EGR 226: Microcontroller Programming and Applications

Spring/Summer 2020

Instructor: Prof. Trevor Ekin

Final Project

Developing an Interactive System with the MSP432

Dustin Matthews

7/29/2020

Contents

List of Figures

1. Objectives

The main objective of the project was to design a functional control system utilizing several components from the Lab portion of EGR 226 and integrating various capabilities of the MSP432 microcontroller.

2. Equipment

Part	Description	Model	Measured Value	Notes
Code Composer Studio	Texas Instruments Programming Environment	Version 9.30	n/a	n/a
MSP432P401R	Texas Instruments Microcontroller	n/a	n/a	n/a
Keypad	4x3 Matrix Array	n/a	n/a	n/a
16x4 LCD	Dot-Matrix Liquid Crystal Display	HD44780U	n/a	n/a
Transistors	n/a	2N7000	n/a	n/a
Resistors	n/a	n/a	100 Ohm and 220 Ohm	n/a
DC Motor	n/a	n/a	n/a	n/a
Servo	n/a	SG90	n/a	n/a
LEDs	5mm Red, Green, Blue, Yellow	n/a	n/a	n/a
Small Breadboard x 2	n/a	n/a	n/a	n/a
Breadboard Jumper Wires	n/a	n/a	n/a	n/a
Trim Potentiometers	Three terminal variable resistor	n/a	10k	n/a
Push Buttons	Push button switch	B3F	n/a	n/a
Optocoupler	Opto-Isolator	H11B	n/a	n/a

3. Introduction and Requirements

2.1 Basic Overview:

The control system of the project was comprised of three main components: the MSP432, a numeric Keypad, and a 16x4 LCD. This formed an interactive system which could receive input, display output, and perform actions utilizing various other peripheral components. Four basic user menus were required: a main menu, a door menu, a motor menu, and a lights menu. Navigating through the menus required printing a basic GUI to the LCD to allow the user to make selections. The program starts by welcoming the user and giving basic instruction, as seen in the figures below.

An extra component integrated into this project was a physical enclosure for the components, seen in Figure 1 Final Project Enclosure. A platform and wall were constructed of plastic. Slots and holes were drilled and cut into the wall in order to mount the LCD, Servo, LEDs, and DC motor. A slot was cut beneath the LCD to house a breadboard, to which was wired the two interrupt buttons and two potentiometers which controlled the brightness/contrast of the LCD. A physical door was constructed and mounted to the Servo as well.

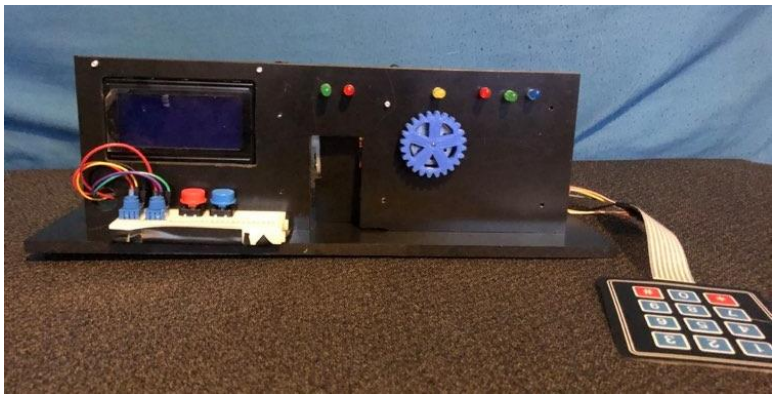


Figure 1 Final Project Enclosure

2.2 Menu Navigation

The Main Menu displayed on the LCD allowed the user to use the keypad to choose [1] for Door, [2] for Motor, and [3] for Lights.

In all menus, the user must press the [#] key to make their selection, and error checking within the function for reading the keypad would print an error message if the user entered a key that was not within the scope of the menu. Each time the user pressed a key on the keypad, it would be printed to the LCD. At any time during selection, the user could press the [*] key to clear their selection, which was more useful when choosing a 1-3 digit length duty cycle within the Lights menu. See Figure 2 Startup and Main Menu.

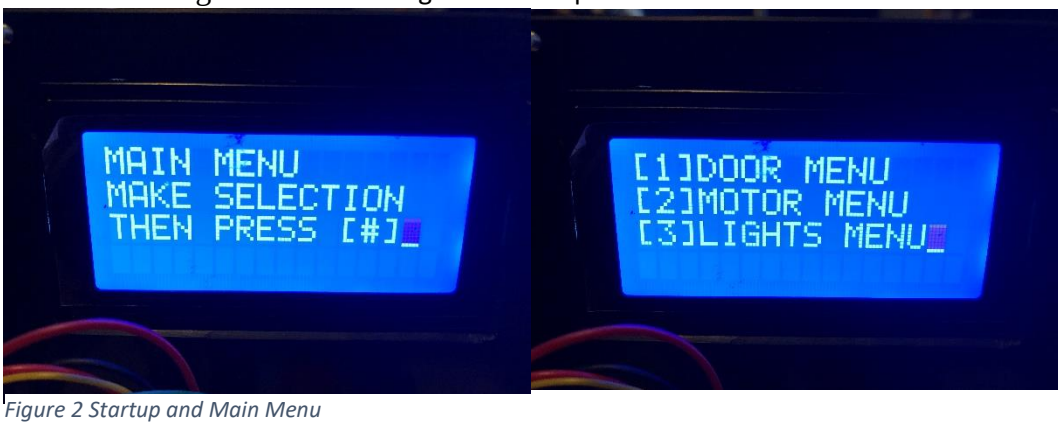


Figure 2 Startup and Main Menu

2.3 Door Menu

Within the Door Menu, shown below, the user has the option of entering [1] or [2] to open or close the door. Upon program startup, the door would be closed, and a red LED would be on. Upon opening the door, the red LED would be toggled off, and a green LED was toggled on. The “door” consisted of a Servo motor with a plastic door attached, which would physically open or close.

Figure 3 Door Menu

2.4 Motor Menu

The motor menu allowed the user to activate a DC motor operating at 40Hz to a speed of [0] to [9] with the keypad. Upon startup, the DC motor would be off. An external Emergency

Stop button was integrated into the circuit that would reset the motor to a speed of 0 at any time.

An extra functionality integrated into the project was a warning LED. If the user set the motor speed to 8 or 9, meaning the motor was at or above 80% speed, a yellow LED would blink on and off at .25 second intervals until the motor was set below 80% again, either in the Motor Menu or by using the Emergency Stop button.

Figure 4 Motor Menu

2.5 Lights Menu

Within the lights menu, the user was able to choose between a Red, Green, or Blue LED by entering [1], [2], or [3] on the keypad.

Figure 5 Lights Menu

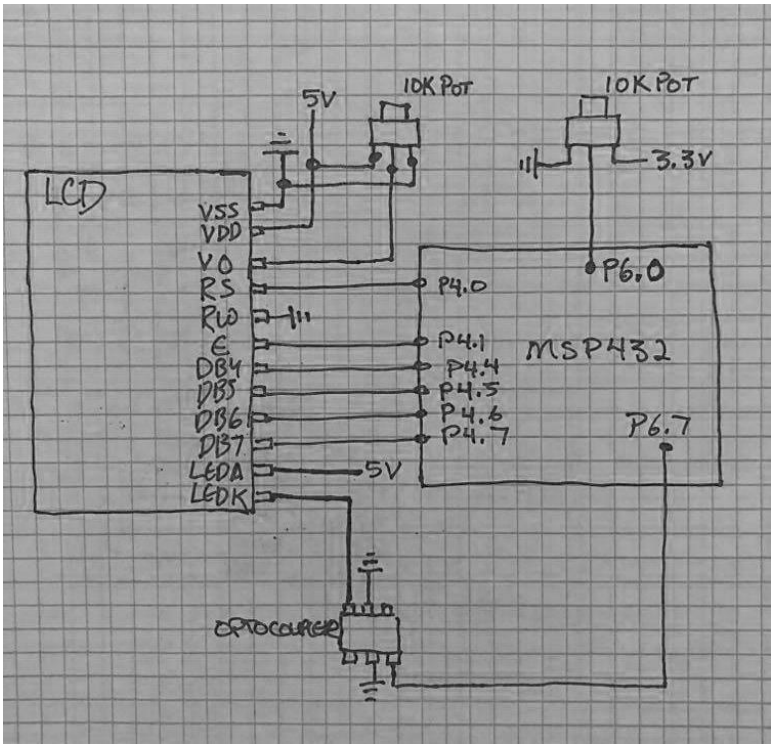
The GUI would then navigate to a new screen, allowing the user to set the brightness level of the chosen LED from 0 – 100. Each LED is able to be driven at its own duty cycle.

Figure 6 Set Brightness Interface

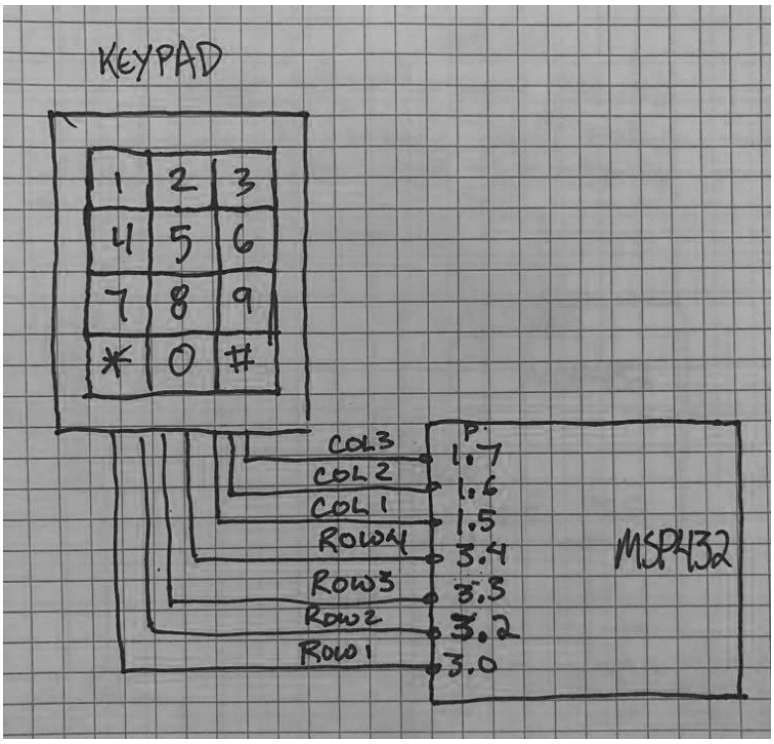
An external switch was wired into the system, which would toggle the LEDs on and off at their previously set duty cycles. The logic for this function was such that even if one LED had been set, then toggled off, and then another LED was set, the button would turn off all LED's and then turn them all back on together at the next press, rather than flip flopping them.

