16/20

8/10 cover page through system architecture
8/10 component design to end

Oscilloscope

Signal Generator

Mixed Signal Oscilloscope

Mixed Signal Generator

Project 2 - Function Generator

Students: Zach Bunce & Garrett Maxon
Class: CPE 329-03
Quarter: Spring 2018
Date Submitted: 5/4/2108
Professor: Gerfen, Jeffrey

Secret Sauce

An LCD has been added that provides real-time information with respect to the wave being outputted. The LCD shows the the current wave type, frequency, and the duty cycle.

Purpose

The purpose of the project is to design a function generator using the MSP432 microcontroller. The microcontroller is connected to an external DAC the takes in a DAC value number and outputs an analog waveform. The DAC uses an SPI interface to communicate. The function generator can generate sawtooth waveform with variable frequency, a square wave with variable duty cycle, and a sinusoidal waveform with variable frequency. The different outputs can all be selected using the keypad.

System Requirements

- The function generator shall use a microcontroller and an external DAC
- The external DAC hall have an SPI interface and a minimum of 8 bits of precision
- The function generator should be able to produce:
    - A square wave with various duty cycle
    - A sinusoidal waveform
    - A sawtooth waveform
- All waveforms shall be DC-biased around Vdd/2 with Vdd set to 3.3V
- All waveforms shall have adjustable frequencies:
    - 100Hz, 200Hz, 300Hz, 400Hz, 500Hz
    - The frequency of the waveforms should be within 5% of these frequencies.
    - The function generator shall have an output resolution of at least 70% of the maximum theoretical value
    - The output rate (points/sec) shall not change with waveform frequency
- Upon power-up, the function generator shall display a 100 Hz square wave with 50% DC
- The keypad buttons 1-5 shall set the waveform frequency in 100 Hz increments
- The keypad buttons 7, 8, and 9 shall set the output waveform to Square, Sine, and Sawtooth
- The keypad buttons ✱, 0, and # shall change the duty cycle of the square wave
    - The ✱ shall decrease the duty cycle by 10% down to a minimum of 10%
    - The # shall increase the duty cycle by 10% up to a maximum of 90%
    - The 0 key shall reset the duty cycle to 50%
    - The keys ✱, 0, and # shall not affect the sin or sawtooth waveforms
- You must use the on-chip timer to generate interrupts that generate timing events for your function generator. You may not use software delays to generate timing events for outputting to the DAC.

-1
Also include reqs of YOUR OWN system, including what the LCD will display

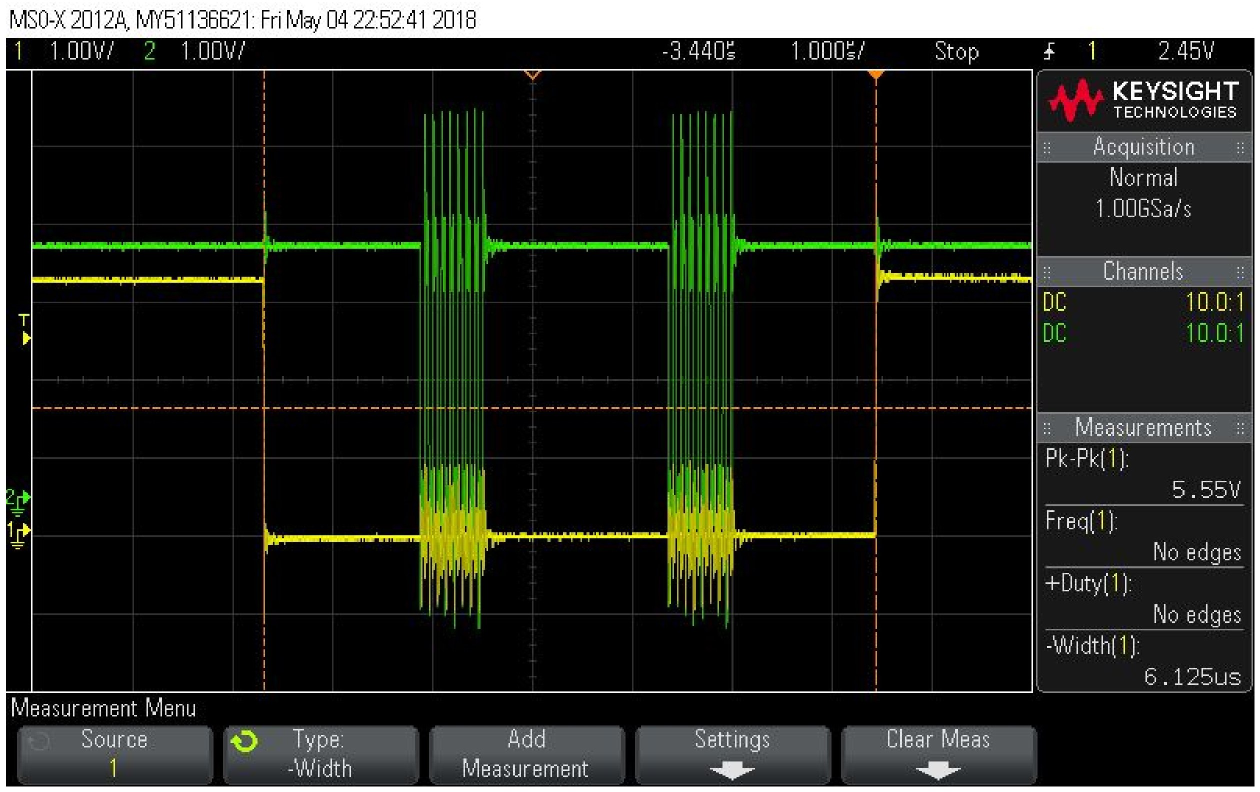


Figure 1: DAC data transmission timing

Save all screen shots and calculations for system integration section

Youtube Video Demonstration

https://youtu.be/sAf2lIaFtF8

Error Calculations

Maximum Theoretical DAC refresh frequency -163 kHz

Wave	**100 Hz**	**200 Hz**	**300 Hz**	**400 Hz**	**500 Hz**
Sine	.4%	.65%	.4%	.21%	.3%
Sawtooth	.7%	.5%	.3%	.15%	.13%
Square	.7%	1%	.6%	.4%	.5%

System Specifications

Table 1: System Specifications

Hardware	LCD NHD-0216HZ-FSW-FBW-33V3C
Dimensions	14mm/36.7mm/65.5mm H/W/L
Character Resolution	2x16 R/C
Operating Voltage	3.3V
Startup Time	45.6316 ms
Response Time	474 µs
Hardware	Texas Instruments MSP432 Microprocessor
I/O Pins Used	14
Operating Voltage	3.3V
Clock Frequency	48MHz
Hardware	Keypad
Key Count	12
Dimensions	12mm/51mm/70mm H/W/L
Hardware	4921 SPI DAC
Number of Bits	12
Operating Voltage	2.7V - 5.5V
Clock Frequency	<20MHz
Response Time	4.5ns
Hardware	Overall System
Dimensions	4.5”/6.5”/6” H/W/L

