Grand Valley State University
School of Engineering

Project Design Document

Project: EGR 326 Embedded System Design and Build- Automotive Style Dashboard

Prepared By: Jordan Hayes

Submitted to: Dr. Kandalaft
EGR 326 Term Project
Fall Term 2020
12/15/2020

Contents

Abstract

The goal of the final design project for EGR 326 is to design, build, and program a vehicular, automotive style dashboard. The heart of the project is the TI MSP432 launchpad microcontroller, creating an embedded system to interface a real time clock IC, a stepper motor, and several sensors and pushbuttons, flash memory chip, as well as a rotary encoder and LCD display screen. These components were used to create user interfaceable menus, a digital and analog speedometer, output warnings to the user, record alarms, initiate turn signals, and collect input to view and access data stored within the system.

1.0 Introduction and Design Background

The goal of the final design project is to fulfil the need for a highly robust automotive style dashboard. The heart of the project is the TI MSP432 launchpad microcontroller that is expected to interface with a real time clock IC and several pushbutton switches and sensors. The microcontroller also collects input from a rotary encoder and the collective input from the pushbutton switches expected to format the display for time, date, and temperature information, as well as report the status of the system collected from various sensors. To achieve these goals, the TI MSP432 with a 48MHz ARM® Cortex®-M4F, 80uA/MHz active power and 660nA RTC operation, SAR Precision ADC with 16-bit performance was used to to create loops and timing intervals that periodically checked the status of the pushbuttons and sensors to update various components that implemented either I2C or SPI communication technologies. Other technologies used included GPIO interrupts, periodic capture and compare, PWM, and analog to digital conversions.

2.0 Requirements

1. The Dashboard must be built in a single unit
2. The Dashboard unit must have an LCD display
3. The dashboard LCD must display time
4. The dashboard LCD must display the day of the week
5. The dashboard LCD must display cabin temperature
6. LCD must display the date
7. The dashboard must have an interactive display
8. The dashboard must provide a way to enter a new time
9. The user must provide a way to enter a new date
10. The cabin temperature must be measured with ±1% accuracy
11. A visible warning must be displayed when the cabin temperature is above 100 degrees Fahrenheit
12. The system must provide a way to navigate through the interface on the LCD display
13. The system must provide menus to guide the user through interface on LCD display
14. The speed must be displayed digitally
15. The speed must be displayed in analog (as a speedometer)
16. The speed displays must be synchronized
17. The ambient lighting must be used to control the brightness of the Unit display
18. Unit must be able to sense a door latch
19. The user must be notified visually of an open door latch
20. Objects behind the dashboard within 15 inches must trigger an audible warning to the user
21. Objects behind the dashboard within 15 inches must trigger a warning screen on the LCD display
22. Objects behind the dashboard within 15 inches must trigger a warning LED
23. The system must maintain a log of the last 5 times an object comes within 15 inches of the rear of the dashboard.
24. The dashboard must function without connection to Code Composer Studio
25. Time must be changed as it receives user input
26. Date must be changed as it receives user input
27. A reset feature must be utilized
28. The system must maintain a log of the last five times excursions are recorded over 85 mph
29. The system must maintain a log of the last five dates excursions are recorded over 85 mph
30. The system must be powered by an AC-DC switching power supply
31. The system must be attached to a faceplate

32. Must provide a way to initiate a turn signal (blinkers)
33. System must provide a way to terminate a turn signal
34. The blinkers must provide visual feedback when activated
35. All I2C communication must be on the same port of the two MSP432’s
36. All components must be attached to, or inside, an enclosure
37. The audible warning from an object being closer than 15 inches from the rear of the design must be a tone repeated at least at 3 Hz
38. The speedometer needle must be glow in the dark to be visible when dark
39. The displays on the faceplate must be outlined with glow in the dark material
40. The cost of specially ordered components in the system must cost less than $50
41. The height of the design must be limited to 24 inches tall
42. The depth of the design must be limited to 24 inches deep

3.0 System Architecture

3.1 Hardware Specifications

The High Level Hardware block diagram below in Figure 1 shows the component connection to the MSP.

