**9/12/2020**

Joshua Frazer

Lab 1

Flashing the LEDs

**Introduction:**

Within this lab I was instructed to do various tasks that dealt with the green and red LED on the microcontroller. In Part 1, I get introduced to all the documentation of the microcontroller. Proceeding on to Part 2, I started with the red LED. I learned how to turn on/off the LED and how to get it to continuously toggle. With Part 3, I realized that delay loops could be used in place of a 32-bit unsigned integer or vice versa. Part 4, I replicated what I had learned previously for the green LED. For the final part, I was able to grasp how to change the delay individually for each LED.

**Part 1:**

Documentation was introduced.

**Questions:**

No questions were given.

**Part 2:**

All code was given in this part. Nothing needed to be solved or changed.

// Code that flashes the red LED

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LED at P1.0

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

**volatile** **unsigned** **int** i;

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

PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-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

}

}

**Questions:**

* Run in debug mode and test the reset button (the third button on the board); does it work?
  + No.
* Run in normal mode and test the reset button; does it work?
  + Yes.
* Disconnect the board from the computer and plug it back; does the code resume running?
  + Yes.

**Part 3:**

This part was very simple. All that was necessary was to add variable that is a 32-bit unsigned integer, which I could comment out and switch back to the original 16-bit unsigned int. That in which you can see from my code the commented out i and j as they are used for a back-to-back delay loop. The back-to-back delay loop is also commented out as I used the 32-bit unsigned integer to demo my code, but the delay for both are equal.

// Code that flashes the red LED

**#include** <msp430fr6989.h>

**#include** <stdint.h>

**#define** redLED BIT0 // Red LED at P1.0

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

// volatile unsigned int i, j; // Variables for nested delay loop

**volatile** uint32\_t i; // unsigned int 32-bit type

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

PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode

P1DIR |= redLED; // Direct pin as output

P1OUT &= ~redLED; // Turn LED Off

**for**(;;) {

// Nested delay loop

// for(i=0; i<40000; i++) {

// for(j=0; j<3; j++) {}}

// P1OUT ^= redLED; // Toggle the LED

// Delay loop

**for**(i=0; i<120000; i++) {}

P1OUT ^= redLED; // Toggle the LED

}

}

**Questions:**

No questions delivered.

**Part 4:**

This code was very easy to solve due to it being a very similar code needed for the green LED within Part 2. All that was needed was to define the bit (BIT7), direct its pin as output, and set the green LED as on (Opposite of the red LED). Then I toggled the green LED with red LED in the delay loop.

// Code that flashes the red LED

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

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

**volatile** **unsigned** **int** i;

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

PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode

P1DIR |= redLED; // Direct pin as output

P1OUT &= ~redLED; // Turn redLED Off

P9DIR |= greenLED; // Direct pin as output

P9OUT |= greenLED; // Turn greenLED on

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

**for**(;;) {

// Delay loop

**for**(i=0; i<20000; i++) {}

P1OUT ^= redLED; // Toggle the redLED

P9OUT ^= greenLED; // Toggle the greenLED

}

}

**Questions:**

No questions delivered.

**Part 5:**

This code was complicated at first but once you get it, you get it. All that was necessary for this code was to set both LEDs to off and put two delay loops, instead of one within the infinite loop. The first for loop will toggle the red LED and the second for loop will toggle the green and red LED. Thus making the red led two times faster than the green LED.

// Code that flashes the red LED

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Red LED at P9.7

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

**volatile** **unsigned** **int** i;

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

PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode

P1DIR |= redLED; // Direct pin as output

P1OUT &= ~redLED; // Turn redLED Off

P9DIR |= greenLED; // Direct pin as output

P9OUT &= ~greenLED; // Turn greenLED Off

**for**(;;) {

// Delay loop

**for**(i=0; i<10000; i++) {}

P1OUT ^= redLED; // Toggle the redLED

**for**(i=0; i<20000; i++) {}

P9OUT ^= greenLED; // Toggle the greenLED

P1OUT ^= redLED; // Toggle the redLED

}

}

**Questions:**

No questions delivered.

**Student Q&A:**

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 delays?  
   **The effect on the battery life with delay loops is substantial. With the knowledge that I know now, I do not think it’s a good way of generating delays if it is battery operated, but if it isn’t, I don’t see why not.**
2. The MSP430 CPU runs on a clock that can be slowed down or sped up; what happens to the delay generated by the delay loop if the clock driving the CPU speeds up or slows down?  
   **If the clock driving the CPU speeds up, you would have more clocks per cycle. If it slows down, you will have less clocks per cycle. This means you would have a fast delay loop with faster clock speeds or a slow delay loop with slower clock speeds.**
3. How does the code run in the debug mode? Is the microcontroller running as an independent computer?  
   **The code run in debug mode is running depending on what you have going on in debug. For instance, if you set a breakpoint on the toggling of an LED, Code composer would stop it on that toggle. In this case, the microcontroller does not run as an independent computer.**
4. How does the code run in the normal mode? Is the microcontroller running as an independent computer?  
   **In normal mode the code runs as it was programmed. Meaning, in the case of this lab, that it will blink forever until the power is removed. The microcontroller is running as an independent computer.**
5. In which mode does the reset button work?  
   **The reset button only works in normal mode.**
6. What is the data type uint16\_t? What about int16\_t? Are these standard C syntax?  
   **The data type of uint16\_t is 16 bit unsigned integer. For int16\_t, it is a 16-bit integer. Yes, this is standard C syntax.**

**Conclusion:**

All in all, this lab was very informative. I learned how to toggle LEDs, turn each one on/off, create delay loops, create nested delay loops (Back-to-back delay loop), and how to work with the bits on the microcontroller. I would say all parts of this lab were significant.