System Architecture

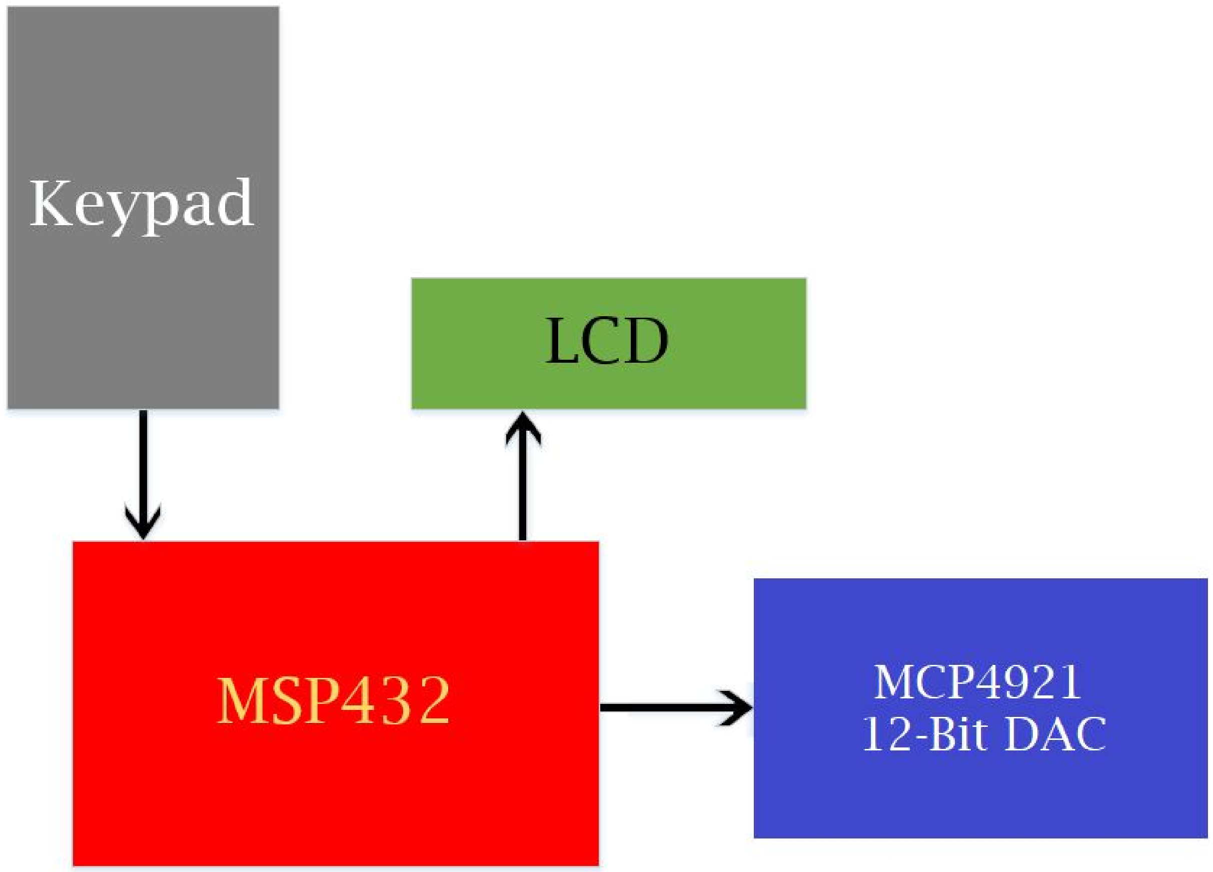


Figure 2: High Level System Block Diagram

-1
Missing high level software flowchart

Component Design

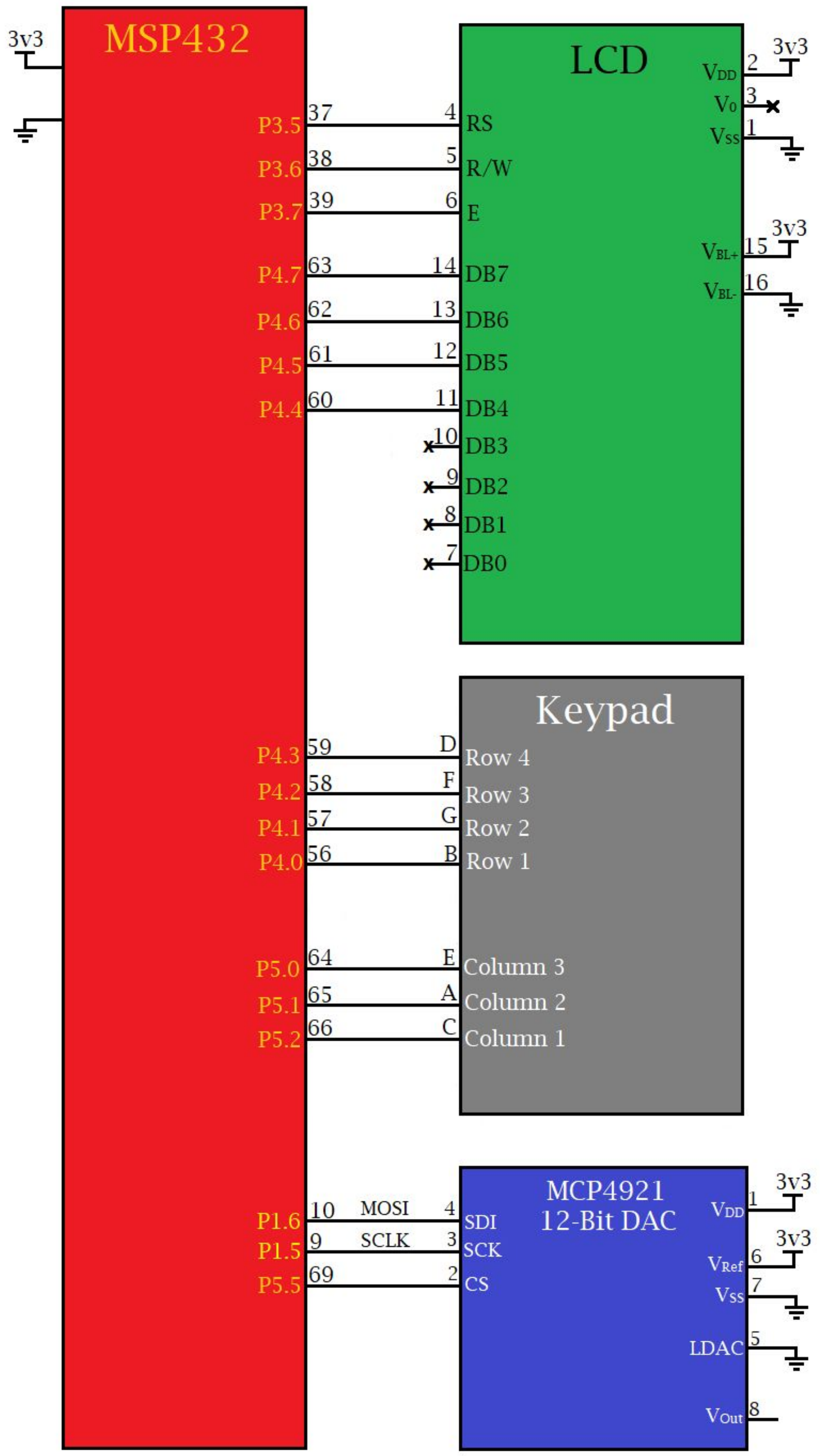


Figure 3: Schematic diagram of complete system including MSP432, LCD, keypad, and DAC

