREEN LED
        if((P1IN & BUT2 ) == 0)
        P9OUT |= greenLED;
        else
        P9OUT &= ~greenLED;

    }
}
```

Part 3: Using Two Buttons with Exclusive Access

For this part of the lab, we are to set parameters for BUT1 and BUT 2 such that if one of the buttons is pressed the LED will turn on, and while its pressed, when you try to press the other button, the other LED wont turn on because of a stopping mechanism or coding sequencing. To do this we will use a while loop to code it in such a way that when the button is pressed, it is stuck in an infinite loop and the other button cannot send an input in unless the button is depressed. Below is the code I used with while loops. When both are pressed at the same time, whichever happens to be pressed first turns on, otherwise if exactly simultaneously, I believe nothing would happen.

BUT1 infinite on with while loop displaying a redLED, same with BUT2 with greenLED.

```
#include <msp430fr6989.h>
#define redLED BIT0 //red led at p1.0

#define greenLED BIT7 //green led at p9.7
#define BUT1 BIT1 //button s1 at p1.1
#define BUT2 BIT2 //button s1 at p1.1
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;    //enable the gpio pins
    volatile unsigned int i;

    //Configure and initialize LEDs
    P1DIR |= redLED;//Direct pin as output
    P9DIR |= greenLED;//Direct pin as output
    P1OUT &= ~redLED;//Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure buttons
    P1DIR &= ~BUT1;
    P1REN |= BUT1;
    P1OUT |= BUT1;

    //Configure buttons
    P1DIR &= ~BUT2;
    P1REN |= BUT2;
    P1OUT |= BUT2;

    //Polling the button in an infinite loop
    for(;;){
        //Fill the if-statement below... RED LED
        while((P1IN & BUT1 ) == 0){
        if((P1IN & BUT1 ) == 0)
        P1OUT |= redLED;
        }
        P1OUT &= ~redLED;
        //Fill the if-statement below... GREEN LED
        while((P1IN & BUT2 ) == 0){
        if((P1IN & BUT2 ) == 0)
        P9OUT |= greenLED;
        }
        P9OUT &= ~greenLED;
    }
}
```

Part 4: Your Own Design

For my own design, I tried to incorporate the last lab by having the LEDs blink. So I made a dedicated on off button, where BUT1 is ON and BUT2 is OFF. When BUT1 is pressed the redLED will stay lit even if BUT1 is depressed, but the greenLED will blink until BUT1 is depressed, and depending on the state of greenLED, the light will stay solid green or turn off. Below is the code below:

BUT1 turns on redLED and blinks greenLED, BUT2 turns off redLED and greenLED.

```
#include <msp430fr6989.h>
#define redLED BIT0 //red led at p1.0

#define greenLED BIT7 //green led at p9.7
#define BUT1 BIT1 //button s1 at p1.1
#define BUT2 BIT2 //button s1 at p1.1
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;    //enable the gpio pins
    volatile unsigned int i;

    //Configure and initialize LEDs
    P1DIR |= redLED;//Direct pin as output
    P9DIR |= greenLED;//Direct pin as output
    P1OUT &= ~redLED;//Turn LED off
    P9OUT &= ~greenLED; //Turn LED off

    //Configure buttons
    P1DIR &= ~BUT1;
    P1REN |= BUT1;
    P1OUT |= BUT1;

    //Configure buttons
    P1DIR &= ~BUT2;
    P1REN |= BUT2;
    P1OUT |= BUT2;

    //Polling the button in an infinite loop
    for(;;){
        //BUT1 turns on the redLED till BUT2 pressed, and
        //Blinks the green LED
        if((P1IN & BUT1 ) == 0){
            for(i=0;i<25000;i++){}
                P9OUT ^= greenLED;
                P1OUT |= redLED;
        }
        //BUT2 turns off redLED and greenLED
        if((P1IN & BUT2 ) == 0){
               P9OUT &= ~greenLED;
               P1OUT &= ~redLED;
        }
    }
}
```

QUESTIONS:

1. **When a pin is configured as input, P1IN is used for data, which leaves P1OUT available for other use? In such case, what is P1OUT used for?**
   P1OUT is used to configure the resistor as a pull-up resistor or a pull-down resistor when the P1IN is used for data.

2. **A programmer wrote this line of code to check if bit 3 is equal to 1: if((Data & BIT3) == 1). Explain why this if-statement is incorrect?**
   The statement is incorrect because if BUT3 & Data are used, it would come up false because the bit 1 is not on the rightmost position whereas it's better to check for != 0 or == 0 as an alternative. It would poll a continuous false.

3. **Comment on the codes power-efficiency if the device is battery operated. Is reading the button via polling power efficient?**
   It is not efficient because the act of polling does not stop, so if the device was battery operated it would continuously drain the battery. The code continues to check the output till a 0 is masked TRUE for BIT3 & Data(P1IN). Because of that, the clock continues running. According to the lab manual a more efficient way is to trigger an interrupt and put the MCU in low power mode and wait for the push button.