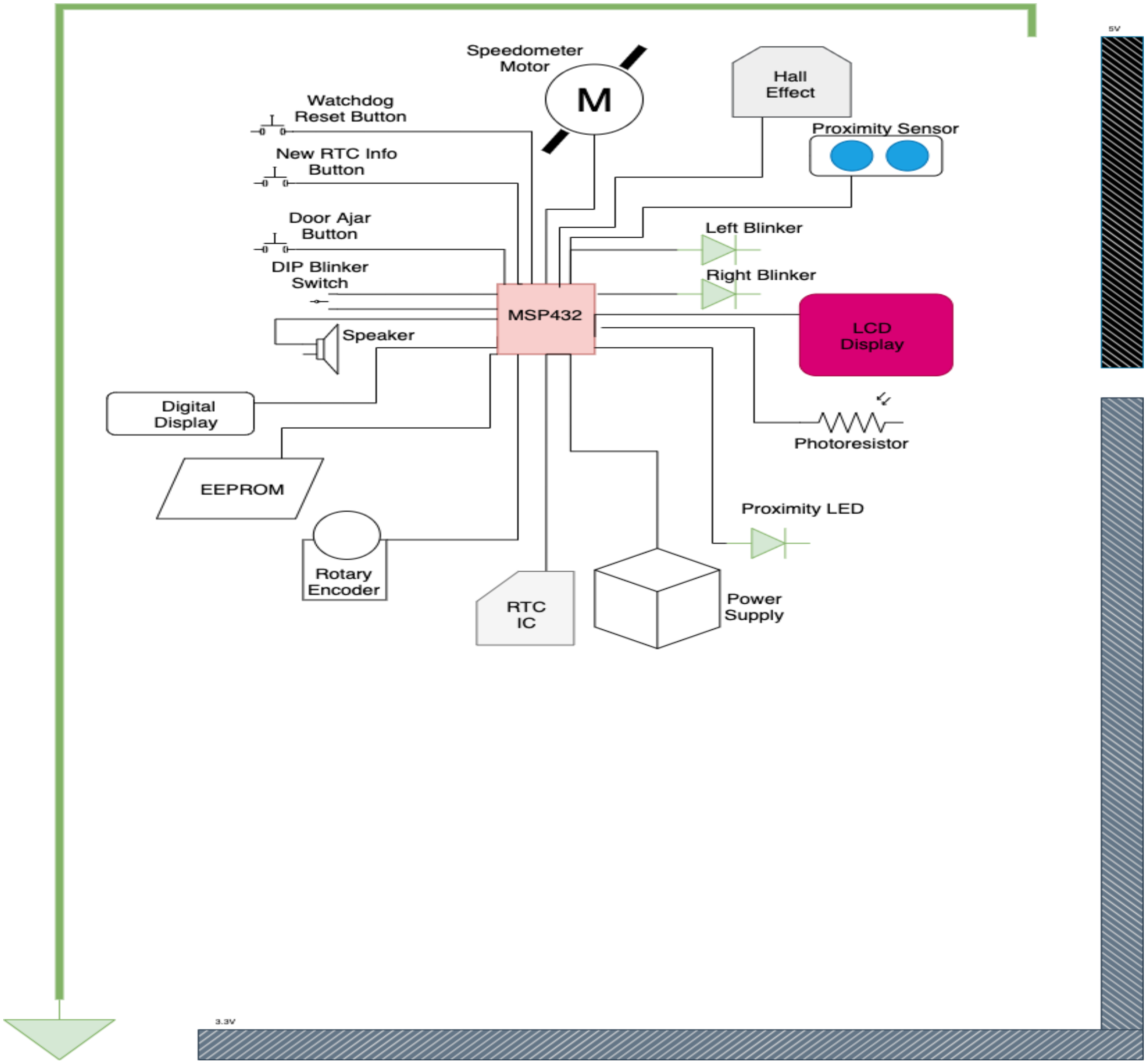


Figure 1: High Level Block Diagram

3.1.1 Hardware Design

Each block contributes to the Dashboard function in the following ways:

The RTC IC holds the data for the date, day, time, and temperature. This data is read and displayed on the LCD. When the user chooses to change the date or time, the data is stored in this component. It is communicated with using I2C communication.