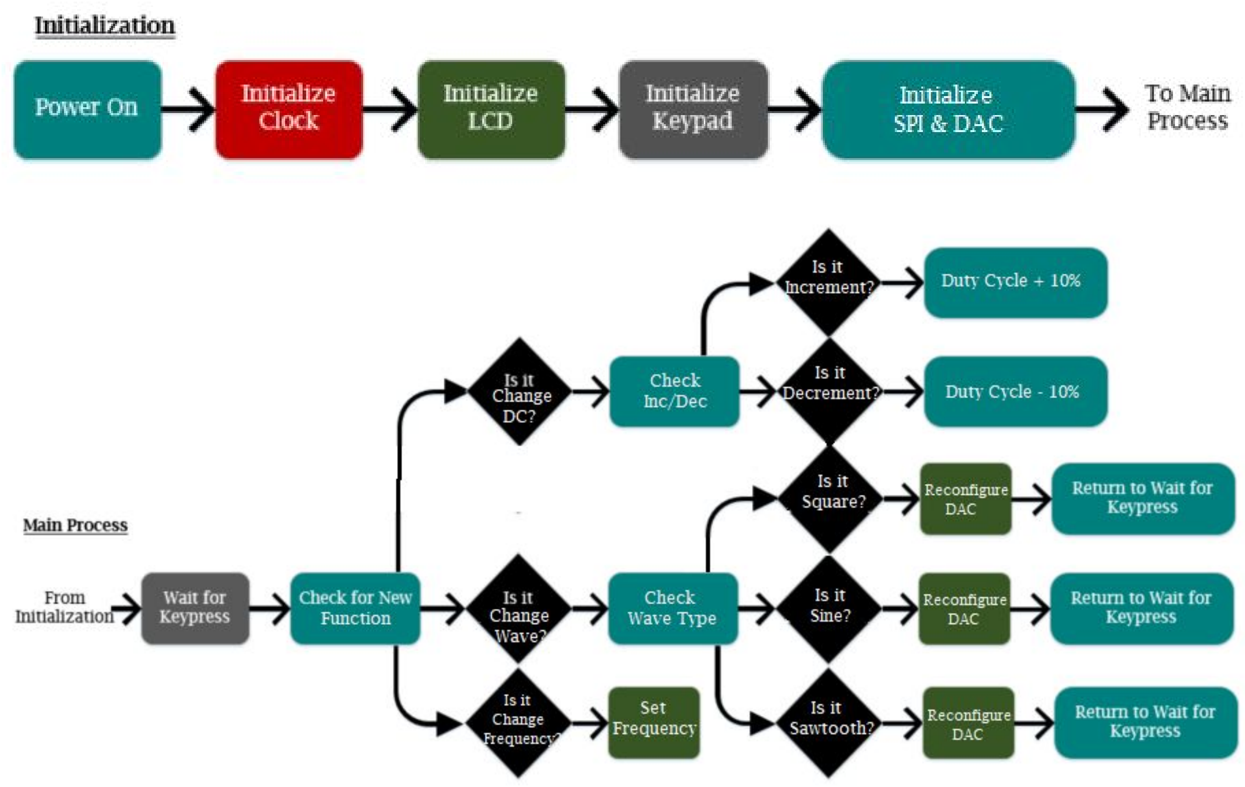


Figure 4: Process Flow Diagram

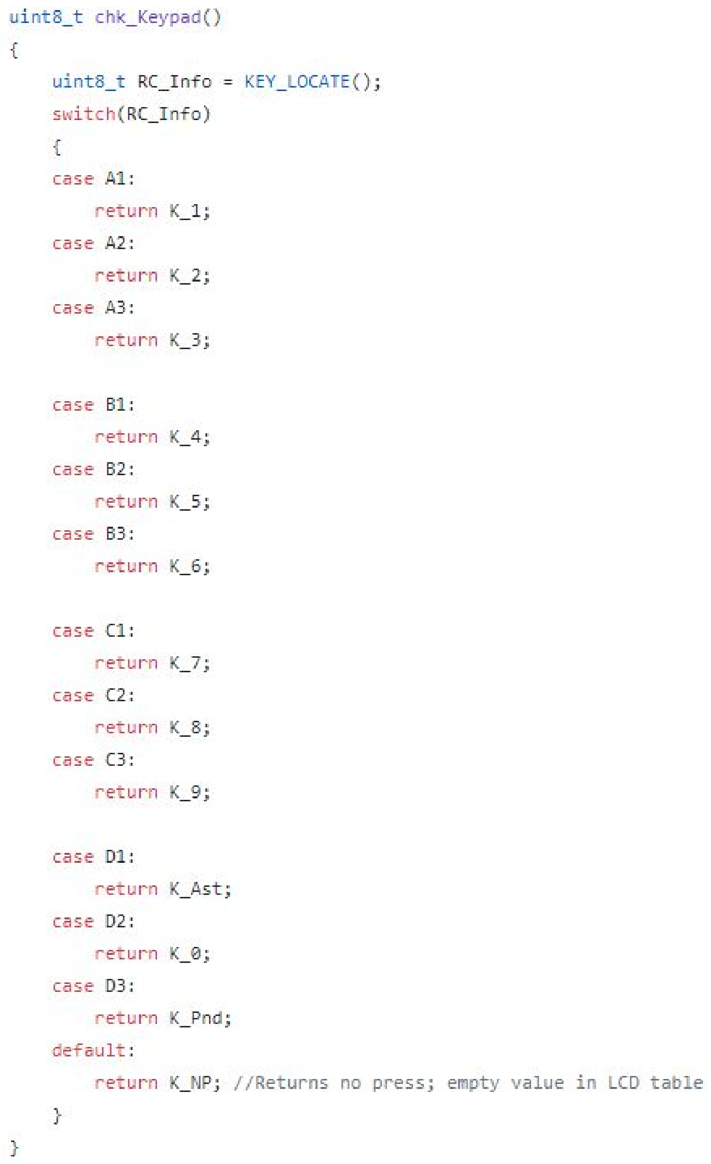
Bill Of Materials

Table 2: Bill of Materials

#	Part Description	Part Number	Distributor	Quantity	Price
1	MSP432 Launchpad	MSP-EXP432P401R	Digikey	1	$13.49
2	LCD Module	NHD-0216HZ-FSW-FBW-33V3 C	Digikey	1	$11.60
3	12 Key Switch Keypad	COM-08653-ND	Digi-Key	1	$3.95
4	6" M/M & M/F Jumpers (Strip of 10)	DZ-DBX-01	Amazon	1	$6.98
5	25 Header Pin Strip	78511-436HLF	Digi-Key	1	$1.38
6	Breadboard 830 Point (3 pieces)	EL-CP-003	Amazon	1	$9.99
7	4921 SPI DAC	MCP4921-E/P	Digi-Key	1	$2.03
				TOTAL	**$49.38**

System Integration

In order to construct the complete system, the individual peripheral libraries were first constructed and tested separately over the course of Assignments 5-7. As Assignment 6 required the integration of interrupts and timers, both of which were done in previous assignments and verified to be working. This meant that the wave generating capabilities were already confirmed and just needed to be adjusted to create new kinds of waves, specifically a sawtooth and sine wave. Thus the design process started with calculating lookup table for the sine wave and the sawtooth wave. Once that was done everything that was needed was available and just needed to be interacted together. The first thing that was done was using the get_Key function which is a get function for chk_Keypad(). (See ***Figure 5***)

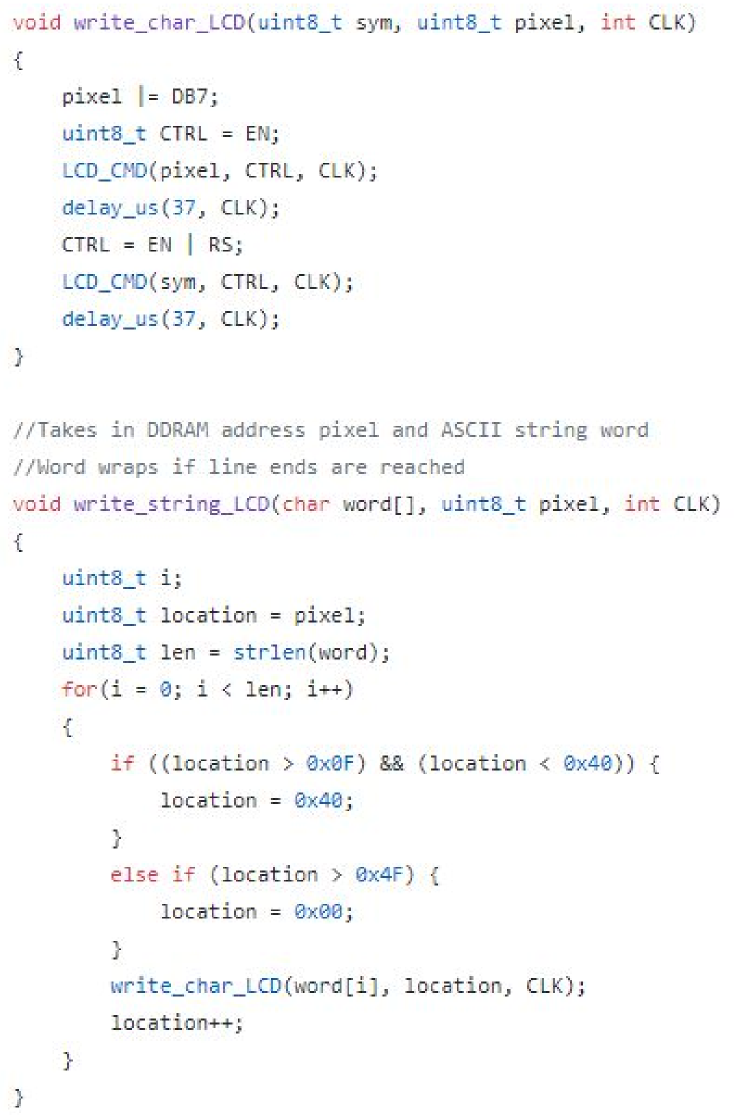


```
uint8_t chk_Keypad()
{
    uint8_t RC_Info = KEY_LOCATE();
    switch(RC_Info)
    {
    case A1:
        return K_1;
    case A2:
        return K_2;
    case A3:
        return K_3;

    case B1:
        return K_4;
    case B2:
        return K_5;
    case B3:
        return K_6;

    case C1:
        return K_7;
    case C2:
        return K_8;
    case C3:
        return K_9;

    case D1:
        return K_Ast;
    case D2:
        return K_0;
    case D3:
        return K_Pnd;
    default:
        return K_NP; //Returns no press; empty value in LCD table
    }
}
```