4. Schematics

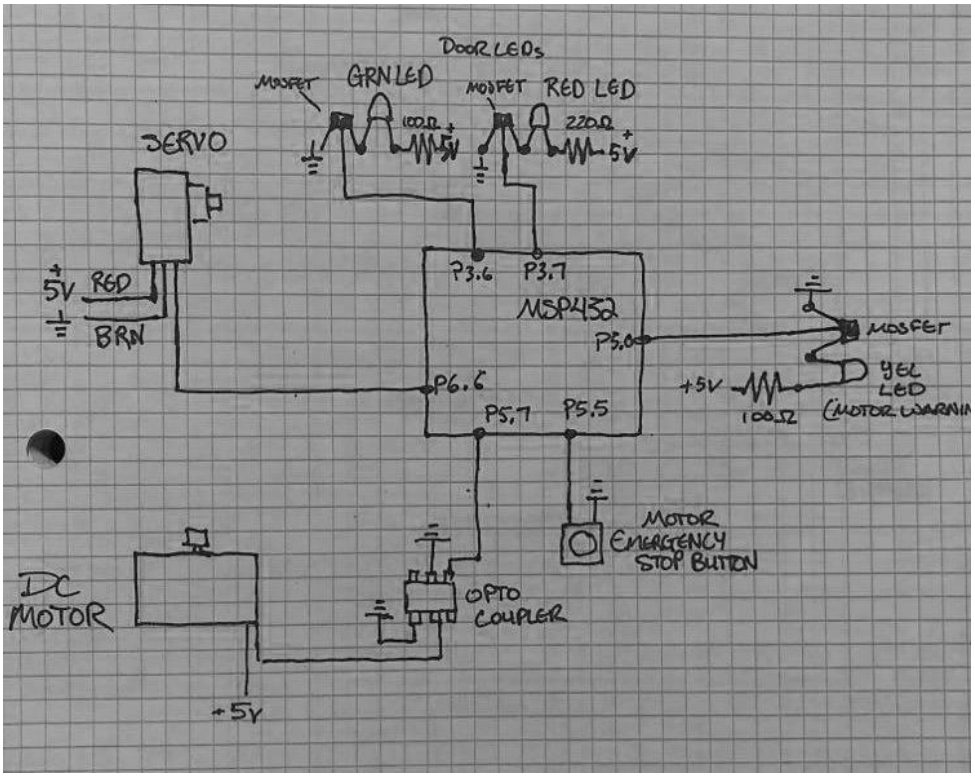
LCD Wiring



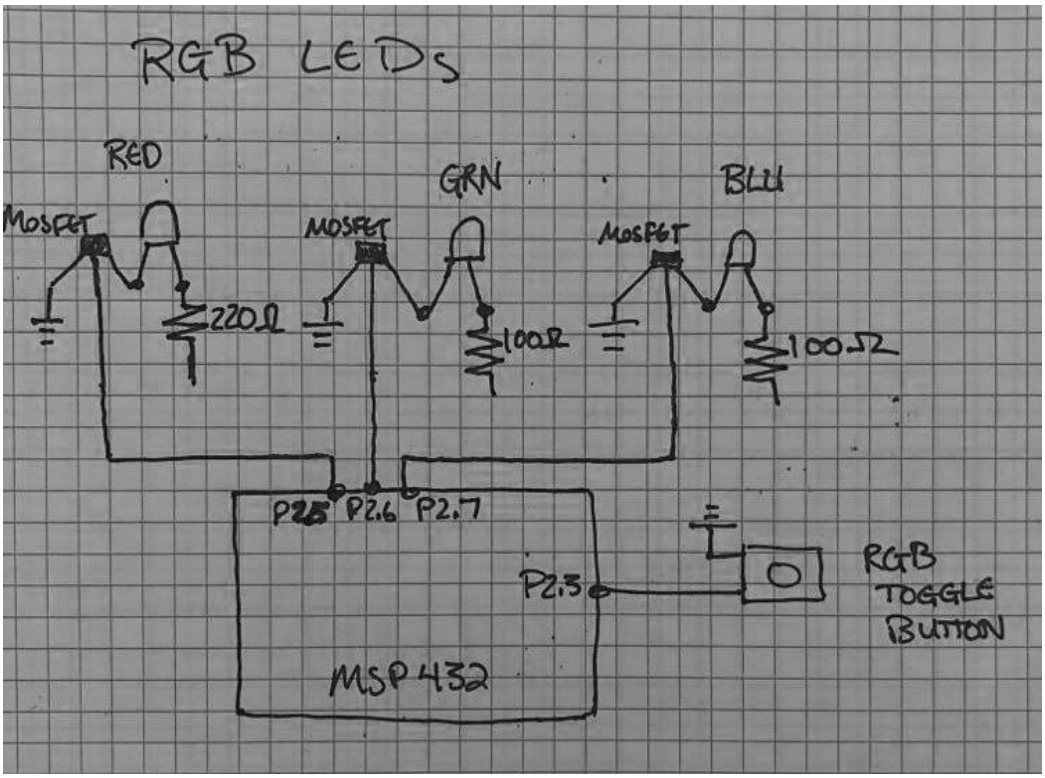
Keypad Wiring



Motor and Servo Wiring

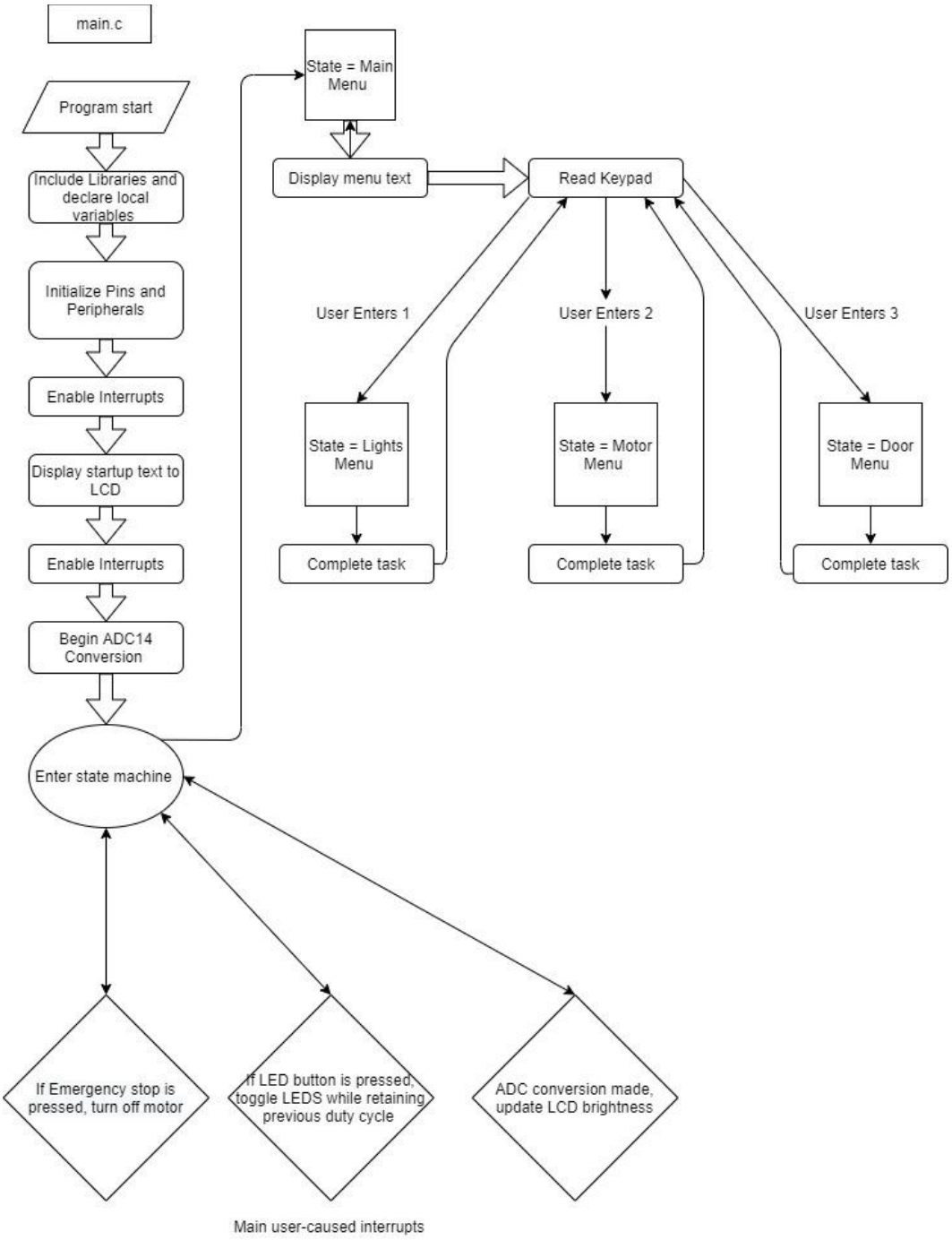


RGB Wiring

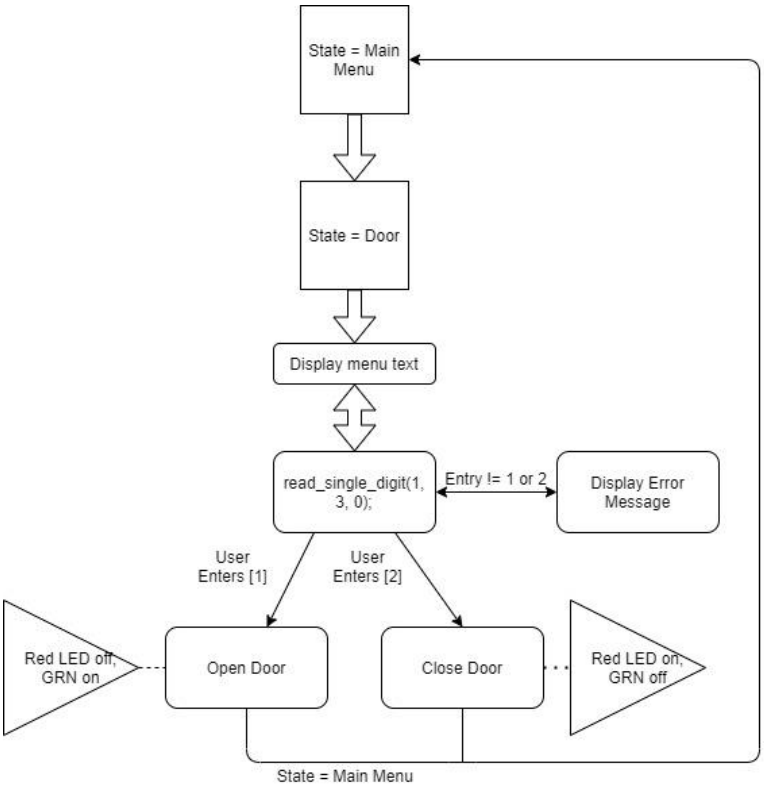

RGB LEDs
RED
GRN
BLU
MOSFET
MOSFET
MOSFET
220Ω
100Ω
100Ω
P2.5
P2.6
P2.7
P2.3
MSP432
RGB TOGGLE BUTTON

5. Flow Charts

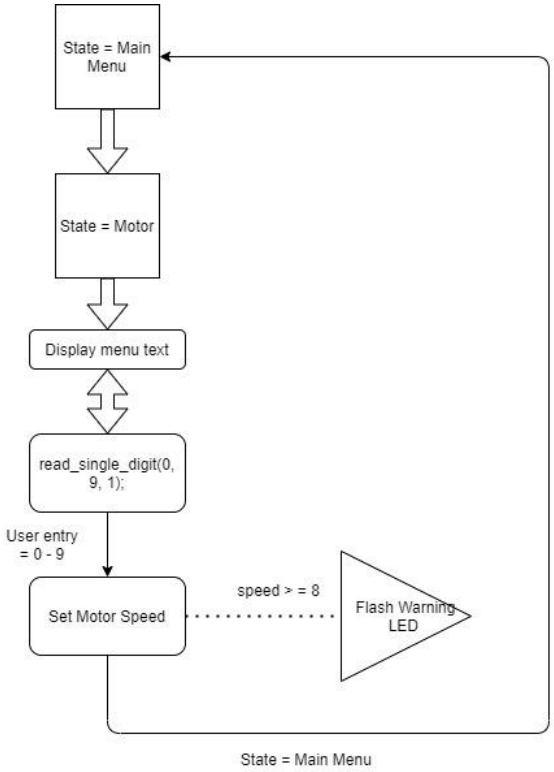
General Program Flow



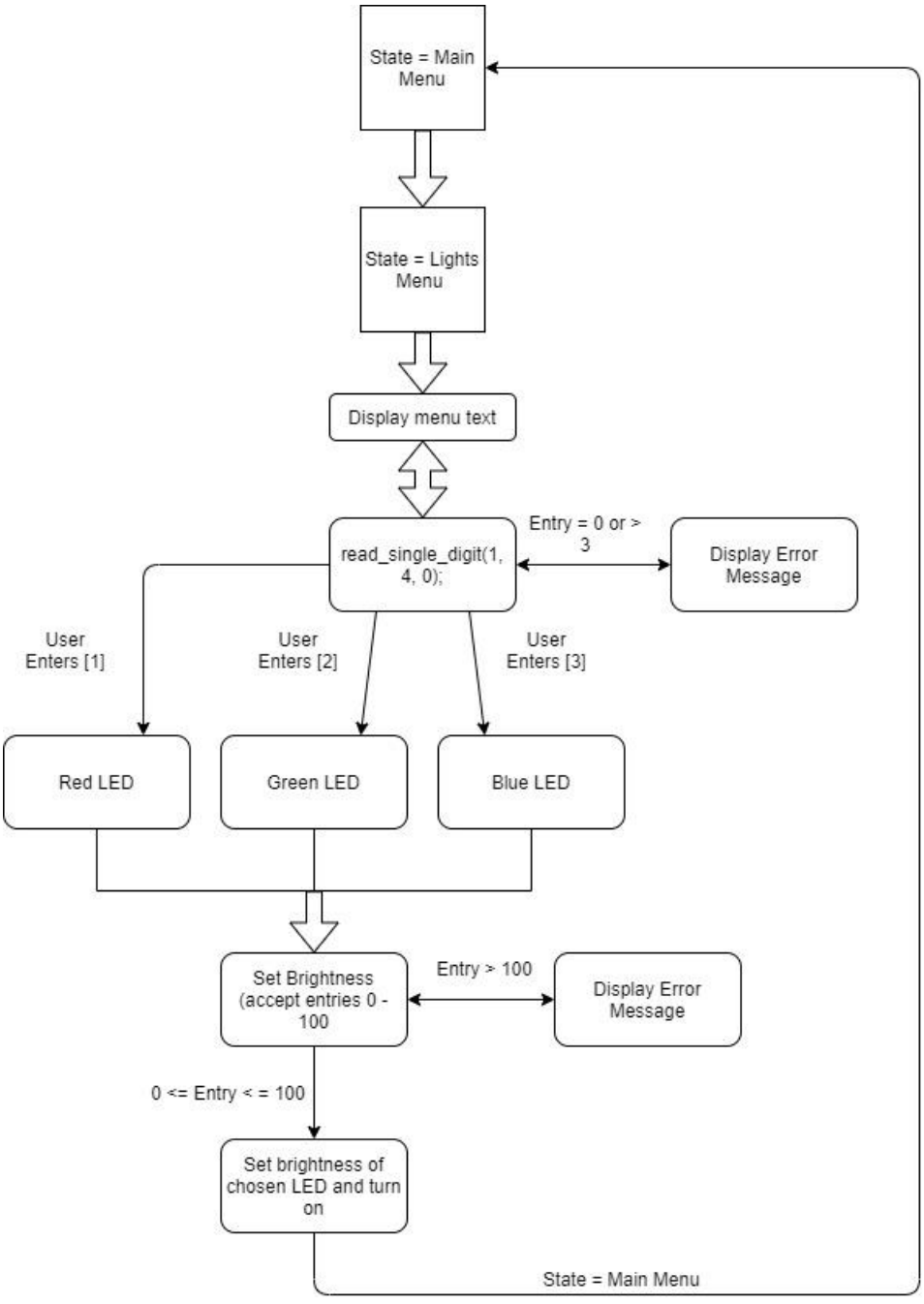
Door Menu



Motor Menu



Lights Menu


State = Main Menu
State = Lights Menu
Display menu text
read_single_digit(1, 4, 0);
Entry = 0 or > 3
Display Error Message
User Enters [1]
User Enters [2]
User Enters [3]
Red LED
Green LED
Blue LED
Set Brightness (accept entries 0 - 100
Entry > 100
Display Error Message
0 <= Entry < = 100
Set brightness of chosen LED and turn on
State = Main Menu

6. Procedure

6.1 Overview

Given the number of functions and peripherals within the scope of the project, a large number of libraries and source files were created in order to keep the main.c source file relatively tidy. After initializing all the necessary peripherals, main.c enters a basic state machine to switch between the different menu functions. Once an action is completed, it will return to the main menu.

