ECE2049 C-Term 2021
Lab #0 -- Sign-off Sheet

(***This sheet MUST be attached to Lab Report!***)

Report due: **Tuesday 02/10/21**

Student 1: Jack C. Bergin **ECE mailbox:** _____

Student 2: N/A **ECE mailbox:** _____

Board #: -

BOTH PARTNERS MUST BE PRESENT AT SIGN-OFF!

Task Max points	***Max points***	***TA's assessment***
Build and demonstrate blink.c tutorial	10	
Download, build and demonstrate MSP430 development board demo	10	
Demonstrate modified demo project	10	
Answer to TA Questions	5	Student 1 Student 2
Report (Include answers to all questions and code submitted on-line)	15	
Total points	**50**	

****BOTH students MUST be present for ALL sign-offs!***

***Note*: Be sure to include your original sign-off sheet with your lab report!**

Introduction:

In this lab the primary objective is to set up our base circuit configuration, get situated with the CCS IDE, run our first couple of test programs, and then from there create a modified program. To begin the lab I attached the MSP430F5529 board (now referred to as MSP430 for short) to the LCD and wired the number pad and the piezo buzzer according to the image instructions. The set up for the LCD to the MSP430 board was a simple snap one on top of the other. To connect the piezo buzzer, a wire connecting to ground was attached to the MSP430F5529 and a second wire was connected to P3.5. To connect the number pad to the MSP430F5529. It should be noted that this basic set up will be used throughout the class and that getting all of the pins connected properly is essential to this lab's success as well as future ones.

Discussion and Results:

Blink Demo (Part I)

In this portion of the lab we were to download a completed program, “blink.c” off of the ECE 2049 canvas page to run as a test on our MSP430 boards. Once running the program a red LED on the MSP430 board would blink on and off utilizing a SW delay for the pause in between “blinks”. When breaking down the code the LED on the board was being turned on when P1.0 (the pinout on the board controlling the on board LED) was being XORed with BIT0 (defined as 0x0001). This was then executed again in the loop shutting the LED off. P1OUT during the on state of the LED holds the value of 1 when on and when off changes to the value of P1OUT to 0. In the code there were two other logic statements commented out for testing to see if they would also properly operate the LED. The first commented line was using the &~ operator and was not effective in turning the LED on and off. The second commented line used the |= comparator and

was also ineffective in turning the LED on and off. This means that both of these lines are ineffective in toggling the P1OUT value between 1 and 0.

LCD Demo (Part II)

In the second portion of this lab I was given a second program to run, breakdown, and then modify based on one assigned aspect and then personal preference. When the program was run for the first time it displayed three lines of text all centered in the LCD and then a centered character output of numbers from keypad input. Keypad input would also trigger the piezo buzzer when the star key (‘*’) on the keyboard was pressed and brought to a stop when the pound key (‘#’) was pressed. Upon closer inspection, the code that stores the values of the keypad is returning ASCII values, not integer values. Say the number 7 is pressed on the keypad. Rather than storing the integer value of 7 in the keypad it stores a value of 55 or 0x37h.

When the keys for the piezo buzzer were pressed, they would go from “main.c” to the program titled “peripherals.c” where there are state methods titled “BuzzerOn” and “BuzzerOff”. These methods were used to do exactly what they are titled. Through the usage of several logic gates, the buzzer is able to be turned on from “BuzzerOn” and off from “BuzzerOff”. These methods defined in “peripherals” are called in “main.c” and executed within an infinite while loop.

After messing around with mostly the hardware I then began to modify the lab through creating a char array storing my first and last name along with a couple other modifications. When creating my name array I had to change the size from 14 to 11 to match the character count in the array. My name is at a length of 11 characters from first to last and if 14 was the length then the program would still work but 3 characters would not be defined. If my name was

bigger than the determined length of 14, the program would throw an error. A NULL terminator is not needed because the function used to print the character array will treat the array as a string.

To integrate my name into the LCD screen I looked at the method Graphics_drawStringCentered. The important values to look for in this method were the strings in the second position of the Graphics_drawStringCentered method input and then the numbers in the fourth and fifth position of the method input. The second position string value is for what is going to be displayed on the LCD screen. The fourth position number value was for horizontal positioning of the string and the fifth position number value for vertical positioning of the string. I then created a new line in the code that took my name stored in the char array and printed it below the three string line outputs on the LCD using the Graphics_drawStringCentered method. I further modified the overall formatting of the program to print everything to the far left of the LCD screen and put another Graphics_drawStringCentered method in to display “Keypad: ” in front of the keypad output on the LCD.

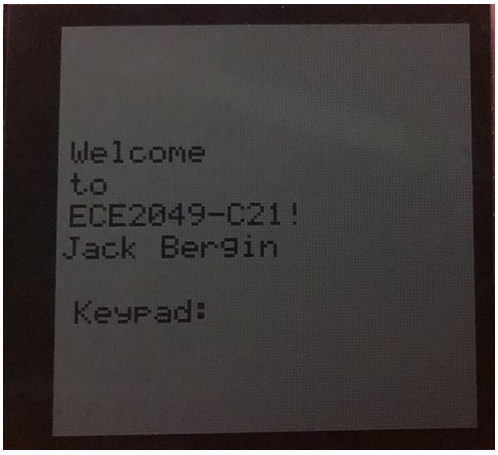

Figure 1. Modified screen output on the LCD.

Once finished with my modification I began a more indepth walk through of the code to complete the lab. I went to the top of the page and noticed a few variables not really utilized in

the LCD code. The variables were as follows; a float variable called “a_flt”, an integer called “test”, a long called “X”, and a char variable called “myGrade”. An analysis of these variable types would show differences in storage size for each value. The float is able to hold a 32 bit maximum of data, a long also being able to hold 32 bits, an integer being able to hold 16 bits, and a char being able to hold a maximum of 8 bits. The value stored in “myGrade” is the a character but upon further examination of the code showed the operation:

```
a_flt = a_flt*test;
X = test+X;
test = test-myGrade;
```

This operation will work because while “myGrade” is storing a character, it is transcribed as an ASCII value that can be manipulated by numerical values to change the character value. In the case of the code above it does the following with the test value being in bold below:

test = 0x0600h - 65 = 1536 - 65 = **1471**

Summary and Conclusions:

Lab 0 was used as a setup tutorial for the CCS IDE, a very basic introduction into the C language, and an introduction into how the C code communicates with the MSP430. The first program was used to show how different logic can affect the outcome of a program on the software side which directly impacts the hardware demonstration of the code working. The second program was utilized to expose the student to more C code where they were then able to modify it to their liking and gain a better understanding of the program and its relationship to the hardware.