**9/20/2020**

Joshua Frazer

Lab 2

Using the Push Buttons

**Introduction:**

For this lab I dealt solely on the push buttons. Part 1 we start to get the gist of how to operate the functionalities of the button S1 to turn the red LED on and off. For the next part, we basically copy to what we did in Part 1 for an additional button S2 and the green LED. Part 3 keeps everything we have learned thus far and motivated us to push towards exclusive access on both buttons. As for Part 4, I decided to reverse the effects of Part 3. Meaning, instead of having the LEDs turn on when you click the button, I had them turn off.

**Part 1:**

For this part, we gathered much information on the buttons associated with our board. Our most significant bit of information we gained was the port configuration for the buttons that would be essential for the solution of this part. Once the configuration of the button was set up properly, we had to find out what masking operation was necessary to accomplish our task. As you can see we had to compare the input pin and whether BUT1 was active or not.

// Turning on the red LED while button S1 is pushed

**#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 P1.112

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

// 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; // Direct pin as input

P1REN |= BUT1; // Enable built-in resistor

P1OUT |= BUT1; // Set resistor as pull-up

// Polling the button in an infinite loop

**for**(;;) {

// Fill the if-statement below...

**if** ((P1IN & BUT1) == 0)

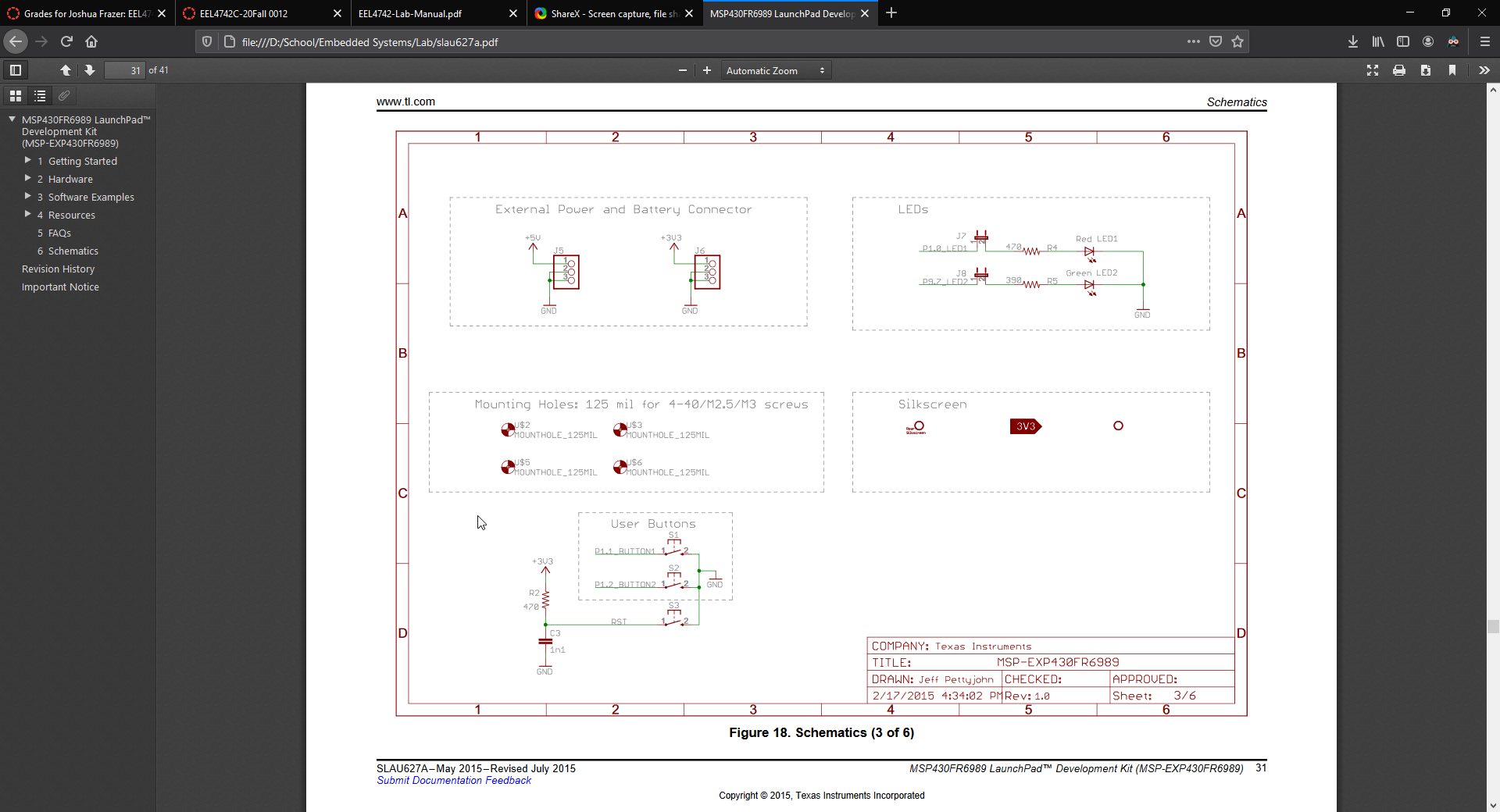
P1OUT |= redLED; // Turn red LED on

**else**

P1OUT &= ~redLED; // Turn red LED off

}

}

**Button Schematic:**

The active low and pull-up resistor is used so you can read that the button is active upon the push of it.

**Questions:**

No questions were given.

**Part 2:**

This part was very simple. Many copy and pasting was utilized to achieved our final goal. We are tasked to activate and configure BUT2 just like BUT1. Similarly we are tasked to set up the green LED just like the red LED in Part 1. As you can see from my code. I copied the same configuration for BUT2 from BUT1 and I added another if-else statement to toggle the green LED just like the red LED.