The encoder enables the user to scroll through menu options of the display, and the switch on the encoder enables the user to move through menus, as well as return to the main startup menu.

The LCD displays the day, date, time, and temperature on the startup menu. In other menus it displays the menu options and is updated through user input or displays the alarms in an attractive and readable way. Visible warnings for high cabin temp and an object on close proximity are also displayed on the LCD. It is communicated with using SPI communication.

The proximity sensor will measure the distance of any object behind the rear of the vehicle.

The hall effect sensor will detect rotations of a motor with a magnet on both ends of rotating arms.

The speedometer motor will show the speed in an analog display.

The digital display will show the speed in digital form. It will also warn the user of an open door latch. It is communicated with using SPI communication.

The speaker will output audible sounds to the user caused by alarms.

The proximity LED will light as a warning when an object approaches within 5 inches of the rear of the dashboard unit.

The door ajar button will act as a door latch. When pressed, the dashboard will know that a door is open.

The DIP Blinker switch will act as a toggle switch for the blinkers, enabling the user to initiate and terminate turn signals.

The New RTC Info Button will enable the user to input the currently highlighted digit displayed on the LCD in certain menus to the RTC as the new day, date, or time. After entering the last necessary digit, the button will return the display to the home startup menu.

The Watchdog Reset Button allows the user to reset the dashboard system after 4 seconds.

The left blinker LED acts as left turn signal, blinking when the user chooses to initiate a left turn signal with the toggle switch.

The right blinker LED acts as right turn signal, blinking when the user chooses to initiate a right turn signal with the toggle switch.

The EEPROM holds the date and time of the last 5 alarms in flash memory. The data will be read in the alarm menu or set when speeds are sensed over 85 mph or an object is sensed within 15 inches of the rear of the dashboard. It is communicated with using I2C communication.

The photoresistor will be used to measure the ambient light around the dashboard and change the backlight brightness of the LCD accordingly.

The power supply provides AC-DC power to the MSP432 from a wall outlet, allowing the dashboard to be autonomous and function without connection to Code Composer Studio.

3.1.2 Hardware Implementation

Shown below is a table showing the physical pin connections on each of the components and the MSP

Table 1: MSP432 Pin Connections and Their Function

	MSP Pin / Connection	Device
Device	Pins	Function
RTC	P6.4	I2C SDA
	P6.5	i2C SCL
EEPROM	5V w/ diode	A0
	Gnd	A1
	Gnd	A2
	Gnd	WP
	P6.4	I2C SDA
	P6.5	i2C SCL
Proximity Sensor	P7.7	Pulse Pin -> TimerA PWM
	P7.6	Echo -> TimerA Capture and Compare
Watchdog Reset	P3.3	GPIO Input
	Gnd	Pin2
Photoresistor	P6.1	ADC Channel 14
Stepper Motor	P4.0	IN1 -> Motor1_1
	P4.1	IN2 -> Motor1_2

	P4.2	IN3 -> Motor2_1
	P4.3	IN4 -> Motor2_2
Hall Effect	5V	Vcc
	Gnd	Pin2
	P5.2	Vo -> GPIO Input
ST7735	P9.2	Reset -> GPIO Output
	P9.3	D/C -> GPIO Output
	P9.4	CS -> GPIO Output
	P9.5	SPI SCL
	P9.7	SPI SDA
	P2.4	LED Backlight -> TimerA PWM
Left Blinker	P8.3	LED-> GPIO Output
Right Blinker	P8.4	LED-> GPIO Output
Toggle Switch	P3.6	GPIO Input
	P3.7	GPIO Input
Encoder	P1.5	SW
	P1.6	DT
	P1.7	CLK
Proximity LED	P8.2	LED