Figure 5 - chk_Keypad() function used for get_Key() function

?
Why does this paragraph start like this?

from the previous project. With this information a switch-case statement was made that changes certain variables depending on what was pressed. For example, if 1-5 is pressed then the variable the stores the frequency is changed to match that of the key press. Then the write_string_LCD() (see figure 6) function grabs the new variable and changes the LCD accordingly.



```
void write_char_LCD(uint8_t sym, uint8_t pixel, int CLK)
{
    pixel |= DB7;
    uint8_t CTRL = EN;
    LCD_CMD(pixel, CTRL, CLK);
    delay_us(37, CLK);
    CTRL = EN | RS;
    LCD_CMD(sym, CTRL, CLK);
    delay_us(37, CLK);
}

//Takes in DDRAM address pixel and ASCII string word
//Word wraps if line ends are reached
void write_string_LCD(char word[], uint8_t pixel, int CLK)
{
    uint8_t i;
    uint8_t location = pixel;
    uint8_t len = strlen(word);
    for(i = 0; i < len; i++)
    {
        if ((location > 0x0F) && (location < 0x40)) {
            location = 0x40;
        }
        else if (location > 0x4F) {
            location = 0x00;
        }
        write_char_LCD(word[i], location, CLK);
        location++;
    }
}
```

Figure 6: write_char_LCD() and write_string_LCD() functions

The DAC was integrated from assignment 6. The DAC uses the SPI communication protocol to interface between the board and the chip itself.

-1
Discuss hardware development as well.

Conclusion

This project was a good combination of various previous assignments that have been worked on. It combined the keypad, interrupts, DAC, and the LCD into one whole project creating a working function generator. During the overall process of interfacing the DAC and interrupts throughout the various assignments, as well as constructing the other libraries used such as the delays, many important and not immediately clear details were found that were necessary to ensure proper integration of each peripheral. For example how to handled the DAC input code, and translated that to the correct voltages. To solve this a supplied formula was used to translate a binary number to an output voltage. For example, giving the DAC input code of 4096 will give an output of Vref (3.3V). The next issue was translating a voltage which is a discrete value to enough points to create a continuous waveform. This issue was fixed by using the timers to determine the time needed between points and then triggering interrupts when the next point was reached then repeating many times a second to create a contain. The sine wave was different from every wave in the sense that it needed a lookup table to be calculated. This process was much harder than expected since the values needed to be calculated in excel and the format was not acceptable. To fix this extensive excel formatting was used to get the table in the form of "0x0000". Once the tables were created they were then pasted directly into the code files so they could be indexed into later. To use the table values were utilized just by indexing into it since it was stored as an array and then incrementing by a certain value to achieve the correct frequency. This method produces a very accurate reproduction and takes away almost all the error.

This project ensured a thorough understanding of the ways in which the interrupt system could be managed and communicated with, as error free ISR behavior required optimized ISR code.

Improvements that could be made include additional wave types and increased resolution on the high frequency waveforms.

The complete system designed helped to provide further experience in large scale, interconnected system design.

Appendix A: C Code

Main C Function

```
/**
 * main.c
 *
 * Sets up SPI and generates specified wave
 *
 * Date: April 2 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include <stdint.h>
#include <math.h>
#include <string.h>
#include "set_DCO.h"
#include "delays.h"
#include "keypad.h"
#include "SPI.h"
#include "DAC.h"
#include "LCD.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;    // stop watchdog timer

    char labels[] = "Type:     DC:  %Freq:    Hz"; //Initializes LCD variables
    int incFlag = 0;
    char freq_Xfr[3] = "100";
    char waveType[3] = "SQR";
    char duty_Xfr[2] = "50";

    uint8_t key = K_NP;                       //Initializes key press variable
    char waveT = 1;                           //Default values for square wave used upon startup
    int frequency = 100;
    int  duty = 50;

    int CLK = 120;
    set_DCO(CLK);                             //Sets clock to 12MHz
    LCD_INIT(CLK);                            //Initializes LCD w/ proper CLK
    KEYPAD_INIT();                            //Initializes keypad
    SPI_INIT();                               //Initializes SPI comms

    write_string_LCD(labels, 0, CLK);         //Writes the default starting state to the LCD

    FG_INIT();                                //Initializes function generator

    while(1)
    {
        switch (key)
```

```
{
case K_1:
    frequency = 100;                //Sets the frequency to 100 and prints it out
    strcpy(freq_Xfr, "100");
    break;
case K_2:
    frequency = 200;                //Sets the frequency to 200 and prints it out
    strcpy(freq_Xfr, "200");
    break;
case K_3:
    frequency = 300;                //Sets the frequency to 300 and prints it out
    strcpy(freq_Xfr, "300");
    break;
case K_4:
    frequency = 400;                //Sets the frequency to 400 and prints it out
    strcpy(freq_Xfr, "400");
    break;
case K_5:
    frequency = 500;                //Sets the frequency to 500 and prints it out
    strcpy(freq_Xfr, "500");
    break;
case K_7:
    waveT = 1;                      //Set wave type to square wave and prints it out
    duty = 50;
    strcpy(duty_Xfr, "50");
    strcpy(waveType, "SQR");
    break;
case K_8:
    waveT = 3;               //Set wave type to sine wave and prints it out
    strcpy(waveType, "SIN");
    break;
case K_9:
    waveT = 4;               //Set wave type to sawtooth wave and prints it out
    strcpy(waveType, "SAW");
    break;
case K_Ast:
    if (duty > 10) {         //Subtracts 10 to the duty cycle everytime * is pressed
        duty -= 10;
    }
    switch(duty) {
    case 10:
        strcpy(duty_Xfr, "10");     //Prints out "10" if duty cycle is 10
        break;
    case 20:
        strcpy(duty_Xfr, "20");     //Prints out "20" if duty cycle is 20
        break;
    case 30:
        strcpy(duty_Xfr, "30");     //Prints out "30" if duty cycle is 30
        break;
    case 40:
        strcpy(duty_Xfr, "40");     //Prints out "40" if duty cycle is 40
        break;
    case 50:
        strcpy(duty_Xfr, "50");     //Prints out "50" if duty cycle is 50
        break;
    case 60:
        strcpy(duty_Xfr, "60");     //Prints out "60" if duty cycle is 60
        break;
    case 70:
        strcpy(duty_Xfr, "70");     //Prints out "70" if duty cycle is 70
        break;
    case 80:
        strcpy(duty_Xfr, "80");     //Prints out "80" if duty cycle is 80
        break;
    case 90:
        strcpy(duty_Xfr, "90");     //Prints out "90" if duty cycle is 90
        break;
    }
    break;
case K_Pnd:
    if (duty < 90) {         //Adds 10 to the duty cycle every time # is pressed
        duty += 10;
    }
    switch(duty) {
    case 10:
        strcpy(duty_Xfr, "10");            //Prints out "10" if duty cycle is 10
        break;
    case 20:
        strcpy(duty_Xfr, "20");            //Prints out "20" if duty cycle is 20
        break;
    case 30:
        strcpy(duty_Xfr, "30");            //Prints out "30" if duty cycle is 30
        break;
    case 40:
        strcpy(duty_Xfr, "40");            //Prints out "40" if duty cycle is 40
        break;
    case 50:
        strcpy(duty_Xfr, "50");            //Prints out "50" if duty cycle is 50
        break;
    case 60:
        strcpy(duty_Xfr, "60");            //Prints out "60" if duty cycle is 60
        break;
    case 70:
        strcpy(duty_Xfr, "70");            //Prints out "70" if duty cycle is 70
        break;
    case 80:
        strcpy(duty_Xfr, "80");            //Prints out "80" if duty cycle is 80
        break;
    case 90:
        strcpy(duty_Xfr, "90");            //Prints out "90" if duty cycle is 90
```

```
                break;
            }
            break;
        case K_0:
            duty = 50;                      //Sets duty cycle back to 50% if 0 is pressed
            strcpy(duty_Xfr, "50");
            break;
        }

        if (waveT == 1)      //If square wave then print duty cycle, freq, and waveform
            write_string_LCD(waveType, 0x06, CLK);
            write_string_LCD(duty_Xfr, 0x0D, CLK);
            write_string_LCD(freq_Xfr, 0x46, CLK);
        }
        else {        //Print frequency, blank duty, and waveform otherwise
            strcpy(duty_Xfr, "  ");
            write_string_LCD(waveType, 0x06, CLK);
            write_string_LCD(duty_Xfr, 0x0D, CLK);
            write_string_LCD(freq_Xfr, 0x46, CLK);
        }

        key = K_NP;                                //If no press delay
        delay_ms(200, CLK);

        while(key == K_NP) {                       //If no press print same thing
            makeWave(waveT, frequency, duty, CLK);
            key = get_Key();
        }
    }
}
```