// Turning on the red LED while button S1 is pushed

// and turning on the green LED while button S2 is pushed

**#include** <msp430fr6989.h>

**#include** <stdio.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 S2 at P1.2

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

// 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; // Direct pin as input

P1REN |= BUT1; // Enable built-in resistor

P1OUT |= BUT1; // Set resistor as pull-up

P1DIR &= ~BUT2; // Direct pin as input

P1REN |= BUT2; // Enable built-in resistor

P1OUT |= BUT2; // Set resistor as pull-up

// Polling the button in an infinite loop

**for**(;;) {

// Fill the if-statement below...

**if** ((P1IN & BUT1) == 0)

P1OUT |= redLED; // Turn red LED on

**else**

P1OUT &= ~redLED; // Turn red LED off

**if** ((P1IN & BUT2) == 0)

P9OUT |= greenLED; // Turn green LED on

**else**

P9OUT &= ~greenLED; // Turn green LED off

}

}

**Questions:**

No questions were given.

**Part 3:**

Part 3 keeps everything the same outside of the infinite loop. The only code that gets modified within this part is within the infinite loop. To make both buttons exclusive access, you have the case of if a button is pushed and activated the other button should not matter. This is both my if in the if-else if statements that I have. The other case is on the toggle of the LED. We need to make sure that if an LED is already on we can not toggle another one until it is deactivated. This solution is found within the else if part of the if-else if statements that I have. Once these both are implemented you then have exclusive access for both buttons.

// Turning on the red LED while button S1 is pushed

// and turning on the green LED while button S2 is pushed with exclusive access

**#include** <msp430fr6989.h>

**#include** <stdio.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 S2 at P1.2

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

// 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; // Direct pin as input

P1REN |= BUT1; // Enable built-in resistor

P1OUT |= BUT1; // Set resistor as pull-up

P1DIR &= ~BUT2; // Direct pin as input

P1REN |= BUT2; // Enable built-in resistor

P1OUT |= BUT2; // Set resistor as pull-up

// Polling the button in an infinite loop

**for**(;;) {

// Fill the if-statement below...

**if** ((P1IN & BUT2) == 0)

;

**else** **if** ((P1IN & BUT1) == 0 && (P9OUT & greenLED) == 0)

P1OUT |= redLED; // Turn red LED on

**else**

P1OUT &= ~redLED; // Turn red LED off

**if** ((P1IN & BUT1) == 0)

;

**else** **if** ((P1IN & BUT2) == 0 && (P1OUT & redLED) == 0)

P9OUT |= greenLED; // Turn green LED on

**else**

P9OUT &= ~greenLED; // Turn green LED off

}

}

**Questions:**

No questions were given.

**Part 4:**

For this part, we were given the option to create our own design. Mine was an inverted version of Part 3. All I needed to change was the check of the LEDs, which would, instead of 0, would be itself. Also I changed the initial condition of the LEDs to be set as on. This caused me to change the toggling a slight bit to account for this change.

// CUSTOMIZE TO YOUR OWN LIKING

// Turning off the red LED while button S1 is pushed

// and turning off the green LED while button S2 is pushed

**#include** <msp430fr6989.h>

**#include** <stdio.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 S2 at P1.2

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

// Configure and initialize LEDs

P1DIR |= redLED; // Direct pin as output

P9DIR |= greenLED; // Direct pin as output

P1OUT |= redLED; // Turn LED On

P9OUT |= greenLED; // Turn LED On

// Configure buttons

P1DIR &= ~BUT1; // Direct pin as input

P1REN |= BUT1; // Enable built-in resistor

P1OUT |= BUT1; // Set resistor as pull-up

P1DIR &= ~BUT2; // Direct pin as input

P1REN |= BUT2; // Enable built-in resistor

P1OUT |= BUT2; // Set resistor as pull-up

// Polling the button in an infinite loop

**for**(;;) {

// Fill the if-statement below...

**if** ((P1IN & BUT2) == 0)

;

**else** **if** ((P1IN & BUT1) == 0 && (P9OUT & greenLED) == greenLED)

P1OUT &= ~redLED; // Turn red LED Off

**else**

P1OUT |= redLED; // Turn red LED On

**if** ((P1IN & BUT1) == 0)

;

**else** **if** ((P1IN & BUT2) == 0 && (P1OUT & redLED) == redLED)

P9OUT &= ~greenLED; // Turn green LED Off

**else**

P9OUT |= greenLED; // Turn green LED On

}

}

**Questions:**

No questions were delivered.

**Student Q&A:**

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 would be used to activate something else. Like for instance, within this lab it would have been LEDs.**
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.  
   **This if-else statement is incorrect because this would never equal 1. This would only either equal BIT3 or 0 due to you comparing xxxx xxxx (data which is unknown) and 0000 1000 (BIT3) which will only give you two solutions: 0000 1000 or 0000 0000. To correct this code he would have to change this to:** if((Data & BIT3) == BIT3)
3. Comment on the codes’ power-efficiency if the device is battery operated. Is reading the button via polling power efficient?  
   **The codes’ power-efficiency if the device is battery operated would not be optimal. Reading the button via polling is not power efficient because you have a constant for loop that is on rechecking itself to see whether the environment has changed or not. After a while, your battery will die of juice using this method.**

**Conclusion:**

I learned a lot within this lab. Although, I would have to say most of the learning gained was within Part 1 of the lab. Once you get the grasp of how to operate the buttons, most everything else can be done through the recent knowledge gained from Lab 1.