6.1.1 Pin Planning

One of the first steps taken before implementing any significant amount of code was to carefully plot out all the specialized pins that would need to be utilized, especially Timer A pins (avoiding soldering in extra pins was a goal that was met), as the code would be heavily dependent on various PWM signals. Listing all the major pins used in a Pin Map at the beginning of main.c provided an easy reference point as the rest of the code was developed.

6.2 Program Development

6.2.1 LCD Display

The first component to integrate into the program was the LCD display- this would comprise the user interface and had to be functional in order to navigate to further peripherals as they were developed. One of the requirements for the LCD display was that its brightness be adjustable in two steps:

- Read a potentiometer using the ADC15 functionality of the MSP432
- Use the potentiometer reading to calculate a duty cycle and apply it via a TimerA PWM.

In order to have a smooth brightness adjustment at any point in the program, the ADC15 channel 15 was configured to act on interrupts any time a conversion was made, as seen in Figure 7 ADC Configuration.

```
41 /*******************************************************************************
42  * Function:    LCD_Pot()
43  * Description: Set up ADC15 on Pin P6.0 with interrupts to read the potentiometer when
44  *              a conversion is made.
45 *******************************************************************************/
46 void LCD_Pot(){
47
48     /* Read Pin for the Potentiometer (A15) */
49     P6->SEL1 |= BIT0;                          //Tertiary Function
50     P6->SEL0 |= BIT0;                          //Tertiary Function
51     P6->DIR &=~ BIT0;                          //Configure for input
52
53     /* Initialize the ADC14 to read the potentiometer */
54     ADC14->CTL0 &=~ ADC14_CTL0_ENC;            //disable ADC converter
55     ADC14->CTL0 |= 0x44540210;                 //Set/hold pulse mode, SMCLK, 16clks, continuous sampling, ADC on
56     ADC14->CTL1 = 0x000F0030;                  //14 bit resolution, store in mem15 register
57     ADC14->MCTL[15] =0x0000000F;               //channel 15 for mem15 interrupts
58     ADC14->IER0 |= 0x00008000;                 //enable interrupts on mem15
59     ADC14->CTL0 |= ADC14_CTL0_ENC;             //enable ADC14
60 }
```

Figure 7 ADC Configuration

Within the interrupt handler, the new duty cycle is calculated and assigned to TimerA2.4, as seen in Figure **Error! Reference source not found.**. TimerA2 was used for several other PWM functions, all of which operated at 50Hz. An attempt was made on lines 102-103 to turn off the

```
93 /*****************************************************************************
94  * Function     ADC14_IRQHandler()
95  * Description  Read potentiometer if a conversion is made, adjust LCD brightness
96  *****************************************************************************/
97 void ADC14_IRQHandler(void){
98
99     ADC14->CLRIFGR0 |= 0x00008000;
100    read = ADC14->MEM[15];                  //read mem[15] register
101    lcd_dc = (read / 16384) * 60000;        //calculate new duty cycle using the 14-bit resolution value
102    if (lcd_dc < 100)                       //if duty cycle < 100, turn off LCD (avoids sputtering activation)
103        lcd_dc = 0;
104    TIMER_A2->CCR[4] = lcd_dc;              //adjust LCD brightness
105    ADC14->CTL0 |= ADC14_CTL0_SC;
106 }
107
```

Figure 8 ADC ISR

LCD if the reading of the potentiometer went below a certain threshold to avoid flickering, but more experimentation would need to be done with the value of this threshold to make the function effective.

To print displays to the LCD, the library of functions provided by the professor was used. Earlier in the class, some basic code for printing to the LCD had been written, but it was deemed that using the provided library would ensure a smoother and less error-prone experience as the project was developed.

6.2.2 Keypad

The basic code for initializing and reading the keypad had been developed earlier in the semester. These functions were modified to reflect a change in the pins used on the MSP, which were chosen within Port 1 for the columns and Port 3 for the rows. This made setup much quicker, as bit masks could be easily used to set the registers.

6.2.3 User Input

As the program is dependent on user input to perform any tasks, two different functions were developed to accept user input.

The first was to be used most often; as seen in the Introduction, most menus require only a single digit to be entered within a specific range, i.e. 1-2 or 1-3. As such, within any menu that requires a single digit this same function is called, and the desired range is passed to it. An additional variable, zero_enable, is also passed to the function. The reason for this being that the keypad has a [*] key between [9] and [0], so an entry of [0] actually returns 11 from the keypad. Zero_enable allows the [0] to also be accepted, and was specifically used in the Motor Menu. In a while(1) loop, the keypad is continually read and error checked for an appropriate entry. When no button is pressed, the read_keypad() function continually returns -1. If a numeric key is pressed, it is printed to the LCD and loaded into a constantly updated array variable.

Requiring the [#] key to be pressed for an entry ensures that the user will not accidentally make an entry they did not wish to make. If a valid entry is made, it is returned for further action, else an error message is displayed. The function can be seen below in Figure 9 Read Single Digit Function

```
int read_single_digit(int min, int max, int zero_enable){
    int i = 0, val = -1;
    char select[2] = {'0', '\0'};              //array to store the last valid keypad press
       while(1){
           i = read_keypad();
           if (i > 0)
               lcdSetChar(keys[i], 0, 3);       //write entry to the LCD
           if ((i > 0) && (i != 10) && (i != 12)){
               select[0] = keys[i];
               val = atoi(select);              //store the value as an integer
           }
           else if ((zero_enable) && (i == 11)){
               select[0] = '0';
               val = atoi(select);
           }
           if (i == 10){                            // [*] to clear entry
               memset(select, '0', strlen(select)); //clear the array
               val = atoi(select);
               lcdSetText("CLEARING...", 0, 3);
               SysTick_delay_ms(500);
               lcdSetText("           ", 0, 3);
           }
           if ((i == 12) && (val >= min) && (val < max)){   //value is in range
             lcdClear();
             lcdSetText("YOU ENTERED", 0, 0);
             lcdSetChar(select[0], 0, 1);
             SysTick_delay_ms(2000);
             lcdClear();
             return val;                                    //return the user selection
           }
           else if ((i == 12) && (val < min) | (val >= max)){
               lcdSetText("INVALID ENTRY", 0, 3);
               SysTick_delay_ms(2000);
               lcdSetText("             ", 0, 3);
           }
       }
}
```

Figure 9 Read Single Digit Function

The second function for processing user input was specifically developed for setting the brightness of the LEDs in the Lights Menu, which required a value of 0-100 to be entered. This function is very similar to the first, the major difference being that the input is placed into a larger array, allowing three digits to be stored. This is again error checked to see if it is within the desired range, else an error message is printed to the LCD. The [*] button is also useful within this function, as it allows the user to clear their current entry if they pressed the wrong key. This will print "CLEARING…" to the LCD and erase the current entry from the screen.

6.2.4 Door Menu

Once the program was able to successfully receive keypad input and tested to verify that the user could navigate through the menus with the GUI, the Door Menu was developed. As in all menus, the Door Menu first prints its display to the LCD. The keypad is then read until the user makes an entry of [1] or [2] to open or close the door.

The opening or closing of the door is accomplished by manipulating the duty cycle of a servo motor, which is set as a PWM in TimerA2.3, as seen in Figure 10 Door Open and Close Functions. The exact PWM values were tweaked until suitable and did not line up with the values that were given in the data for the servo. Each time the door was opened, a green LED configured with GPIO was turned on, and any time the door was closed, a red LED configured with GPIO was

turned on. The respective LEDs would remain on as long as the door remained in its current state.

```
242 /*****************************************************************************
243  * Function:    door_open()
244  * Description: Adjusts servo PWM to open the door
245  *              Turn Red indicator LED off and Green LED on
246 *****************************************************************************/
247 void door_open(){
248     TIMER_A2->CCR[3] = 3100;               //Set Servo PWM to open door
249     SysTick_delay_ms(500);                 //delay to allow door to open
250     TIMER_A2->CCR[3] = 0;                  //disable PWM
251     P3->OUT &= ~BIT7;                      //turn off Red LED
252     P3->OUT |= BIT6;                       //turn on Green LED
253 }
254
255 /*****************************************************************************
256  * Function:    door_close()
257  * Description: Adjusts servo PWM to close the door
258  *              Turn Green indicator LED off and Red LED on
259 *****************************************************************************/
260 void door_close(){
261     TIMER_A2->CCR[3] = 6000;               //Set Servo PWM to close door
262     SysTick_delay_ms(500);                 //delay to allow door to close
263     TIMER_A2->CCR[3] = 0;                  //disable PWM
264     P3->OUT &= ~BIT6;                      //turn off Green LED
265     P3->OUT |= BIT7;                       //turn on Red LED
266 }
267
```

Figure 10 Door Open and Close Functions

6.2.5 Motor Menu

The Motor Menu utilized the read_keypad() function to accept a key press of 0-9 (using the zero_enable variable). The user's entry would set the speed of the motor to 0 – 90% duty cycle, which was configured in TimerA2.2, operating at 50Hz. The function void motor() would update the duty cycle using the equation on line 283 in Figure 11 Motor Menu Function below.

```
268 /*****************************************************************************
269  * Function:    motor()
270  * Description: Interface for the door menu.
271  *              Receives user keypad input to set the speed of the DC motor.
272  *              Displays the user's choice on the LCD before returning to main menu.
273  *              If duty cycle exceeds 80%, initiate a TA0 interrupt to toggle a blinking
274  *              warning LED.
275 *****************************************************************************/
276 void motor(){
277     float motor_set;
278     motor_set = read_single_digit(0, 10, 1);        //Accept input 0 - 9
279     if(motor_set>=0){
280         lcdSetText("SPEED", 0, 0);                  //Write user's choice to the LCD
281         lcdSetInt(motor_set, 0, 1);
282         SysTick_delay_ms(500);
283         TIMER_A2->CCR[2] = (motor_set / 10) * 60000;//update DC motor duty cycle
284         if (TIMER_A2->CCR[2] >= 48000)              //duty cycle >= 80%
285             TIMER_A1->CTL |= (BIT4 | TIMER_A_CTL_CLR);  //initiate TA0 interrupt
286         else TIMER_A1->CTL &= ~BIT4;                //duty cycle < 80%, disable interrupt
287         SysTick_delay_ms(1500);
288         lcdClear();
289         state = MAIN_MENU;
290     }
291 }
```

Figure 11 Motor Menu Function

The Motor Menu was developed further by adding a warning LED that would activate and blink whenever the motor's duty cycle met or exceeded 80%. This was done by activating TimerA1.0 with interrupts, which was initialized off, and toggling the LED within the ISR every

.25 seconds. If the motor's duty cycle is brought below 80%, the interrupt is deactivated to turn off the LED.

6.2.6 Lights Menu

Unlike the previous menus, which had one stage of interaction with the user, the Lights Menu was developed in two stages. The first portion of the Lights Menu prompts the user to select an LED- Red, Green, or Blue, by pressing [1], [2], or [3], respectively. Upon selection of the LED, the program shifts to the second portion of the menu, wherein the user can select a brightness level for the chosen LED with a value of 0-100. Once the brightness level is selected, a duty cycle is calculated and assigned to the LED. Issues were had with the functionality of the RGB module in the project kit, so three normal LED's were used in its stead.

The initial Lights Menu was rather bulky, having separate if statements for each LED choice the user might make to calculate a duty cycle. This was simplified, however, by integrating a simple for() loop and storing the duty cycle variables for the LED's into a float array, RGB_DC. Using this method, the user's choice could be printed to the LCD and a new duty cycle assigned to any LED, all within a short, much cleaner function, which is seen in Figure 12 Lights Menu Function.

```
292 /*******************************************************************************
293  * Function:    lights()
294  * Description: Interface for the lights menu.
295  *              Receives user keypad input to choose the Red, Green, or Blue LED on the
296  *              RGB module.
297  *              Choice is stored as a variable that reflects which TA0->CCRn instance to
298  *              update. Receives user input 0 - 100 to set the brightness level of the
299  *              chosen LED. New duty cycle calculated and assigned to the proper CCRn register.
300  *              Duty cycles are global variables, allowing the CCRn registers to be
301  *              toggled on and off with a button press.
302  *              Displays the user's choice on the LCD before returning to main menu.
303 *******************************************************************************/
304 void lights(){
305     int i, j;
306     float level;
307
308     i = read_single_digit(1, 4, 0);                 //accept input 1 - 3
309     lcdClear();
310     for(j = 0; j < strlen(led_txt[i-1]); j++){      //write user's LED choice to the LCD
311         lcdSetChar(led_txt[i-1][j], j, 0);
312     }
313     lcdSetText("SET LVL 0-100", 0, 1);              //Display instructions
314     lcdSetText("[*] TO CLR ENTRY", 0, 2);
315     SysTick_delay_ms(500);
316     level = RGB_brightness();                       //read user entry
317     RGB_DC[i] = (level / 100) * 3000;               //calculate new duty cycle
318     if (RGB_DC[i] > 0) LED_check = 1;               //set flag for LED button
319     TIMER_A0->CCR[i + 1] = RGB_DC[i];               //input duty cycle to CCR[i + 1] (starts on CCR2)
320     lcdClear();
321     state = MAIN_MENU;
322 }
```

Figure 12 Lights Menu Function

The PWMs for the LEDs in the Lights Menu were configured within TimerA0.2, 0.3, and 0.4. These were especially convenient as the pins used were sequential (P2.5-2.7), which was necessary for the implementation of the for() loop and array in the lights() function. A frequency of 1000Hz was chosen for the PWM, as this was fast enough to give a solid level of light without blinking, as well as to make the duty cycle calculations simple.