DAC Library

```
/**
 * DAC.c
 *
 * Contains functions for running the DAC
 * Allows for DC voltages and various wave types to be automatically produced
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include <math.h>
#include <stdint.h>
#include "SPI.h"
#include "DAC.h"
#include "delays.h"
#include "LUT.h"

static int z = 0;       //Timer division variable
static int s = 0;       //Sin value index variable
static int t = 0;       //Sawtooth value index variable
static int c = 0;
static int waveType = 1;    //Wave property variables for ISR
static int freq_idx = 1;
static int DC = 50;

static int sqrDel = 600;
static int sinDel = 600;
static int sawDel = 600;
static int divs = 200;

uint16_t sqrVal10_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal20_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal30_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal40_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal50_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal60_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal70_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal80_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sqrVal90_DN[200] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,
0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF,0x0FFF
};

uint16_t sinVal_DN[200] = {
0x0800,0x0840,0x0880,0x08C0,0x0900,0x0940,0x097F,0x09BE,0x09FD,0x0A3B,
0x0A78,0x0AB5,0x0AF1,0x0B2D,0x0B67,0x0BA1,0x0BDA,0x0C12,0x0C49,0x0C7F,
0x0CB3,0x0CE7,0x0D19,0x0D4A,0x0D79,0x0DA8,0x0DD4,0x0E00,0x0E2A,0x0E52,
0x0E78,0x0E9D,0x0EC1,0x0EE2,0x0F02,0x0F20,0x0F3D,0x0F57,0x0F70,0x0F86,
0x0F9B,0x0FAE,0x0FBF,0x0FCE,0x0FDB,0x0FE6,0x0FEF,0x0FF6,0x0FFB,0x0FFE,
0x0FFF,0x0FFE,0x0FFB,0x0FF6,0x0FEF,0x0FE6,0x0FDB,0x0FCE,0x0FBF,0x0FAE,
0x0F9B,0x0F86,0x0F70,0x0F57,0x0F3D,0x0F20,0x0F02,0x0EE2,0x0EC1,0x0E9D,
0x0E78,0x0E52,0x0E2A,0x0E00,0x0DD4,0x0DA8,0x0D79,0x0D4A,0x0D19,0x0CE7,
0x0CB3,0x0C7F,0x0C49,0x0C12,0x0BDA,0x0BA1,0x0B67,0x0B2D,0x0AF1,0x0AB5,
0x0A78,0x0A3B,0x09FD,0x09BE,0x097F,0x0940,0x0900,0x08C0,0x0880,0x0840,
0x0800,0x07BF,0x077F,0x073F,0x06FF,0x06BF,0x0680,0x0641,0x0602,0x05C4,
0x0587,0x054A,0x050E,0x04D2,0x0498,0x045E,0x0425,0x03ED,0x03B6,0x0380,
0x034C,0x0318,0x02E6,0x02B5,0x0286,0x0257,0x022B,0x01FF,0x01D5,0x01AD,
0x0187,0x0162,0x013E,0x011D,0x00FD,0x00DF,0x00C2,0x00A8,0x008F,0x0079,
0x0064,0x0051,0x0040,0x0031,0x0024,0x0019,0x0010,0x0009,0x0004,0x0001,
0x0000,0x0001,0x0004,0x0009,0x0010,0x0019,0x0024,0x0031,0x0040,0x0051,
0x0064,0x0079,0x008F,0x00A8,0x00C2,0x00DF,0x00FD,0x011D,0x013E,0x0162,
0x0187,0x01AD,0x01D5,0x01FF,0x022B,0x0257,0x0286,0x02B5,0x02E6,0x0318,
0x034C,0x0380,0x03B6,0x03ED,0x0425,0x045E,0x0498,0x04D2,0x050E,0x054A,
0x0587,0x05C4,0x0602,0x0641,0x0680,0x06BF,0x06FF,0x073F,0x077F,0x07BF
};


uint16_t sawVal_DN[200] = {
0x0000,0x0014,0x0028,0x003D,0x0051,0x0066,0x007A,0x008F,0x00A3,0x00B8,
0x00CC,0x00E1,0x00F5,0x010A,0x011E,0x0133,0x0147,0x015C,0x0170,0x0185,
0x0199,0x01AE,0x01C2,0x01D7,0x01EB,0x0200,0x0214,0x0228,0x023D,0x0251,
0x0266,0x027A,0x028F,0x02A3,0x02B8,0x02CC,0x02E1,0x02F5,0x030A,0x031E,
0x0333,0x0347,0x035C,0x0370,0x0385,0x0399,0x03AE,0x03C2,0x03D7,0x03EB,
0x0400,0x0414,0x0428,0x043D,0x0451,0x0466,0x047A,0x048F,0x04A3,0x04B8,
0x04CC,0x04E1,0x04F5,0x050A,0x051E,0x0533,0x0547,0x055C,0x0570,0x0585,
0x0599,0x05AE,0x05C2,0x05D7,0x05EB,0x0600,0x0614,0x0628,0x063D,0x0651,
0x0666,0x067A,0x068F,0x06A3,0x06B8,0x06CC,0x06E1,0x06F5,0x070A,0x071E,
0x0733,0x0747,0x075C,0x0770,0x0785,0x0799,0x07AE,0x07C2,0x07D7,0x07EB,
0x0800,0x0814,0x0828,0x083D,0x0851,0x0866,0x087A,0x088F,0x08A3,0x08B8,
0x08CC,0x08E1,0x08F5,0x090A,0x091E,0x0933,0x0947,0x095C,0x0970,0x0985,
0x0999,0x09AE,0x09C2,0x09D7,0x09EB,0x0A00,0x0A14,0x0A28,0x0A3D,0x0A51,
0x0A66,0x0A7A,0x0A8F,0x0AA3,0x0AB8,0x0ACC,0x0AE1,0x0AF5,0x0B0A,0x0B1E,
0x0B33,0x0B47,0x0B5C,0x0B70,0x0B85,0x0B99,0x0BAE,0x0BC2,0x0BD7,0x0BEB,
0x0C00,0x0C14,0x0C28,0x0C3D,0x0C51,0x0C66,0x0C7A,0x0C8F,0x0CA3,0x0CB8,
0x0CCC,0x0CE1,0x0CF5,0x0D0A,0x0D1E,0x0D33,0x0D47,0x0D5C,0x0D70,0x0D85,
0x0D99,0x0DAE,0x0DC2,0x0DD7,0x0DEB,0x0E00,0x0E14,0x0E28,0x0E3D,0x0E51,
0x0E66,0x0E7A,0x0E8F,0x0EA3,0x0EB8,0x0ECC,0x0EE1,0x0EF5,0x0F0A,0x0F1E,
0x0F33,0x0F47,0x0F5C,0x0F70,0x0F85,0x0F99,0x0FAE,0x0FC2,0x0FD7,0x0FEB,
};

static uint16_t sqr_ST = 0x000;  //Square wave state variable for ISR
static int16_t DN_Point;
static int intFlag = 0;

void FG_INIT()
{
    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE;     //Enables TACCR0 interrupt
    //Runs Timer A on SMCLK and in continuous mode
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_MC__CONTINUOUS;
    //SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;        //Enables sleep on exit from ISR

    __enable_irq();                             //Enables global interrupts
    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);   //Links ISR to NVIC
    TIMER_A0->CCR[0] = 600;               //Initializes first high time count
}

void makeDC(int volt)
{
    uint16_t DN = (((4096*volt)/(Vref))*10);    //Maps the 12-bit DAC value for the desired
output voltage
    write_DAC(DN);                              //Writes value to the DAC
}

void makeWave(int waveT, int freq, int duty, int CLK)
{
    waveType = waveT;   //Links outside specified parameters to ISR globals
    freq_idx = freq / 100;
    DC = duty;

    write_DAC(DN_Point);
}

void write_DAC(uint16_t data)
{
    uint8_t up_Byte;
    uint8_t low_Byte;

    //Shifts the upper nibble into the lower, masks the 4 data bits, appends control bits
    up_Byte  =  ((data >> 8) & 0x0F) | (GAIN1 | SDOFF);
    low_Byte =   (data & 0x00FF);       //Masks bottom 8 data bits

    P5 -> OUT &= ~BIT5;                 //Lowers chip select
    sendByte_SPI(up_Byte);              //Transmits upper byte on SPI line
    sendByte_SPI(low_Byte);             //Transmits lower byte on SPI line
    P5 -> OUT |= BIT5;                  //Sets chip select
}

// Timer A0 interrupt service routine
void TA0_0_IRQHandler(void) {
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;  //Clears interrupt flag

    if (waveType == square)
    {
        c += freq_idx;
        if (c >= divs) {
            c = 0;
        }
        switch(DC)
        if (DC == 10) {
            DN_Point = sqrVal10_DN[c];
        }
        else if (DC == 20) {
            DN_Point= sqrVal20_DN[c];
        }
        else if (DC == 30) {
            DN_Point= sqrVal30_DN[c];
        }
        else if (DC == 40) {
            DN_Point= sqrVal40_DN[c];
        }
        else if (DC == 50) {
            DN_Point= sqrVal50_DN[c];
        }
        else if (DC == 60) {
            DN_Point= sqrVal60_DN[c];
        }
        else if (DC == 70) {
            DN_Point= sqrVal70_DN[c];
        }
        else if (DC == 80) {
            DN_Point= sqrVal80_DN[c];
        }
        else if (DC == 90) {
            DN_Point= sqrVal90_DN[c];
        }

        TIMER_A0->CCR[0] += sqrDel;              //Adds next offset to TACCR0
    }
    else if(waveType == sine) {
        s += freq_idx;
        if (s >= divs) {
            s = 0;
        }
        DN_Point = sinVal_DN[s];
        TIMER_A0->CCR[0] += sinDel;              //Adds next offset to TACCR0
    }
    else if(waveType == sawtooth) {
        t += freq_idx;
        if (t >= divs) {
            t = 0;
        }
        DN_Point = sawVal_DN[t];
        TIMER_A0->CCR[0] += sawDel;              //Adds next offset to TACCR0
    }
}
```

```c
/**
 * DAC.h
 *
 * Header for running the DAC
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

#ifndef DAC_H_
#define DAC_H_

#define square      1       //Wave type definitions
#define triangle    2
#define sine        3
#define sawtooth    4
#define Vref        33      //Reference voltage * 10
#define SDOFF       BIT4    //Shutdown mode OFF
#define GAIN1       BIT5    //Unity gain mode
#define HIGH        0xFFF   //Definitions for square wave states
#define LOW         0x000

#define F_100Hz     100
#define F_200Hz     200
#define F_300Hz     300
#define F_400Hz     400
#define F_500Hz     500

void write_DAC(uint16_t);
void makeDC(int);
void makeWave(int, int, int, int);
int  chk_FGFlag();
void clr_FGFlag();
void FG_INIT();

#endif /* DAC_H_ */
```

LCD Library

```
/*
 * LCD.c
 * Code file for LCD control.
 * Functions designed for external use:
 * write_char_LCD, clear_LCD, & home_LCD
 *
 * Date: April 10, 2018
 * Authors: Zach Bunce, Garret Maxon
 */

#include "msp.h"
#include "delays.h"
#include "LCD.h"
#include <string.h>

void LCD_CMD(uint8_t, uint8_t, int);
void LCD_CTRL(uint8_t, int);
void write_char_LCD(uint8_t, uint8_t, int);
void clear_LCD(int);

//Takes in DDRAM address pixel and ASCII character sym
//0x00...0x0F DDRAM Addresses
//0x40...0x4F
void write_char_LCD(uint8_t sym, uint8_t pixel, int CLK)
{
    pixel |= DB7;
    uint8_t CTRL = EN;
    LCD_CMD(pixel, CTRL, CLK);
    delay_us(37, CLK);
    CTRL = EN | RS;
    LCD_CMD(sym, CTRL, CLK);
    delay_us(37, CLK);
}

//Takes in DDRAM address pixel and ASCII string word
//Word wraps if line ends are reached
void write_string_LCD(char word[], uint8_t pixel, int CLK)
{
    uint8_t i;
    uint8_t location = pixel;
    uint8_t len = strlen(word);
    for(i = 0; i < len; i++)
    {
        if ((location > 0x0F) && (location < 0x40)) {
            location = 0x40;
        }
```

```c
        else if (location > 0x4F) {
            location = 0x00;
        }
        write_char_LCD(word[i], location, CLK);
        location++;
    }
}

//1.52 ms delay required after operation
void home_LCD(int CLK)
{
    LCD_CMD(HOME_RET, EN, CLK);
    delay_ms(2, CLK);
}

//1.52 ms delay required after operation
void clear_LCD(int CLK)
{
    LCD_CMD(DISP_CLR, EN, CLK);
    delay_ms(2, CLK);
}

//Clears the specified line of the LCD and puts the cursor on the next line
void line_clear_LCD(int line, int CLK)
{
    char blank[] = "                ";
    if (line == TOP) {
        write_string_LCD(blank, 0x00, CLK);
        home_LCD(CLK);
    }
    else if (line == BOTTOM) {
        write_string_LCD(blank, 0x40, CLK);
        write_char_LCD(0x10, 0x3F, CLK);
    }
}

//Sets up I/O register direction
//Runs through LCD setup procedure
//USE LCD_3.3V_LCD_NHD DATASHEET PROCEDURE
//Currently only works for 4-bit mode; scared of timings
void LCD_INIT(int CLK)
{
    P3 -> DIR |= RS | RW | EN;  //Sets reg directions
    P4 -> DIR |= DB7 | DB6 | DB5 | DB4; //Not using all 4 so don't be lazy

    P3 -> OUT &= ~(RS | RW | EN);
    P4 -> OUT &= ~(DB7 | DB6 | DB5 | DB4); //Sets output low
    delay_ms(40, CLK);  //Waits for safe power up
    P4 -> OUT |= 0x30;   //Sets wake up command
    delay_ms(5, CLK);
    LCD_CTRL(EN, CLK);  //Wake up #1
    P3 -> OUT &= ~EN;
    delay_us(160, CLK);

    LCD_CTRL(EN, CLK);  //Wake up #2
    P3 -> OUT &= ~EN;
    delay_us(160, CLK);

    LCD_CTRL(EN, CLK);  //Wake up #3
    P3 -> OUT &= ~EN;
    delay_us(160, CLK);

    P4 -> OUT &= ~(DB7 | DB6 | DB5 | DB4); //Sets output low
    P4 -> OUT |= 0x20;   //Guess we're awake
    LCD_CTRL(EN, CLK);
    P3 -> OUT &= ~EN;

    //The rest sets up LCD settings
    LCD_CMD(FXN_SET,    EN, CLK);
    LCD_CMD(SHIFT_SET,  EN, CLK);
    LCD_CMD(DISP_SET,   EN, CLK);
    LCD_CMD(ENTRY_SET,  EN, CLK);

    clear_LCD(CLK); //Bill Murray
}

//Sets control bits to states given in CTRL
//CTRL: Top three bits EN, RW, RS respectively
void LCD_CTRL(uint8_t CTRL, int CLK)
{
    P3 -> OUT &= ~(RS | RW | EN); //Clears RS, RW, and EN
    P3 -> OUT |= CTRL; //Sets RS, RW, and EN to desired state
    delay_us(30, CLK); //Nominal 460 ns W / 480 ns R
}

//Writes to the 8 data bits of the LCD
//Can access both data and instruction registers
//30 us delays to ensure known timings
void LCD_CMD(uint8_t CMD, uint8_t CTRL, int CLK)
{
    P4 -> OUT &= ~(DB7 | DB6 | DB5 | DB4); //Sets output low
    P4 -> OUT |= CMD & 0xF0;
    LCD_CTRL(CTRL, CLK);
    P3 -> OUT &= ~EN;   //Nibble 1
    delay_us(30, CLK);  //Nominal 10 ns
    if ((FXN_SET & DB4) == 0x00) {
        CMD = CMD << 4;
        P4 -> OUT &= ~(DB7 | DB6 | DB5 | DB4); //Sets output low
        P4 -> OUT |= CMD & 0xF0;
```

```c
        LCD_CTRL(CTRL, CLK);
        P3 -> OUT &= ~EN;   //Nibble 2
        delay_us(30, CLK);  //Nominal 10 ns
    }
    P4 -> OUT &= ~(DB7 | DB6 | DB5 | DB4);
    delay_us(80, CLK); //Nominal 730 ns
}

/*
 * LCD.h
 * Header file for LCD control.
 *
 * Date: April 10, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#ifndef LCD_H_
#define LCD_H_

#define RS          BIT5
#define RW          BIT6
#define EN          BIT7
#define DB0         BIT0
#define DB1         BIT1
#define DB2         BIT2
#define DB3         BIT3
#define DB4         BIT4
#define DB5         BIT5
#define DB6         BIT6
#define DB7         BIT7

#define BOTTOM      1       //Defines top and bottom row indicators
#define TOP         0
#define DISP_CLR    0x01
#define HOME_RET    0x02
//Change the definitions below to change initialization
#define DISP_SET    0x0F    //0x0F Disp ON, Cursor ON, Blink ON
#define FXN_SET     0x28    //0x28 4-Bit, 2 line, 5x8 font
#define SHIFT_SET   0x10    //0x10 shifts cursor or disp
#define ENTRY_SET   0x06    //0x06 -> cursor++

void LCD_INIT(int);
void LCD_CMD(uint8_t, uint8_t, int);
void write_char_LCD(uint8_t, uint8_t, int);
void write_string_LCD(char word[], uint8_t, int);
void clear_LCD(int);
void line_clear_LCD(int, int);
void home_LCD(int);

#endif /* LCD_H_ */
```