6.3.1 Button Interrupts

Besides using the potentiometer driving the brightness level of the LCD, the user could trigger two other interrupts in the program using two buttons. One button was an Emergency Stop for the DC motor, the second was a toggle switch for the RGB LEDs. Each of these two buttons was configured and wired to always read HIGH unless pressed, at which point a port interrupt would be entered.

In past Labs, SysTick was typically used to debounce buttons, but this project already relied heavily on SysTick for reading the keypad as well as writing text to the LCD. A quick experiment with a SysTick debounce in the port handlers revealed that doing so would stall the program. An alternative method was developed, in which the port handler would simply initialize the counter on TimerA3.0, which triggered an interrupt after 15ms. The state of the button was then checked within the TA3.0 handler and the button's task would be completed.

If the Emergency Stop button was pressed, the handler would set TIMER_A2->CCR[2] to 0 and clear the OUT register of the warning LED.

Several logic methods were experimented with for the RGB toggle button. One of the requirements of the button was that the previous duty cycle of the LED be maintained each time it was toggled back on. To do this, the duty cycle array, RGB_DC[] was moved into the Interrupt_Handlers library so that it could be manipulated directly in the handler. Earlier methods tried would toggle the LEDs on and off, but if one LED was toggled off, and another received a new duty cycle from the lights menu, the button would then switch back and forth between the two LEDs. To get around this, a new variable was added into the lights menu, LED_check, and was set to 1 any time an LED was updated to greater than 0 in the Lights Menu. If the variable was true once the ISR was entered, all the LEDs would be toggled off, otherwise all LEDs would be toggled on (to whatever duty cycle they had been assigned). This variable was also manipulated within the ISR to keep track of which state the LEDs were in each time the ISR was entered as seen in Figure 13 TA3_0 Handler.

```
/*****************************************************************************
* Function      TA3_0_IRQHandler()
* Description   Debounce timer for buttons.
*               If the DC motor killswitch is pressed, set the Duty Cycle to 0.
*
*               If the RGB toggle button is pressed, manipulate the CCRn registers
*               to toggle the RGB module while retaining the duty cycle values, which
*               are saved as global variables.
*****************************************************************************/
void TA3_0_IRQHandler(){
    int i;

    TIMER_A3->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;  //clear interrupt flag
    if((P5->IN & BIT5) == 0){                   //If switch is still low, update press variable
        TIMER_A2->CCR[2] = 0;                   //Set DC motor duty cycle to 0
        TIMER_A1->CTL &= ~BIT4;                 //disable warning LED Timer
        P5->OUT &= ~BIT0;                       //turn off warning LED
    }
    if((P2->IN & BIT3) == 0){                   //RGB button pressed, manipulate CCRn registers
        if (LED_check){                         //if any LEDs have been set, toggle all off
            for (i = 2; i < 5; i++){
                TIMER_A0->CCR[i] = 0;
            }
            LED_check = 0;
        }
        else{
            for (i = 2; i < 5; i++){            //else toggle LEDs all on
            TIMER_A0->CCR[i] = RGB_DC[i - 1];
            }
            LED_check = 1;
        }

    }
    TIMER_A3->CTL &= ~BIT4;                     //Halt TA3
    TIMER_A3->CTL |= TIMER_A_CTL_CLR;           //Clear TA3R count
}

```

Figure 13 TA3_0 Handler

7 Conclusion and Future Work

7.1 Overview

The project was successful in that it met all requirements and functioned without any issues such as lag or errors in programming.

-The GUI was easy to navigate, and error checking ensured that the program would not get stalled out due to an unforeseen keypad entry.

-The door successfully opened, lighting the green LED, and successfully closed, lighting the red LED. Upon startup, the door would be closed and the red LED was on.

-The motor speed was fully programmable from 0-9 as desired. The emergency stop button worked without any issues, and the warning LED flashed at the appropriate times.

-The red, green, and blue LEDs were fully programmable with independent brightness levels of 0-100. The toggle switch worked without any issue.

-After each task, the program would return to the main menu.

7.2 Challenges

Implementing the various tasks required of the project required extensive interaction between and uses of interrupts and timer peripherals. Setting up the ADC on a new pin that was not used

in the ADC lab also required extensive reading in order to navigate the 32 bit registers, but a much greater understanding of registers and their use was gained. The SysTick timer had to be integrated carefully into the code at times, in order to prevent the program from stalling out. As noted earlier, this happened when attempting to debounce a port interrupt button, and an alternative debounce was developed using a TimerA interrupt.

7.3 Future Work

One further function, a screensaver, was attempted but was not successfully implemented before the demonstration of the project. This utilized a Timer32 interrupt that would instigate the screensaver after 10 seconds of inactivity from the keypad. The Timer32 LOAD register would be reset for 3 seconds while in the screensaver state, and a variable would be updated that prompted the screensaver function to print a message in various locations of the LCD. If a key was pressed, the program was to return to the main menu. The main issue behind the screensaver not working was found to be some variables declared as extern type within one of the libraries, rather than as volatile, which meant they were not updated properly in the Timer32 ISR. With some minimal tweaking, the screensaver could be implemented and fully functional.

Appendices

A. Main.c

```
/*****************************************************************************************
**********
 *Author        Dustin Matthews
 *Course        EGR 226: Microcontroller Programming and Applications
 *Assignment    Final Project
 *Date          7/26/20
 *Instructor    Professor Ekin
 *File          EGR226_FinalProject_Matthews
 *Description   This program uses an LCD to act as a GUI consisting of
 *              four basic menus: Main, Door, Motor, and Lights. The user
 *              navigates the menus and interacts with the program using a
 *              keypad. After each action the program returns to the main menu.
 *
 *              The LCD brightness can be adjusted at any time with a potentiometer,
 *              which is tied to an ADC15 interrupt. The potentiometer level is used to set a
 *              PWM with a Timer A instance to adjust the LCD's brightness.
 *
 *              The door menu allows the user to open or close the door,
 *              which will toggle either a green LED (open) or a red LED
 *              (closed).
 *
 *              The motor menu allows the user to set the speed of a DC motor
 *              (0-9). The DC motor can be stopped at any time using a kill
 *              switch, which is tied to a port interrupt and sets the duty
 *              cycle to 0. If the motor duty cycle is at or above 80%, a warning
 *              LED will blink every .25 seconds until is goes below the 80% level.
 *
 *              The lights menu allows the user to select a red, green, or blue
 *              LED on an RGB module, and then set the brightness of the LED
 *              from 0-100. A button will toggle the LEDs on or off at anytime,
 *              retaining their previously set brightness level.
 *
```

```
 *Notes:        Various functions are stored in the attached libraries to make the main.c file
easier to read
 *              and navigate.

*************************************************************************************************
**********/
#include "msp.h"
#include "LCD_Library.h"
#include "Keypad_Library.h"
#include "SysTick_Library.h"
#include "LCD_GUI.h"
#include "User_Keypad_Input.h"
#include "TIMER_A_Library.h"
#include "Button_Pot_Init.h"
#include "Interrupt_Handlers.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*********************************
 *** Local function declarations ***
 *********************************/
void initialize();
void main_menu_interface();
void program_state();
void door();
void door_open();
void door_close();
void motor();
void lights();


/**********PIN MAP************
 * * * * * * * * * * * * * * *
 *********PWM PINS************
 *  TA0.2 RED on RGB    > P2.5
 *  TA0.3 GRN on RGB    > P2.6
 *  TA0.4 Blue on RGB   > P2.7
 *                        *
 *  TA2.2 DC Motor      > P5.7
 *  TA2.3 Servo         > P6.6
 *  TA2.4 LCD V0        > P6.7
 *                        *
 *********Keypad*************
 *-> GPIO                 *
 *  Row 1               > P3.0
 *  Row 2               > P3.2
 *  Row 3               > P3.3
 *  Row 4               > P3.5
 *  Col 1               > P1.5
 *  Col 2               > P1.6
 *  Col 3               > P1.7
 *                        *
 *********LCD****************
 *-> GPIO                 *
 *  RS                  > P4.0
 *  E                   > P4.1
 *  DB4                 > P4.4
 *  DB5                 > P4.5
 *  DB6                 > P4.6
 *  DB7                 > P4.7
```

```
 *                           *
 *  ADC Pot Input       > P6.0
 *                           *
 ********Buttons*************
 *-> Set as Port Interrupts  *
 *                           *
 *  LED Toggle          > P2.3
 *  Motor Stop          > P5.5
 *
 ********Warning LED**********
 *-> Set as GPIO             *
 *                           *
 *  Red LED             > P5.0
 *                           *
 ********Door LEDs************
 *-> Set as GPIO             *
 *                           *
 *  Green LED           > P3.6
 *  Red LED             > P3.7
 *                           *
 ***************************/

/*********************************
***** Char array for RGB text *****
*********************************/
char led_txt[3][6] = {"RED", "GREEN", "BLUE"};

/*********************************
***** State machine variables *****
*********************************/
enum State_enum {MAIN_MENU, DOOR, MOTOR, LIGHTS};
static unsigned int state;

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

       initialize();                              //initialize peripherals

       /* Enable necessary interrupts */
    NVIC_EnableIRQ(ADC14_IRQn);
    NVIC_EnableIRQ(PORT2_IRQn);
    NVIC_EnableIRQ(PORT5_IRQn);
    NVIC_EnableIRQ(TA1_0_IRQn);
    NVIC_EnableIRQ(TA3_0_IRQn);
    __enable_irq();

    startup_display();                          //Display starting text on LCD
    ADC14->CTL0 |= ADC14_CTL0_SC;               //Start conversion of LCD potentiometer
       while (1){
           program_state();                        //Enter user interface program
       }
}

/*****************************************************************************************
 * Function:    initialize()
 * Description: Runs all initialization/setup functions for timers and external
 *              components
```

```
*******************************************************************************************/
void initialize(){

    init_buttons();
    SysTickInit_NoInterrupts();
    init_TA0_Pins();
    init_TA1();
    init_TA2_Pins();
    TIMER_A3_delay();
    keypad_init();
    LCD_Pot();
    lcdInit();
}

/*******************************************************************************************
 * Function:    program_state()
 * Description: State machine to facilitate the program's primary functions.
 *              Each state reflects an interactive menu, and first calls a display
 *              function to inform the user of their options before receiving input.
 *              Each state then calls an interface function, which may consist of several
 *              subfunctions, gathering keypad input from the user and performing a task
 *              before returning to the main menu.
*******************************************************************************************/
void program_state(){

    switch(state){
        case MAIN_MENU:
            main_menu_display();
            main_menu_interface();
            break;
        case DOOR:
            door_menu_display();
            door();
            break;
        case MOTOR:
            motor_menu_display();
            motor();
            break;
        case LIGHTS:
            lights_menu_display();
            lights();
            break;
    }
}
/*******************************************************************************************
 * Function:    main_menu_interface()
 * Description: Read the keypad for user input. User will choose Door, Motor, or Lights
 *              Menu.
*******************************************************************************************/
void main_menu_interface(){

    int i;
    i = read_single_digit(1, 4, 0);     //accept user input 1 - 3
    if (i == 1)
        state = DOOR;
    if (i == 2)
        state = MOTOR;
    if (i == 3)
        state = LIGHTS;
    lcdClear();
```

```
}
/*****************************************************************************************
 * Function:    door()
 * Description: Interface for the door menu.
 *              Receives user keypad input to open or close the door with the assistance
 *              of subfunctions door_open() and door_close().
 *              Displays the user's choice on the LCD before returning to main menu.
*****************************************************************************************/
void door(){
    int door_state;
    door_state = read_single_digit(1, 3, 0);    //accept input 1 or 2
    if (door_state == 1){
        door_open();                            //Manipulate servo
        lcdClear();
        SysTick_delay_ms(100);
        lcdSetText("DOOR OPEN", 0, 0);
        SysTick_delay_ms(1500);
        lcdClear();
        state = MAIN_MENU;
    }
    if (door_state == 2){
        door_close();                           //Manipulate servo
        lcdClear();
        SysTick_delay_ms(100);
        lcdSetText("DOOR CLOSED", 0, 0);
        SysTick_delay_ms(1500);
        lcdClear();
        state = MAIN_MENU;
    }
}
/*****************************************************************************************
 * Function:    door_open()
 * Description: Adjusts servo PWM to open the door
 *              Turn Red indicator LED off and Green LED on
*****************************************************************************************/
void door_open(){
    TIMER_A2->CCR[3] = 3100;                //Set Servo PWM to open door
    SysTick_delay_ms(500);                  //delay to allow door to open
    TIMER_A2->CCR[3] = 0;                   //disable PWM
    P3->OUT &= ~BIT7;                       //turn off Red LED
    P3->OUT |= BIT6;                        //turn on Green LED
}

/*****************************************************************************************
 * Function:    door_close()
 * Description: Adjusts servo PWM to close the door
 *              Turn Green indicator LED off and Red LED on
*****************************************************************************************/
void door_close(){
    TIMER_A2->CCR[3] = 6000;                //Set Servo PWM to close door
    SysTick_delay_ms(500);                  //delay to allow door to close
    TIMER_A2->CCR[3] = 0;                   //disable PWM
    P3->OUT &= ~BIT6;                       //turn off Green LED
    P3->OUT |= BIT7;                        //turn on Red LED
}

/*****************************************************************************************
 * Function:    motor()
 * Description: Interface for the door menu.
 *              Receives user keypad input to set the speed of the DC motor.
 *              Displays the user's choice on the LCD before returning to main menu.
```

```
 *              If duty cycle exceeds 80%, initiate a TA0 interrupt to toggle a blinking
 *              warning LED.
*******************************************************************************************/
void motor(){
    float motor_set;
    motor_set = read_single_digit(0, 10, 1);        //Accept input 0 - 9
    if(motor_set>=0){
        lcdSetText("SPEED", 0, 0);                  //Write user's choice to the LCD
        lcdSetInt(motor_set, 0, 1);
        SysTick_delay_ms(500);
        TIMER_A2->CCR[2] = (motor_set / 10) * 60000;//update DC motor duty cycle
        if (TIMER_A2->CCR[2] >= 48000)              //duty cycle >= 80%
            TIMER_A1->CTL |= (BIT4 | TIMER_A_CTL_CLR);  //initiate TA0 interrupt
        else TIMER_A1->CTL &= ~BIT4;                //duty cycle < 80%, disable interrupt
        SysTick_delay_ms(1500);
        lcdClear();
        state = MAIN_MENU;
    }
}
/*******************************************************************************************
 * Function:    lights()
 * Description: Interface for the lights menu.
 *              Receives user keypad input to choose the Red, Green, or Blue LED on the
 *              RGB module.
 *              Choice is stored as a variable that reflects which TA0->CCRn instance to
 *              update. Receives user input 0 - 100 to set the brightness level of the
 *              chosen LED. New duty cycle calculated and assigned to the proper CCRn
register.
 *              Duty cycles are global variables, allowing the CCRn registers to be
 *              toggled on and off with a button press.
 *              Displays the user's choice on the LCD before returning to main menu.
*******************************************************************************************/
void lights(){
    int i, j;
    float level;

    i = read_single_digit(1, 4, 0);                 //accept input 1 - 3
    lcdClear();
    for(j = 0; j < strlen(led_txt[i-1]); j++){      //write user's LED choice to the LCD
        lcdSetChar(led_txt[i-1][j], j, 0);
    }
    lcdSetText("SET LVL 0-100", 0, 1);              //Display instructions
    lcdSetText("[*] TO CLR ENTRY", 0, 2);
    SysTick_delay_ms(500);
    level = RGB_brightness();                       //read user entry
    RGB_DC[i] = (level / 100) * 3000;               //calculate new duty cycle
    if (RGB_DC[i] > 0) LED_check = 1;               //set flag for LED button
    TIMER_A0->CCR[i + 1] = RGB_DC[i];               //input duty cycle to CCR[i + 1] (starts
on CCR2)
    lcdClear();
    state = MAIN_MENU;
}
```