SPI Library

```
/*
 * SPI.c
 *
 * Contains functions for both SPI setup and transmission
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

//P1.5 SCLK
//P1.6 MOSI
//P1.7 MISO

#include "msp.h"
#include <stdint.h>
#include "SPI.h"

void SPI_INIT()
{
    P5 -> DIR  |= BIT5;                         //CS Output
    P5 -> OUT  |= BIT5;                         //Initializes CS high
    P1 -> SEL0 |= (BIT5 | BIT6 | BIT7);         //SPI Line pinout setup
    P1 -> SEL1 &= ~(BIT5 | BIT6);

    EUSCI_B0 -> CTLW0 |= EUSCI_B_CTLW0_SWRST;   //Reset
    EUSCI_B0 -> CTLW0  = EUSCI_B_CTLW0_SWRST |
                         EUSCI_B_CTLW0_MST   |
                         EUSCI_B_CTLW0_SYNC  |
                         EUSCI_B_CTLW0_CKPL  |
                         EUSCI_B_CTLW0_MSB;

    EUSCI_B0 -> CTLW0 |= EUSCI_B_CTLW0_SSEL__SMCLK; //SMCLK

    EUSCI_B0 -> BRW = 0x01;                     //BRW = 1; SDICLK = SMCLK
    EUSCI_B0 -> CTLW0 &= ~EUSCI_B_CTLW0_SWRST;  //Start FSM
    //SPI ready to go as soon as TX data dropped in buffer
    EUSCI_B0 -> IFG |= EUSCI_B_IFG_TXIFG;
}

//void EUSCIB0_IRQHandler(void) {
//    volatile uint8_t RX_Data;
//    //Checks if RX flag went high
//    if (EUSCI_B0 -> IFG & EUSCI_B_IFG_RXIFG) {
//        RX_Data = EUSCI_B0 -> RXBUF;
//        P2 -> OUT &= ~(BIT0 | BIT1 | BIT2);
//        P2 -> OUT |= (RX_Data & 0x07);
//    }
//}

void sendByte_SPI(uint8_t data)
{
    EUSCI_B0 -> TXBUF = data;                        //Drops data into buffer
    while (!(EUSCI_B0->IFG & EUSCI_B_IFG_TXIFG));   //Waits for TX flag to go low
}


/**
 * SPI.h
 *
 * Header for SPI communication
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */



#ifndef SPI_H_
#define SPI_H_

void SPI_INIT();
void sendByte_SPI(uint8_t);

#endif /* SPI_H_ */
```

Delays Library

```
/*
 * delays.c
 * Code file for both the ms and us delay functions.
 * Utilizes __delay__cycles
 * Divide us by 2 cause ?
 *
 * Date: April 9, 2018
 * Authors: Zach Bunce, Garret Maxon
 */




#include "msp.h"
#include "delays.h"

//Takes in desired time delay in ms and clock frequency in MeHz
//Accurate to less than 1%
void delay_ms(int time_ms, int freq_MeHz)
{
    int i;
    //Unit Conversion: MeHz * ms = 10^5 * 10^-3 = 10^2 = 100
    switch (freq_MeHz)
    {
    case F_1p5_MeHz:
        //Accurate to 0.1%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(1500);
        }
        break;
    case F_3_MeHz:
        //Accurate to 0.4%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(3000);
        }
        break;
    case F_6_MeHz:
        //Accurate to 0.6%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(6000);
        }
        break;
    case F_12_MeHz:
        //Accurate to 0.7%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(12000);
        }
        break;
    case F_24_MeHz:
        //Accurate to 0.6%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(24000);
        }
        break;
    case F_48_MeHz:
        //Accurate to 0.4%
        //Div by 2 cause ?
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(24000);
        }
        break;
    default:
        break;
    }
}

//Takes in desired time delay in us and clock frequency in MeHz
//1.5, 3, 6, 12 MHz NOT TUNED
void delay_us(int time_us, int freq_MeHz)
{
    int i;
    int j;
    int z;
    int time_fix;
    //Unit Conversion: MeHz * us = 10^5 * 10^-6 = 10^-1 = 0.1; Accounted for in decrement
    switch (freq_MeHz)
    {
    case F_1p5_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            for (j = 10; j > 0; j--) {
                __delay_cycles(15);
            }
        }
        break;
    case F_3_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(3);
        }
        break;
    case F_6_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(6);
        }
        break;
    case F_12_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(12);
        }
        break;
    case F_24_MeHz:
        //Within +1us 26 < t < 100; Accurate within 1% up to 1000 us
        time_fix = (time_us - 1) / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(24);
            z++;
            z++;
            z++;
        }
        break;
    case F_48_MeHz:
        //Within +1us 25 < t < 100; Accurate within 2% up to 1000 us
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(48);
            z++;
            z++;
            z++;
            z++;
        }
        break;
    default:
        break;
    }
}
/*
 * delays.h
 * Header file for both the ms and us delay functions.
 *
 * Created on: Apr 9, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#ifndef DELAYS_H_
#define DELAYS_H_

#define F_1p5_MeHz  15  //Defines various frequency values in almost MHz (10^5)
#define F_3_MeHz    30  //MeHz labels are used to indicate this
#define F_6_MeHz    60  //Blame data type truncation
#define F_12_MeHz   120
#define F_24_MeHz   240
#define F_48_MeHz   480

void delay_ms(int, int);
void delay_us(int, int);

#endif /* DELAYS_H_ */
```