B. LCD_Libary.h

```
/*******************************************************************************************
*                               LCD_Library.h
*                               Trevor Ekin (Adapted from Dr. Nabeeh Kandalaft)
*                        EGR226      Date: February-21-2019
*
*  This is a library for the 4x16 LCD.
```

```
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
*  All pins are set with default values (see below) but they can be easily changed.  See
*  description below in "Pins" comment block
* ************************************************************************************/

/************************************************************************************
*Revision Log:
*   (v2.0) 6/15/2020: Reset Byte macros to make reset sequence a more understandable.
* ************************************************************************************/

#ifndef LCD_LIBRARY_H_
#define LCD_LIBRARY_H_

//#include "driverlib.h"  // for use with driverlib
#include "msp.h"
#include <stdint.h>

/************  Pins   ******************************
*          MSP432 PINS (Default)
 *********LCD*****************
 *-> GPIO                    *
 *  RS                  > P4.0
 *  E                   > P4.1
 *  DB4                 > P4.4
 *  DB5                 > P4.5
 *  DB6                 > P4.6
 *  DB7                 > P4.7
* NOTICE:
*   This section is configurable and dynamic.  If you
*   would like to use different pins, make the swaps
*   here.  For example, if you would like D5 to be on
*   P2.4 instead, make following changes:
*       D5_SEL1 P2SEL1
*       D5_SEL0 P2SEL0
*       D5_DIR P2DIR
*       D5_OUT P2OUT
*       D5 BIT4
* ************************************************/

// SEL1 registers (modify as needed)
#define EN_SEL1  P4SEL1
#define RS_SEL1  P4SEL1
#define RW_SEL1  P10SEL1
#define D4_SEL1  P4SEL1
#define D5_SEL1  P4SEL1
#define D6_SEL1  P4SEL1
#define D7_SEL1  P4SEL1

// SEL0 registers (modify as needed)
#define EN_SEL0  P4SEL0
#define RS_SEL0  P4SEL0
#define RW_SEL0  P10SEL0
#define D4_SEL0  P4SEL0
#define D5_SEL0  P4SEL0
#define D6_SEL0  P4SEL0
#define D7_SEL0  P4SEL0
```

```
// direction registers (modify as needed)
#define EN_DIR  P4DIR
#define RS_DIR  P4DIR
#define RW_DIR  P10DIR
#define D4_DIR  P4DIR
#define D5_DIR  P4DIR
#define D6_DIR  P4DIR
#define D7_DIR  P4DIR

// Port registers (modify as needed)
#define EN_OUT  P4OUT
#define RS_OUT  P4OUT
#define RW_OUT  P10OUT
#define D4_OUT  P4OUT
#define D5_OUT  P4OUT
#define D6_OUT  P4OUT
#define D7_OUT  P4OUT

// Pin BITs (modify as needed)
#define EN  BIT1
#define RS  BIT0
#define RW  BIT0
#define D4  BIT4
#define D5  BIT5
#define D6  BIT6
#define D7  BIT7

// Bit Toggling (DO NOT CHANGE)
#define EN_LOW (EN_OUT &= ~EN)
#define EN_HIGH (EN_OUT |= EN)
#define RS_LOW (RS_OUT &= ~RS)
#define RS_HIGH (RS_OUT |= RS)
#define RW_LOW (RW_OUT &= ~RW)
#define RW_HIGH (RW_OUT |= RW)
#define D4_LOW (D4_OUT &= ~D4)
#define D4_HIGH (D4_OUT |= D4)
#define D5_LOW (D5_OUT &= ~D5)
#define D5_HIGH (D5_OUT |= D5)
#define D6_LOW (D6_OUT &= ~D6)
#define D6_HIGH (D6_OUT |= D6)
#define D7_LOW (D7_OUT &= ~D7)
#define D7_HIGH (D7_OUT |= D7)

// Select Clearing (DO NOT CHANGE)
#define EN_GPIO (EN_SEL1 &= ~EN);(EN_SEL0 &= ~EN)
#define RS_GPIO (RS_SEL1 &= ~RS);(RS_SEL0 &= ~RS)
#define RW_GPIO (RW_SEL1 &= ~RW);(RW_SEL0 &= ~RW)
#define D4_GPIO (D4_SEL1 &= ~D4);(D4_SEL0 &= ~D4)
#define D5_GPIO (D5_SEL1 &= ~D5);(D5_SEL0 &= ~D5)
#define D6_GPIO (D6_SEL1 &= ~D6);(D6_SEL0 &= ~D6)
#define D7_GPIO (D7_SEL1 &= ~D7);(D7_SEL0 &= ~D7)

/// represent provided nibble on data lines use ternary statements and multi-line macro
/// *  note: ternary statements are like an "if / else" statement.
/// *      if/else example:
/// *
/// *      if(x & 0x01) {
/// *          D4_HIGH;
/// *      } else {
/// *          D4_LOW;
/// *      }
```

```
/// *
/// *      Ternary version with exact same result:
/// *      (x & 0x01) ? D4_HIGH : D4_LOW;
/// */
#define SetNibble(x) \
    ((x & 0x01) ? D4_HIGH : D4_LOW); \
    ((x & 0x02) ? D5_HIGH : D5_LOW); \
    ((x & 0x04) ? D6_HIGH : D6_LOW); \
    ((x & 0x08) ? D7_HIGH : D7_LOW);


/************************ Commands *********************************
*******************************************************************/
#define CLEAR       0x01
#define HOME        0x02

// RESET COMMANDS TO INITIALIZE DEVICE: Send 4 nibbles {3, 3, 3, 2} to restart device and set
it for 4-wire mode
#define RESET_BYTE_1    0x33    // Reset sequence, part 1 and 2
#define RESET_BYTE_2    0x32    // Reset sequence, part 3 and 4


// ENTRY MODE COMMAND: 0b0000 01[I][S] -- I = Increment, S = Shift
#define ENTRYMODE_00    0x04    // no auto increment, no display shift  (DEFAULT)
#define ENTRYMODE_01    0x05    // no auto increment, display shift
#define ENTRYMODE_10    0x06    // auto increment, no display shift
#define ENTRYMODE_11    0x07    // auto increment, display shift

// DISPLAY CONTORL COMMAND: 0b0000 1[D][C][B] -- D = Display, C = Cursor, B = Blinking
#define DISPLAY_000     0x08    // display off, cursor off, blinking off    (DEFAULT)
#define DISPLAY_001     0x09    // display off, cursor off, blinking on     (not useful)
#define DISPLAY_010     0x0A    // display off, cursor on, blinking off     (not useful)
#define DISPLAY_011     0x0B    // display off, cursor on, blinking on      (not useful)
#define DISPLAY_100     0x0C    // display on, cursor off, blinking off
#define DISPLAY_101     0x0D    // display on, cursor off, blinking on
#define DISPLAY_110     0x0E    // display on, cursor on, blinking off
#define DISPLAY_111     0x0F    // display on, cursor on, blinking on

// CURSOR/DISPLAY SHIFT COMMAND: 0b0001 [DC][RL] * * -- DC = Display or Cursor, RL = Right or
Left, * = don't care
#define SHIFT_00        0x10;   // cursor shift to the left
#define SHIFT_01        0x14;   // cursor shift to the right
#define SHIFT_10        0x18;   // display shift to the left
#define SHIFT_11        0x1C;   // display shift to the right

// FUNCTION SET COMMAND: 0b001[DL] [N][F] * * -- DL = Data length, N = display lines, F =
character font
#define FSET_000        0x20    // 4-bit data, 1 line, 5x8 dot font
#define FSET_001        0x24    // 4-bit data, 1 line, 5x10 dot font
#define FSET_010        0x28    // 4-bit data, 2 lines, 5x8 dot font
#define FSET_011        0x2C    // 4-bit data, 2 lines, 5x10 dot font
#define FSET_100        0x30    // 8-bit data, 1 line, 5x8 dot font     (standard)
#define FSET_101        0x34    // 8-bit data, 1 line, 5x10 dot font
#define FSET_110        0x38    // 8-bit data, 2 lines, 5x8 dot font
#define FSET_111        0x3C    // 8-bit data, 2 lines, 5x10 dot font

// CGRAM ADDRESS
#define CGRAM           0x40    // start address for CGRAM data, custom graphics

/************************ Commands *********************************
*******************************************************************/
```

```
/********************* Structure Definitions ***************************
*************************************************************************/
/// custom_char_t is a struct containing 8 bytes of data, representing the
///      8 rows of dots that create an LCD character. Each bit is either
///      on (1) or off (0) to display the image you desire.
/// You can make a custom character at (https://omerk.github.io/lcdchargen/)
typedef struct custom_char{
    uint8_t line0;
    uint8_t line1;
    uint8_t line2;
    uint8_t line3;
    uint8_t line4;
    uint8_t line5;
    uint8_t line6;
    uint8_t line7;
}custom_char_t;
/********************* Structure Definitions ***************************
*************************************************************************/

/********************** Global Definitions ****************************
*************************************************************************/
uint8_t _offset;                // offset in CGRAM for new custom characters
/********************** Global Definitions ****************************
*************************************************************************/

/********************** Function Prototypes ***************************
*************************************************************************/
void lcdInit ();
void lcdClear();
void lcdTriggerEN();
void lcdWriteData(unsigned char data);
void lcdWriteCmd (unsigned char  cmd);
void lcdSetText(char * text, int x, int y);
void lcdSetChar(char c, int x, int y);
void lcdSetInt (int val,     int x, int y);
uint8_t lcdCreateCustomChar(custom_char_t* cc);
/********************** Function Prototypes ***************************
*************************************************************************/



#endif /* LIQUID_CRYSTAL_H_ */
```

C. LCD_Library.c

```
/*************************************************************************************
 *                                LCD_Library.c
 *                                Trevor Ekin (Adapted from Dr. Nabeeh Kandalaft)
 *                          EGR226      Date: February-21-2019
 *
 *  This is a library for the 4x16 LCD.
 *
 *  All functions are briefly described in their comment blocks.  The /// notation makes
 *  it so the function description block is visible when you hover over a function call
 *  in any file (this feature is called Intellisense).
 *
 *  All pins are set with default values (see below) but they can be easily changed with
 *  in LCD_Library.h to any pin configuration (follow instructions in header file).
```

```
 *  Likely, these pins willnot work for you unless you've soldered on the additional
 *  header at the end opposing USB connection on your MSP.
 *
 ***************************  Pins   ***********************************************
 *                            MSP432 PINS (Default, see LCD_Library.h to change these)
 *                         *********LCD*****************
 *  RS                   > P4.0
 *  E                    > P4.1
 *  DB4                  > P4.4
 *  DB5                  > P4.5
 *  DB6                  > P4.6
 *  DB7                  > P4.7
 * ***************************************************************************************/

/***************************************************************************************
*Revision Log:
*   (v2.0) 6/15/2020: Reset Byte macros to make reset sequence a more understandable.
* ***************************************************************************************/

#include <stdint.h>
#include "LCD_Library.h"
#include <stdio.h>
#include "SysTick_Library.h"

/// ****| lcdInit | *********************************//*
/// * Brief: Initialize the LCD with chosen connection
/// *          pins.  Send configuration sequence.
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
void lcdInit() {
    // Initialize all communication pins as GPIOs by clearing SEL0 and SEL1 (see LCD_Library.h
for PSEL# macro definitions)
    EN_GPIO;
    RS_GPIO;
    RW_GPIO;
    D4_GPIO;
    D5_GPIO;
    D6_GPIO;
    D7_GPIO;

    // Initialize all communication pins as outputs (see LCD_Library.h for PDIR macro
definitions)
    RS_DIR |= RS;
    RW_DIR |= RW;
    EN_DIR |= EN;
    D4_DIR |= D4;
    D5_DIR |= D5;
    D6_DIR |= D6;
    D7_DIR |= D7;

    // Initialize all communication pins as low (see LCD_Library.h for Bit Toggling macro
definitions)
    RS_LOW;
    RW_LOW;
    EN_LOW;
    D4_LOW;
    D5_LOW;
    D6_LOW;
```

```
    D7_LOW;

    SysTick_delay_ms(100);  // Allow ample time for pin configuration to settle.

    //-------------------------------------------------------------------------------------
    // RESET COMMANDS TO INITIALIZE DEVICE: Send 4 nibbles {3, 3, 3, 2} to restart device and
set it for 4-wire mode
    //RESET_BYTE_1    0x33    -- Reset sequence, part 1 and 2
    //RESET_BYTE_2    0x32    -- Reset Sequence, part 3 and 4
    //-------------------------------------------------------------------------------------
    lcdWriteCmd(RESET_BYTE_1);      SysTick_delay_ms(100);  //reset sequence, part 1
    lcdWriteCmd(RESET_BYTE_2);      SysTick_delay_ms(100);  //reset sequence, part 2


    //-------------------------------------------------------------------------------------
    // FUNCTION SET COMMAND: 0b001[DL] [N][F] * * -> DL = Data length, N = display lines, F =
character font
    //FSET_000        0x20    -- 4-bit data, 1 line, 5x8 dot font
    //FSET_001        0x24    -- 4-bit data, 1 line, 5x10 dot font
    //FSET_010        0x28    -- 4-bit data, 2 lines, 5x8 dot font
    //FSET_011        0x2C    -- 4-bit data, 2 lines, 5x10 dot font
    //FSET_100        0x30    -- 8-bit data, 1 line, 5x8 dot font     (standard)
    //FSET_101        0x34    -- 8-bit data, 1 line, 5x10 dot font
    //FSET_110        0x38    -- 8-bit data, 2 lines, 5x8 dot font
    //FSET_111        0x3C    -- 8-bit data, 2 lines, 5x10 dot font
    //-------------------------------------------------------------------------------------
    lcdWriteCmd(FSET_010);      SysTick_delay_ms(100);  //reset sequence, part 3

    // HOME command is 0x02
    lcdWriteCmd(HOME);          SysTick_delay_ms(100);  //send cursor home

    //-------------------------------------------------------------------------------------
    // ENTRY MODE COMMAND: 0b0000 01[I][S] -> I = Increment, S = Shift
    //ENTRYMODE_00    0x04    // no auto increment, no display shift  (DEFAULT)
    //ENTRYMODE_01    0x05    // no auto increment, display shift
    //ENTRYMODE_10    0x06    // auto increment, no display shift
    //ENTRYMODE_11    0x07    // auto increment, display shift
    //-------------------------------------------------------------------------------------
    lcdWriteCmd(ENTRYMODE_10);  SysTick_delay_ms(100);  //set up for auto incrementing

    lcdWriteCmd(CLEAR);         SysTick_delay_ms(100);  //clear screen (again)

    //-------------------------------------------------------------------------------------
    // DISPLAY CONTORL COMMAND: 0b0000 1[D][C][B] -> D = Display, C = Cursor, B = Blinking
    //DISPLAY_000     0x08    -- display off, cursor off, blinking off    (DEFAULT)
    //DISPLAY_001     0x09    -- display off, cursor off, blinking on     (not useful)
    //DISPLAY_010     0x0A    -- display off, cursor on, blinking off     (not useful)
    //DISPLAY_011     0x0B    -- display off, cursor on, blinking on      (not useful)
    //DISPLAY_100     0x0C    -- display on, cursor off, blinking off
    //DISPLAY_101     0x0D    -- display on, cursor off, blinking on
    //DISPLAY_110     0x0E    -- display on, cursor on, blinking off
    //DISPLAY_111     0x0F    -- display on, cursor on, blinking on
    //-------------------------------------------------------------------------------------
    lcdWriteCmd(DISPLAY_111);   SysTick_delay_ms(100);  //turn on display with blinking cursor

    // CLEAR command is 0x01
    lcdWriteCmd(CLEAR);         SysTick_delay_ms(100);  //clear screen

    // reset CGRAM offset address
    _offset = 0;
}
```

```
/// ****| lcdTriggerEN | ***************************//*
/// * Brief: Pulse the enable pin to notify the LCD to
/// *          latch the current data inputs.
/// * param:
/// *      (unsigned char) data: 8-bit data to send
/// * return:
/// *      N/A
/// ************************************************/
void lcdTriggerEN() {
    EN_HIGH;
    SysTick_delay_us(50);
    EN_LOW;
    SysTick_delay_us(50);
}

/// ****| lcdWriteData | ***************************//*
/// * Brief: Send data one nibble at a time to LCD via
/// *          SetNibble macro (see LCD_Library.h)
/// * param:
/// *      (unsigned char) data: 8-bit data to send
/// * return:
/// *      N/A
/// ************************************************/
void lcdWriteData(unsigned char data) {
    RS_HIGH;
    RW_LOW;
    SysTick_delay_us(50);
    SetNibble(data >> 4);   // Upper nibble
    SysTick_delay_us(50);
    lcdTriggerEN();
    SetNibble(data);        // Lower nibble
    SysTick_delay_us(50);
    lcdTriggerEN();
    SysTick_delay_us(50);
    SetNibble(0x00);        // clear output
}

/// ****| lcdWriteCmd | ****************************//*
/// * Brief: Send command one nibble at a time to LCD
/// *          via SetNibble macro (see LCD_Library.h)
/// * param:
/// *      (unsigned char) cmd: 8-bit command to send
/// * return:
/// *      N/A
/// ************************************************/
void lcdWriteCmd(unsigned char cmd) {
    RS_LOW;
    RW_LOW;
    SysTick_delay_us(50);
    SetNibble(cmd >> 4);     // Upper nibble
    SysTick_delay_us(50);
    lcdTriggerEN();
    SetNibble(cmd);          // Lower nibble
    SysTick_delay_us(50);
    lcdTriggerEN();
    SysTick_delay_us(50);
    SetNibble(0x00);        // clear output
}

/// ****| lcdSetText | ****************************//*
```

```
/// * Brief: Display character string on the LCD at the
/// *          chosen coordinates.
/// * param:
/// *      (char*) text: character string to display
/// *      (int) x:      x-coordinate
/// *      (int) y:      y-coordinate
/// * return:
/// *      N/A
/// ************************************************/
void lcdSetText(char* text, int x, int y) {
    int i;
    if (x < 16) {
        x |= 0x80;      // Set LCD for first line write
        switch (y){
        case 0:
            x |= 0x00;  // Set LCD for first line write
            break;
        case 1:
            x |= 0x40;  // Set LCD for Second line write
            break;
        case 2:
            x |= 0x10;  // Set LCD for Third line write
            break;
        case 3:
            x |= 0x50;  // Set LCD for Fourth line write
            break;
        case 5:
            x |= 0x20;  // Set LCD for second line write reverse
            break;
        }
        lcdWriteCmd(x);
    }
    i = 0;
    while (text[i] != '\0') {
        lcdWriteData(text[i]);
        i++;
    }
}

/// ****| lcdSetChar | ****************************//*
/// * Brief: Display character on the LCD at the
/// *          chosen coordinates.
/// * param:
/// *      (char) c:     character to display (can be
/// *                       custom character if c = offset
/// *                       of custom character)
/// *      (int) x:      x-coordinate
/// *      (int) y:      y-coordinate
/// * return:
/// *      N/A
/// ************************************************/
void lcdSetChar(char c, int x, int y) {
    if (x < 16) {
        x |= 0x80;      // Set LCD for first line write
        switch (y){
        case 0:
            x |= 0x00;  // Set LCD for first line write
            break;
        case 1:
            x |= 0x40;  // Set LCD for Second line write
            break;
```

```
        case 2:
            x |= 0x10;  // Set LCD for Third line write
            break;
        case 3:
            x |= 0x50;  // Set LCD for Fourth line write
            break;
        case 5:
            x |= 0x20;  // Set LCD for second line write reverse
            break;
        }
        lcdWriteCmd(x);
    }

    lcdWriteData(c);
}

/// ****| lcdSetInt | *****************************//*
/// * Brief: Convert integer into character string to be
/// *          displayed on LCD at chosen coordinates.
/// * param:
/// *      (int) val: value to convert to display
/// *      (int) x:   x-coordinate
/// *      (int) y:   y-coordinate
/// * return:
/// *      N/A
/// ************************************************/
void lcdSetInt(int val, int x, int y){
    char number_string[16];
    sprintf   (number_string, "%d\0", val); // Convert the integer to character string
    lcdSetText(number_string, x, y);
}

/// ****| lcdCreateCustomChar | *********************//*
/// * Brief: Creates a custom character in CGRAM based on
/// *          character structure passed.
/// * param:
/// *      (custom_char_t)* cc: custom character struct
/// *                           to place in CGRAM
/// * return:
/// *      (uint8_t) _offset:   offset index of new
/// *                             custom char
/// ************************************************/
uint8_t lcdCreateCustomChar(custom_char_t* cc) {
    lcdWriteCmd(CGRAM+(8*_offset));     // characters placed in intervals of 8 bytes
    lcdWriteData(cc->line0);            // send byte 0 of new character
    lcdWriteData(cc->line1);            // send byte 1 of new character
    lcdWriteData(cc->line2);            // send byte 2 of new character
    lcdWriteData(cc->line3);            // send byte 3 of new character
    lcdWriteData(cc->line4);            // send byte 4 of new character
    lcdWriteData(cc->line5);            // send byte 5 of new character
    lcdWriteData(cc->line6);            // send byte 6 of new character
    lcdWriteData(cc->line7);            // send byte 7 of new character
    return _offset++;                   // return then increment offset value for next
character
}


/// ****| lcdClear | *****************************//*
/// * Brief: Clear all visible characters from the
/// *          screen.
/// * param:
```

```
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
void lcdClear() {
    lcdWriteCmd(CLEAR);
    SysTick_delay_ms(10);
}
```

D. LCD_GUI.h

```
/*****************************************************************************
 *                              LCD_GUI.h
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Contains the various displays to be written to the LCD for the primary menus
 *****************************************************************************/


#ifndef LCD_GUI_H_
#define LCD_GUI_H_

#include "msp.h"
#include "LCD_Library.h"

void startup_display();
void main_menu_display();
void door_menu_display();
void motor_menu_display();
void lights_menu_display();

#endif /* LCD_GUI_H_ */
```

E. LCD_GUI.c

```
/*****************************************************************************
 *                              LCD_GUI.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Contains the various displays to be written to the LCD for the primary menus
 *****************************************************************************/

#include "LCD_Library.h"
#include "SysTick_Library.h"

/*****************************************************************************
* Function     startup_display()
* Description  Initial text and instructions on the LCD
*****************************************************************************/
void startup_display(){

    lcdSetText("MAIN MENU", 0, 0);
    SysTick_delay_ms(1000);
    lcdSetText("MAKE SELECTION", 0, 1);
    lcdSetText("THEN PRESS [#]", 0, 2);
    SysTick_delay_ms(3000);
    lcdClear();
}
```

```
/*****************************************************************************************
* Function     main_menu_display()
* Description  Display menu options
*****************************************************************************************/
void main_menu_display(){
    lcdSetText("[1]DOOR MENU", 0, 0);
    lcdSetText("[2]MOTOR MENU", 0, 1);
    lcdSetText("[3]LIGHTS MENU", 0, 2);
}

/*****************************************************************************************
* Function     door_menu_display()
* Description  Display menu options
*****************************************************************************************/
void door_menu_display(){
    lcdSetText("DOOR MENU", 0, 0);
    lcdSetText("[1] TO OPEN", 0, 1);
    lcdSetText("[2] TO CLOSE", 0, 2);
}

/*****************************************************************************************
* Function     motor_menu_display()
* Description  Display menu options
*****************************************************************************************/
void motor_menu_display(){
    lcdSetText("MOTOR MENU", 0, 0);
    lcdSetText("SET SPEED 0-9", 0, 1);
}

/*****************************************************************************************
* Function     lights_menu_display()
* Description  Display menu options
*****************************************************************************************/
void lights_menu_display(){
    lcdSetText("LIGHTS MENU", 0, 0);
    lcdSetText("RED GRN BLU", 0, 1);
    lcdSetText("[1] [2] [3]", 0, 2);
}
```

F. Keypad_Library.c

```
/*****************************************************************************************
 *                              Keypad_Library.h
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Function declarations for initializing and reading the keypad.
 *****************************************************************************************/


#ifndef KEYPAD_LIBRARY_H_
#define KEYPAD_LIBRARY_H_

void keypad_init();
int read_keypad();
```

```
#endif /* KEYPAD_LIBRARY_H_ */
```