Keypad Library

```c
/*
 * keypad.c
 * Call KEYPAD_INIT to enable use of keypad
 * chk_Keypad meant for external use
 * Returns ASCII code of symbol pressed (Including numbers)
 *
 * Date: April 13, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include "keypad.h"

static uint8_t keyOut = K_NP;
static int keyFlag = 0;

uint8_t get_Key()
{
    keyOut = chk_Keypad();
    return keyOut;
}

int get_Flag()
{
    return keyFlag;
}

void clear_Flag()
{
    keyFlag = 0;
}

//Sets P4.0-4.3 as rows and P5.0-5.2 as columns
void KEYPAD_INIT()
{
    P4 -> DIR |= (ROWA | ROWB | ROWC | ROWD);   //Sets DIR reg of the outputs
    P5 -> DIR &= ~(COL1 | COL2 | COL3);         //Clears DIR reg of the inputs
    P5 -> REN |=  (COL1 | COL2 | COL3);         //Enables PU or PD resistor
    P5 -> OUT |=  (COL1 | COL2 | COL3);         //PU or PD is set through PxOUT reg
}

//Decodes row & column of key pressed into ASCII value
uint8_t chk_Keypad()
{
    uint8_t RC_Info = KEY_LOCATE();
    switch(RC_Info)
    {
```

```
    case A1:
        return K_1;
    case A2:
        return K_2;
    case A3:
        return K_3;

    case B1:
        return K_4;
    case B2:
        return K_5;
    case B3:
        return K_6;

    case C1:
        return K_7;
    case C2:
        return K_8;
    case C3:
        return K_9;

    case D1:
        return K_Ast;
    case D2:
        return K_0;
    case D3:
        return K_Pnd;
    default:
        return K_NP; //Returns no press; empty value in LCD table
    }
}

//Finds and returns row and column of uppermost key pressed
uint8_t KEY_LOCATE()
{
    uint8_t i;
    uint8_t col = 0;
    for (i = 1; i <= 0x08; i = i << 1) {
        P4 -> OUT |= (ROWA | ROWB | ROWC | ROWD); //Sets all rows high
        P4 -> OUT &= ~i & (ROWA | ROWB | ROWC | ROWD); //Sets selected row low
        col = P5->IN & 0x07;   //Reads column states into lower nibble
        if (col == 0x07) {
            //Do nothing
        }
        else {
            i = i << 4; //Shift row info into upper nibble
            col |= i;   //Add row info to column info
            return col; //Return column and row info
        }
    }
    return 0;
}

void keypad_ISR()
{
    keyOut = chk_Keypad;
}

/*
 * keypad.h
 * Header file for the keypad driver functions.
 *
 * Date: April 13, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_

#define ROWA    BIT0
#define ROWB    BIT1
#define ROWC    BIT2
#define ROWD    BIT3

#define COL1    BIT2
#define COL2    BIT1
#define COL3    BIT0

//Rows: A-D; Columns 1-3
//First row
#define A1      0x13
#define A2      0x15
#define A3      0x16

//Second row
#define B1      0x23
#define B2      0x25
#define B3      0x26

//Third row
#define C1      0x43
#define C2      0x45
#define C3      0x46

//Fourth row
#define D1      0x83
#define D2      0x85
#define D3      0x86

//ASCII hex values for keypad characters
```

```
#define K_1     0x31
#define K_2     0x32
#define K_3     0x33

#define K_4     0x34
#define K_5     0x35
#define K_6     0x36

#define K_7     0x37
#define K_8     0x38
#define K_9     0x39

#define K_Ast   0x2A
#define K_0     0x30
#define K_Pnd   0x23

#define K_NP    0x10

void KEYPAD_INIT();
uint8_t get_Key();
uint8_t chk_Keypad();
uint8_t KEY_LOCATE();

#endif /* KEYPAD_H_ */
```

Set DCO Library

```
/*
 *  set_DCO.c
 *  Code file for the DCO frequency change function.
 *
 *  Date: April 6, 2018
 *  Authors: Zach Bunce, Garret Maxon
 */

#include "msp.h"
#include "set_DCO.h"

void set_DCO(int freq)
{
    CS->KEY = CS_KEY_VAL;                                           //Unlocks CS regs
    CS->CTL0 = 0;                                                   //Clears CTL0 reg

    switch (freq)
    {
    case F_1p5_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_0;                               //Sets DC0 to 1.5 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
       //Sets the clock refs
        break;
    case F_3_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_1;                               //Sets DC0 to 3 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
       //Sets the clock refs break;
    case F_6_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_2;                               //Sets DC0 to 6 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
       //Sets the clock refs break;
    case F_12_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_3;                               //Sets DC0 to 12 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
       //Sets the clock refs break;
    case F_24_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_4;                               //Sets DC0 to 24 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
       //Sets the clock refs break;
    case F_48_MeHz:
        // Transition to VCORE Level 1: AM0_LDO --> AM1_LDO
        while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));
        PCM->CTL0 = PCM_CTL0_KEY_VAL | PCM_CTL0_AMR_1;
        while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));

        // Configure Flash wait-state to 1 for both banks 0 & 1
        FLCTL->BANK0_RDCTL = (FLCTL->BANK0_RDCTL
            & ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) | FLCTL_BANK0_RDCTL_WAIT_1;
        FLCTL->BANK1_RDCTL = (FLCTL->BANK0_RDCTL
            & ~(FLCTL_BANK1_RDCTL_WAIT_MASK)) | FLCTL_BANK1_RDCTL_WAIT_1;

        CS->CTL0 = CS_CTL0_DCORSEL_5;                               //Sets DC0 to 48 MHz
        CS->CTL1 = CS->CTL1
            & ~(CS_CTL1_SELM_MASK | CS_CTL1_DIVM_MASK)| CS_CTL1_SELM_3; //Sets MCLK to DCO
        break;
    default:
        break;
    }
    CS->KEY = 0;                                                    //Locks CS regs
}


/*
 *  set_DCO.h
 *  Header file for the DCO frequency change function.
 *
 *  Date: April 6, 2018
 *  Author: Zach Bunce, Garret Maxon
 */
#ifndef SET_DCO_H_
#define SET_DCO_H_

#define F_1p5_MeHz  15  //Defines various frequency values in almost MHz (10^5)
#define F_3_MeHz    30  //MeHz labels are used to indicate this
#define F_6_MeHz    60  //Blame data type truncation
#define F_12_MeHz   120
#define F_24_MeHz   240
#define F_48_MeHz   480

void set_DCO(int);

#endif /* SET_DCO_H_ */
```

-1
Missing references.