G. keypad_init.c

```
/*******************************************************************************
 *                              keypad_init.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Initializes the pins for reading the keypad
 *******************************************************************************/

/*Keypad initialization function.
 * Row 1:       P3.0
 * Row 2:       P3.2
 * Row 3:       P3.3
 * Row 4:       P3.5
 *
 * Column 1:    P1.5
 * Column 2:    P1.6
 * Column 3:    P1.7
 */

//------------------------------------------------------ Library Includes
#include "Keypad_Library.h"
#include "msp.h"
//------------------------------------------------------ Library Includes

void keypad_init(){

uint8_t col_mask = 0b11100000;      //mask for column keypad pins
uint8_t row_mask = 0b00101101;      //mask for row keypad pins

/* Set up Columns*/
P1->SEL1 &= ~col_mask;              //clear col bits for GPIO
P1->SEL0 &= ~col_mask;              //clear col for GPIO
P1->DIR &= ~col_mask;               //clear all col bits initially for input

/* Set up Rows */
P3->SEL1 &= ~row_mask;              //clear row bits for GPIO
P3->SEL0 &= ~row_mask;              //clear row bits for GPIO
P3->DIR &= ~row_mask;               //clear row bits initially for input
P3->REN |= row_mask;                //enable resistors on row pins
P3->OUT |= row_mask;                //enable pull-up resistors on row pins

}
```

H. keypad_read.c

```
/*******************************************************************************
 *                              keypad_init.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Initializes the pins for reading the keypad
 *******************************************************************************/

/*Keypad initialization function.
 * Row 1:       P3.0
 * Row 2:       P3.2
```

```
 * Row 3:       P3.3
 * Row 4:       P3.5
 *
 * Column 1:    P1.5
 * Column 2:    P1.6
 * Column 3:    P1.7
 */

//------------------------------------------------------- Library Includes
#include "Keypad_Library.h"
#include "msp.h"
//------------------------------------------------------- Library Includes

void keypad_init(){

uint8_t col_mask = 0b11100000;      //mask for column keypad pins
uint8_t row_mask = 0b00101101;      //mask for row keypad pins

/* Set up Columns*/
P1->SEL1 &= ~col_mask;              //clear col bits for GPIO
P1->SEL0 &= ~col_mask;              //clear col for GPIO
P1->DIR &= ~col_mask;               //clear all col bits initially for input

/* Set up Rows */
P3->SEL1 &= ~row_mask;              //clear row bits for GPIO
P3->SEL0 &= ~row_mask;              //clear row bits for GPIO
P3->DIR &= ~row_mask;               //clear row bits initially for input
P3->REN |= row_mask;                //enable resistors on row pins
P3->OUT |= row_mask;                //enable pull-up resistors on row pins

}
```

I. User_Keypad_Input.h

```
/*
 * User_Keypad_Input.h
 *
 *  Created on: Jul 26, 2020
 *      Author: dstnm
 */

#ifndef USER_KEYPAD_INPUT_H_
#define USER_KEYPAD_INPUT_H_

#include "Keypad_Library.h"
#include "SysTick_Library.h"

float RGB_brightness();
int read_single_digit(int min, int max, int zero_enable);

#endif /* USER_KEYPAD_INPUT_H_ */
```

J. User_Keypad_Input.c

```
/*****************************************************************************
 *                              User_Keypad_Input.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Functions included to read either single digits or a three digit entry
 *              on the keypad. Includes error checking.
 *              Single digit entry will update a single digit array until the user presses
```

```
 *              the # key to make their choice.
 *              The RGB_brightness() function will update a three digit array until the
 *              user presses the # key to make their choice.
 ********************************************************************************/
#include "msp.h"
#include "Keypad_Library.h"
#include "LCD_Library.h"
#include "SysTick_Library.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*********************************
 *** Char array for keypad input ***
 *********************************/
char keys[14] = {'x', '1', '2', '3', '4', '5', '6', '7', '8', '9', '*',  '0', '#'};

/*******************************************************************************
 * Function:    read_single_digit()
 * Description: Read the keypad and print each key pressed to the LCD. If the entry is
 *              acceptable, return the value as an integer. If the user pushes the [#] key
 *              on an invalid entry outside the given range, an error message is displayed
 *              and the user must make a new entry. The user can also clear their current
 *              entry with the [*] key.
 * Inputs:      min, max, and zero enable. Min and max allow the function to
 *              accept an entry into the select array only if it is in the specified range.
 *              The [0] key lies between the [*] and [#] keys, so the zero enable variable
 *              allows the function to accept a press of the [0] key into the select array.
 *
*******************************************************************************/
int read_single_digit(int min, int max, int zero_enable){
    int i = 0, val = -1;
    char select[2] = {'0', '\0'};               //array to store the last valid keypad press
       while(1){
           i = read_keypad();
           if (i > 0)
               lcdSetChar(keys[i], 0, 3);       //write entry to the LCD
           if ((i > 0) && (i != 10) && (i != 12)){
               select[0] = keys[i];
               val = atoi(select);              //store the value as an integer
           }
           else if ((zero_enable) && (i == 11)){
               select[0] = '0';
               val = atoi(select);
           }
           if (i == 10){                            // [*] to clear entry
               memset(select, '0', strlen(select)); //clear the array
               val = atoi(select);
               lcdSetText("CLEARING...", 0, 3);
               SysTick_delay_ms(500);
               lcdSetText("           ", 0, 3);
           }
           if ((i == 12) && (val >= min) && (val < max)){   //value is in range
             lcdClear();
             lcdSetText("YOU ENTERED", 0, 0);
             lcdSetChar(select[0], 0, 1);
             SysTick_delay_ms(2000);
             lcdClear();
             return val;                                    //return the user selection
           }
           else if ((i == 12) && (val < min) | (val >= max)){
```

```
                lcdSetText("INVALID ENTRY", 0, 3);
                SysTick_delay_ms(2000);
                lcdSetText("             ", 0, 3);
            }
        }
}

float RGB_brightness(){
    int i = 0, brite;
    char RGB[4] = {'0', '0', '0', '\0'};

    while(1){
        i = read_keypad();
        if ((i > 0) && (i != 10) && (i != 12)){
            RGB[0] = RGB[1];
            RGB[1] = RGB[2];
            RGB[2] = keys[i];
            brite = atoi(RGB);
            lcdSetInt(brite, 0, 3);
        }
        if (i == 10){                           // [*] to clear entry
            memset(RGB, '0', strlen(RGB));
            brite = atoi(RGB);
            lcdSetText("CLEARING...", 0, 3);
            SysTick_delay_ms(500);
            lcdSetText("           ", 0, 3);
        }
        if ((i == 12) && (brite <= 100)){
            lcdClear();
            SysTick_delay_ms(500);
            lcdSetText("LED LVL: ", 0, 0);
            lcdSetInt(brite, 0, 1);
            SysTick_delay_ms(1500);
            lcdClear();
            return brite;
        }
        else if ((i == 12) && (brite > 100)){
            lcdSetText("ERROR- TRY AGAIN", 0, 3);
            memset(RGB, '0', strlen(RGB));
            brite = atoi(RGB);
            SysTick_delay_ms(1000);
            lcdSetText("                ", 0, 3);
        }
    }
}
```

K. SysTick_Library.h

```
/*****************************************************************************
*                               SysTick_Library.h
*                          Trevor Ekin / Nabeeh Kandalaft
*                        EGR226      Date: March, 6, 2019
*
*  This is a library for the SysTick Timer Peripheral on the MSP432.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
*
* ****************************************************************************/
```

```
#ifndef SYSTICK_LIBRARY_H_
#define SYSTICK_LIBRARY_H_

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

/******************** Macro Prototypes ******************************
 * SysTick Control and Status Register (STCSR) as discussed in lectures
*********************************************************************/
#define STCSR_CLKSRC    (0x0004)        // This is the CLKSOURSE bit, BIT2
#define STCSR_INT_EN    (0x0002)        // This is the TICKINT  bit, BIT1
#define STCSR_EN        (0x0001)        // This is the ENABLE bit, BIT0
/******************** Macro Prototypes ******************************
*********************************************************************/



/********************* Function Prototypes ***************************
*********************************************************************/
void SysTickInit_NoInterrupts  (void);
void SysTickInit_WithInterrupts(uint32_t);
void SysTick_delay_ms(volatile uint32_t);
void SysTick_delay_us(volatile uint32_t);
/********************* Function Prototypes ***************************
*********************************************************************/

/******************** Global Flags ***************************
*********************************************************************/
volatile uint8_t _flag;
/******************** Global Flags ***************************
*********************************************************************/
#endif /* SYSTICK_LIBRARY_H_ */
```

L. SysTick_Library.c

```
/*****************************************************************************
*                              SysTick_Library.c
*                         Trevor Ekin / Nabeeh Kandalaft
*                       EGR226      Date: March, 6, 2019
*
*  This is a library for the SysTick Timer Peripheral on the MSP432.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
*
* ***************************************************************************/
#include "SysTick_Library.h"
#include "msp.h"

/// ****| SysTickInit_NoInterrupts | ***************//*
/// * Brief: Initialize the SysTick peripheral for use
/// *          without interrupts (busy-waits)
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
```

```
void SysTickInit_NoInterrupts(void){
    SysTick->CTRL  &= ~BIT0;                            //clears enable to stop the counter
    SysTick->LOAD   = 0x00FFFFFF;                       //sets the period... note:
(3006600/1000 - 1) = 1ms
    SysTick->VAL    = 0;                                //clears the value
    SysTick->CTRL   = (STCSR_CLKSRC | STCSR_EN);        //enable SysTick with core clock, no
interrupts -> this is the ENABLE and CLKSOURSE bits: Systic->CTRL |= 0x05;
}

/// ****| SysTickInit_WithInterrupts | *************//*
/// * Brief: Initialize the SysTick peripheral for use
/// *          with interrupts (interrupt delays)
/// * param:
/// *      (uint32_t) ms_delay:  number of milliseconds
/// *                            to delay
/// * return:
/// *      N/A
/// ************************************************/
void SysTickInit_WithInterrupts(uint32_t delay_ms){
    SysTick->CTRL  &= ~BIT0;                                        //clears enable to
stop the counter
    SysTick->LOAD   = delay_ms * 3000;                              //sets the period
    SysTick->VAL    = 0;                                            //clears the value
    SysTick->CTRL   = (STCSR_CLKSRC | STCSR_INT_EN | STCSR_EN);     //enable SysTick with
core clock, interrupts on -> this is the ENABLE, TICKINT, and CLKSOURSE bits: Systic->CTRL |=
0x07;
}

/// ****| SysTick_delay_ms | ************************//*
/// * Brief: Use the SysTick timer to delay a specified
/// *          number of milliseconds
/// * param:
/// *      (uint32_t) ms_delay:  number of milliseconds
/// *                            to delay
/// * return:
/// *      N/A
/// ************************************************/
void SysTick_delay_ms(uint32_t ms_delay){
    //Delays time_ms number of milliseconds
    //Assume 3MHz clock -> 3000 cycles per millisecond
    SysTick->LOAD  = 3000 * (uint32_t)ms_delay;
    SysTick->VAL   = 0;                                 // starts counting from 0
    SysTick->CTRL |= (STCSR_CLKSRC | STCSR_EN);         // ENABLE, CLKSOURSE bits  .... Systic-
>CTRL |= 0x05;
    while(!(SysTick->CTRL & ((uint32_t)1)<<16));        // Continue while bit 16 is high or use
....while( (SysTick->CTRL & BIT16) == 0);
    SysTick->CTRL &= ~(STCSR_CLKSRC | STCSR_EN);        // Disable the Systic timer
.... Systic->CTRL =0 ;
}

/// ****| SysTick_delay_us | ************************//*
/// * Brief: Use the SysTick timer to delay a specified
/// *          number of microseconds
/// * param:
/// *      (uint32_t) us_delay:  number of microseconds
/// *                            to delay
/// * return:
/// *      N/A
/// ************************************************/
void SysTick_delay_us(uint32_t us_delay){
    //Delays time_ms number of milliseconds
```

```
    //Assume 3MHz clock -> 3 cycles per microsecond
    SysTick->LOAD  = us_delay*3 - 1;                //counts up to delay
    SysTick->VAL   = 0;                             //starts counting from 0
    SysTick->CTRL |= (STCSR_CLKSRC | STCSR_EN);     // ENABLE, CLKSOURSE bits  .... Systic-
>CTRL |= 0x05;
    while(!(SysTick->CTRL & ((uint32_t)1)<<16));    // Continue while bit 16 is high   ....
while( (SysTick->CTRL & BIT16) == 0);
    SysTick->CTRL &= ~(STCSR_CLKSRC | STCSR_EN);    // Disable the Systic timer        ....
Systic->CTRL =0 ;
}

/// ****| SysTick_Handler | *************************//*
/// * Brief: SysTick Handler (rewrite for desired use)
/// *
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
void SysTick_Handler(void) {
    _flag = 1;
}
```

M. Button_Pot_Init.h

```
/*****************************************************************************
 *                              Button_Pot_Init.h
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Functions included to initialize pins utilizing interrupts activated by
 *              external components, i.e. buttons and a potentiometer.
 *****************************************************************************/

#ifndef BUTTON_POT_INIT_H_
#define BUTTON_POT_INIT_H_

void init_buttons();
void LCD_Pot();

#endif /* BUTTON_POT_INIT_H_ */
```

N. Button_Pot_Init.c

```
/*****************************************************************************
 *                              Button_Pot_Init.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Functions included to initialize pins utilizing interrupts activated by
 *              external components, i.e. buttons and a potentiometer.
 *****************************************************************************/

#include "msp.h"
#include "Button_Pot_Init.h"

/*****************************************************************************
 * Function:    init_buttons()
 * Description: Set up button ports for GPIO and port interrupts
*****************************************************************************/
```

```
void init_buttons(){

    /* LED Port Interrupt Button P2.3 */
    P2->SEL1 &= ~BIT3;              //Set as GPIO
    P2->SEL0 &= ~BIT3;              //Set as GPIO
    P2->DIR &= ~BIT3;               //Set buttons as input
    P2->REN |= BIT3;                //Enable Internal resistance
    P2->OUT |=BIT3;                 //Set pull-up resistor
    P2->IES |=BIT3;                 //Set pin interrupt to trigger at 1>0 transition
    P2->IE |= BIT3;                 //Enable interrupt for button pins
    P2->IFG = 0;                    //Clear all P6 interrupt flags

    /* Motor Port Interrupt Button P5.5 */
    P5->SEL1 &= ~BIT5;              //Set as GPIO
    P5->SEL0 &= ~BIT5;              //Set as GPIO
    P5->DIR &= ~BIT5;               //Set buttons as input
    P5->REN |= BIT5;                //Enable Internal resistance
    P5->OUT |=BIT5;                 //Set pull-up resistor
    P5->IES |=BIT5;                 //Set pin interrupt to trigger at 1>0 transition
    P5->IE |= BIT5;                 //Enable interrupt for button pins
    P5->IFG = 0;                    //Clear all P5 interrupt flags

}

/*******************************************************************************
 * Function:    LCD_Pot()
 * Description: Set up ADC15 on Pin P6.0 with interrupts to read the potentiometer when
 *              a conversion is made.
*******************************************************************************/
void LCD_Pot(){

    /* Read Pin for the Potentiometer (A15) */
    P6->SEL1 |= BIT0;                           //Tertiary Function
    P6->SEL0 |= BIT0;                           //Tertiary Function
    P6->DIR &=~ BIT0;                           //Configure for input

    /* Initialize the ADC14 to read the potentiometer */
    ADC14->CTL0 &=~ ADC14_CTL0_ENC;             //disable ADC converter
    ADC14->CTL0 |= 0x44540210;                  //Set/hold pulse mode, SMCLK, 16clks,
continuous sampling, ADC on
    ADC14->CTL1 = 0x000F0030;                   //14 bit resolution, store in mem15 register
    ADC14->MCTL[15] =0x0000000F;                //channel 15 for mem15 interrupts
    ADC14->IER0 |= 0x00008000;                  //enable interrupts on mem15
    ADC14->CTL0 |= ADC14_CTL0_ENC;              //enable ADC14
}
```

O. Interrupt_Handlers.h

```
/*******************************************************************************
 *                              Interrupt_Handlers.h
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Defines the various interrupt handlers used in the program. These include
 *              port, TA, and ADC types.
 *
 *              Initializes global variables to be used in the ISR's.
 *******************************************************************************/

#ifndef INTERRUPT_HANDLERS_H_
```

```
#define INTERRUPT_HANDLERS_H_

#include "msp.h"
#include <stdlib.h>

void PORT2_IRQHandler();
void PORT5_IRQHandler();
void TA1_0_IRQHandler();
void TA3_0_IRQHandler();
void ADC14_IRQHandler(void);
void T32_INT2_IRQHandler();

/* Duty Cycle RED = RGB_DC[1], GRN = RGB_DC[2], BLU = RGB_DC[3] */
extern float RGB_DC[5];
extern double read;
extern uint16_t lcd_dc;
extern int LED_check;


#endif /* INTERRUPT_HANDLERS_H_ */
```

P. Interrupt_Handlers.c

```
/*****************************************************************************
 *                              Interrupt_Handlers.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Defines the various interrupt handlers used in the program. These include
 *              port, TA, and ADC types.
 *****************************************************************************/
#include "msp.h"
#include "Interrupt_Handlers.h"
#include <stdlib.h>
#include <stdio.h>

/*********************************
 ***** Define global variables *****
 *********************************/
float RGB_DC[5] = {0, 0, 0, 0};         //duty cycles for the RGB module
double read = 0;                        //reading from the ADC potentiometer
uint16_t lcd_dc = 0;                    //duty cycle for the LCD
int LED_check = 0;                      //check variable for LED button
int screen_save_digit = 0;

/*****************************************************************************
 * Function     PORT5_IRQHandler()
 * Description  Trigger TA3 interrupt to debounce P2.3 button and toggle the RGB LED module
 *****************************************************************************/
void PORT2_IRQHandler(){

    TIMER_A3->CTL |= BIT4;                      //Set TA3 to UP MODE
    TIMER_A3->CTL |= TIMER_A_CTL_CLR;           //restart TA3 count
    P2->IFG = 0;                                //reset all P2 flags
}


/*****************************************************************************
* Function     PORT5_IRQHandler()
* Description  Trigger TA3 interrupt to debounce buttons on P5.5 or P2.3
```

```
*******************************************************************************/
void PORT5_IRQHandler(){

    TIMER_A3->CTL |= BIT4;                      //Set TA3 to UP MODE
    TIMER_A3->CTL |= TIMER_A_CTL_CLR;           //restart TA3 count
    P5->IFG = 0;                                //reset all P5 flags
}

/*******************************************************************************
* Function      TA1_0_IRQHandler()
* Description   Activated when duty cycle of the DC motor reaches or exceeds 80%, toggle an
*               LED every .25 seconds.
*******************************************************************************/
void TA1_0_IRQHandler(){
    TIMER_A1->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;  //clear interrupt flag
    P5->OUT ^= BIT0;                            //toggle warning LED
}

/*******************************************************************************
* Function      TA3_0_IRQHandler()
* Description   Debounce timer for buttons.
*               If the DC motor killswitch is pressed, set the Duty Cycle to 0.
*
*               If the RGB toggle button is pressed, manipulate the CCRn registers
*               to toggle the RGB module while retaining the duty cycle values, which
*               are saved as global variables.
*******************************************************************************/
void TA3_0_IRQHandler(){
    int i;

    TIMER_A3->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;  //clear interrupt flag
    if((P5->IN & BIT5) == 0){                   //If switch is still low, update press
variable
        TIMER_A2->CCR[2] = 0;                   //Set DC motor duty cycle to 0
        TIMER_A1->CTL &= ~BIT4;                 //disable warning LED Timer
        P5->OUT &= ~BIT0;                       //turn off warning LED
    }
    if((P2->IN & BIT3) == 0){                   //RGB button pressed, manipulate CCRn
registers
        if (LED_check){                         //if any LEDs have been set, toggle all off
            for (i = 2; i < 5; i++){
                TIMER_A0->CCR[i] = 0;
            }
            LED_check = 0;
        }
        else{
            for (i = 2; i < 5; i++){            //else toggle LEDs all on
            TIMER_A0->CCR[i] = RGB_DC[i - 1];
            }
            LED_check = 1;
        }

    }
    TIMER_A3->CTL &= ~BIT4;                     //Halt TA3
    TIMER_A3->CTL |= TIMER_A_CTL_CLR;           //Clear TA3R count
}

/*******************************************************************************
 * Function     ADC14_IRQHandler()
 * Description  Read potentiometer if a conversion is made, adjust LCD brightness
 *******************************************************************************/
```

```
void ADC14_IRQHandler(void){

    ADC14->CLRIFGR0 |= 0x00008000;
    read = ADC14->MEM[15];                          //read mem[15] register
    lcd_dc = (read / 16384) * 60000;                //calculate new duty cycle using the 14-bit
resolution value
    if (lcd_dc < 100)                               //if duty cycle < 100, turn off LCD (avoids
sputtering activation)
        lcd_dc = 0;
    TIMER_A2->CCR[4] = lcd_dc;                      //adjust LCD brightness
    ADC14->CTL0 |= ADC14_CTL0_SC;
}
```

Q. Timer_A_Library.h

```
/*****************************************************************************
 *                              TIMER_A_Library.h
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Functions included to initialize Timer A channels and (if applicable)
 *              their associated pins.
 *****************************************************************************/

#ifndef TIMER_A_LIBRARY_H_
#define TIMER_A_LIBRARY_H_

#include "msp.h"

void init_TA0_Pins();
void init_TA1();
void init_TA2_Pins();
void TIMER_A3_delay();


#endif /* TIMER_A_LIBRARY_H_ */
```

R. Timer_A_Library.c

```
/*****************************************************************************
 *                              TIMER_A_Library.c
 *Author        Dustin Matthews
 *Date          7/26/20
 *Instructor    Professor Ekin
 *Description   Functions included to initialize Timer A channels and (if applicable)
 *              their associated pins.
 *****************************************************************************/

#include <msp.h>
#include <SysTick_Library.h>

/*****************************************************************************
 * Function:    init_TA0_Pins()
 * Description: Set up three channels of TimerA0 to drive PWM levels for the RGB LED module
*****************************************************************************/
void init_TA0_Pins(){

    /* TA0 */
    TIMER_A0->CTL = 0b0000001000010100;         //SMCLK | UP Mode | TACLR
```

```
    TIMER_A0->CCR[0] = 3000;                    //PWM Period (1000ms / 1 sec) * (1 sec /
1000Hz ) = 1ms Period
                                                //(3000 clock cycles / second) * 1ms = 3000
clock ticks per Period
    /* PWM Pins for RGB LED Module */
    uint8_t pin_mask = 0b11100000;
    P2->SEL1 &= ~pin_mask;
    P2->SEL0 |= pin_mask;
    P2->DIR |= pin_mask;

    /* Red TA0.2 */
    TIMER_A0->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A0->CCR[2] = 0;                       //RED Duty Cycle - Start 0%

    /* Green TA0.3 */
    TIMER_A0->CCTL[3] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A0->CCR[3] = 0;                       //GRN Duty Cycle - Start 0%

    /* Blue TA0.4 */
    TIMER_A0->CCTL[4] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A0->CCR[4] = 0;                       //BLU Duty Cycle - Start 0%
}

/*******************************************************************************************
*Function:      init_TA1()
*Description:   Set up TA1 for quarter second interrupts using SMCLK in UP mode
*               SMCLK default 3MHz -> /64 to get 46.875 clock ticks per ms
*               46.875 * 250ms = 11718 period, load this into CCR[0]
*               Use CCTL[0] interrupt
*******************************************************************************************/
void init_TA1(){

        TIMER_A1->CCR[0] = 11718;
        TIMER_A1->CTL = 0b0000001011000000;     //Source: SMCLK, Halted, ID /8
        TIMER_A1->EX0 = 0b0000000000000111;     //divide clock further by 8
        TIMER_A1->CCTL[0] = TIMER_A_CCTLN_CCIE;
}

/*******************************************************************************************
 * Function:    init_TA2_Pins()
 * Description: Set up three channels of TimerA2 to drive PWM levels for the DC motor,
 *              Servo, and LCD brightness
*******************************************************************************************/
void init_TA2_Pins(){

    /* TA2 */
    TIMER_A2->CTL = 0b0000001000010100;         //SMCLK | UP Mode | TACLR

    TIMER_A2->CCR[0] = 60000;                   //PWM Period (1000ms / 1 sec) * (1 sec / 50Hz
) = 20ms Period
                                                //(3000 clock cycles / second) * 20ms = 60000
clock ticks per Period
    /* PWM Pin for DC Motor,  */
    P5->SEL1 &= ~BIT7;
    P5->SEL0 |= BIT7;
    P5->DIR |= BIT7;
    P5->OUT &= ~BIT7;

    /* Motor Warning LED Pin */
    P5->SEL0 &= ~BIT0;                          //Set for GPIO
    P5->SEL1 &= ~BIT0;
```

```
    P5->DIR |= BIT0;                            //Set for output
    P5->OUT &= ~BIT0;                           //start with LED off

    /* TA2.2 */
    TIMER_A2->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A2->CCR[2] = 0;                       //DC Motor Duty Cycle - Start 0%

    /* PWM Pin and R/G LEDs for Servo Door */
    P6->SEL1 &= ~BIT6;
    P6->SEL0 |= BIT6;
    P6->DIR |= BIT6;                            //PWM PIN

    P3->SEL0 &= ~(BIT6 | BIT7);
    P3->SEL1 &= ~(BIT6 | BIT7);
    P3->DIR |= (BIT6 | BIT7);                   //R/G LEDs
    P3->OUT &= ~BIT6;                           //turn off Green LED

    /* TA2.3 */
    TIMER_A2->CCTL[3] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A2->CCR[3] = 6000;                    //Servo Duty Cycle - Start Closed
    SysTick_delay_ms(100);
    TIMER_A2->CCR[3] = 0;                       //Set Duty Cycle to 0

    P2->OUT |= BIT0;                            //Turn on RED LED to indicate door closed

    /* PWM Pin for LCD Brightness */
    P6->SEL1 &= ~BIT7;
    P6->SEL0 |= BIT7;
    P6->DIR |= BIT7;

    /* TA2.4 */
    TIMER_A2->CCTL[4] = TIMER_A_CCTLN_OUTMOD_7; //Output mode-> Reset/set
    TIMER_A2->CCR[4] = 45000;                   //LCD Brightness Duty Cycle - Start 75%
}

/*******************************************************************************
 * Function:    TIMER_A3_delay()
 * Description: Set up TimerA3 with interrupt to use as a 10ms debounce of the port buttons
*******************************************************************************/
void TIMER_A3_delay(){

    TIMER_A3->CTL = 0b0000001000000100;         //SMCLK | HALTED | TACLR
    TIMER_A3->CCTL[0] = TIMER_A_CCTLN_CCIE;     //Interrupts enabled
    TIMER_A3->CCR[0] = 30000;                   //debounce 10ms: 3000 cycles per ms * 10 ms
}
```