Door Latch	P3.5	GPIO Input
	Gnd	Pin2
MAX219	P2.0	CS
	P2.1	CLK
	P2.3	SIMO
Speaker	P5.6	TimerA PWM
	Gnd	Pin2
New RTC Info Btn	P3.2	GPIO Input
	Gnd	Pin2

3.1.2.1 Power Supply Schematic

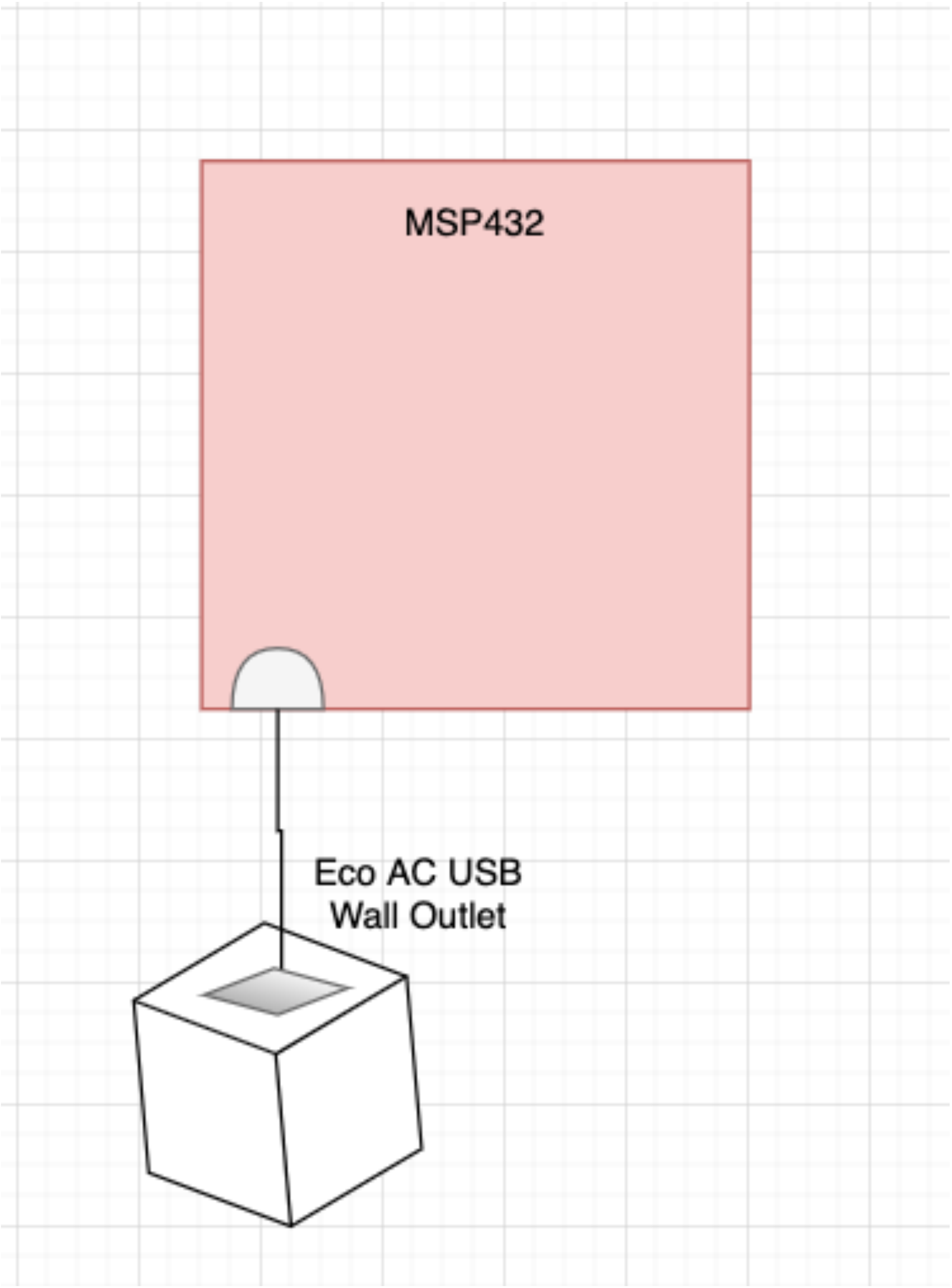


Figure 2: Power Supply Schematic

3.1.2.2 Speedometer Stepper Motor Schematic

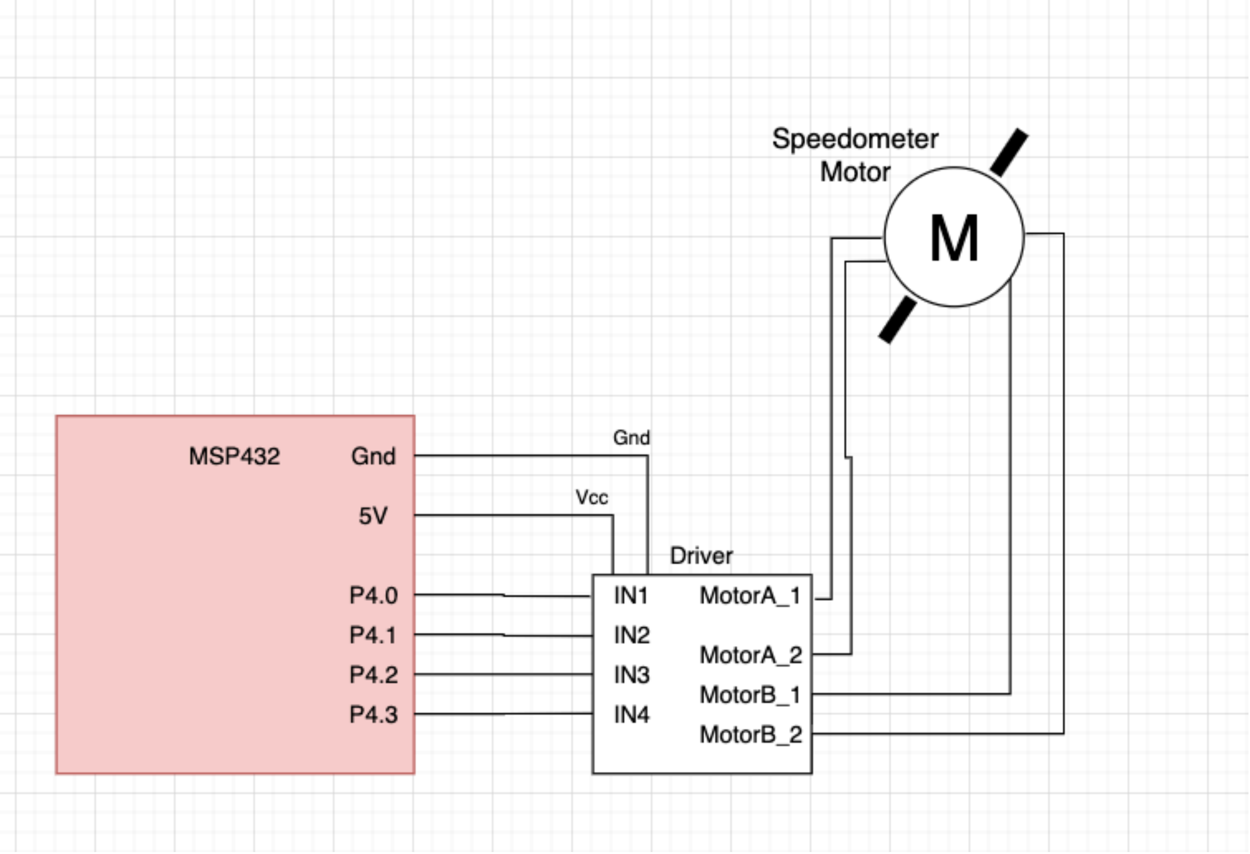


Figure 3: Stepper Motor Schematic

3.1.2.3 RTC Schematic

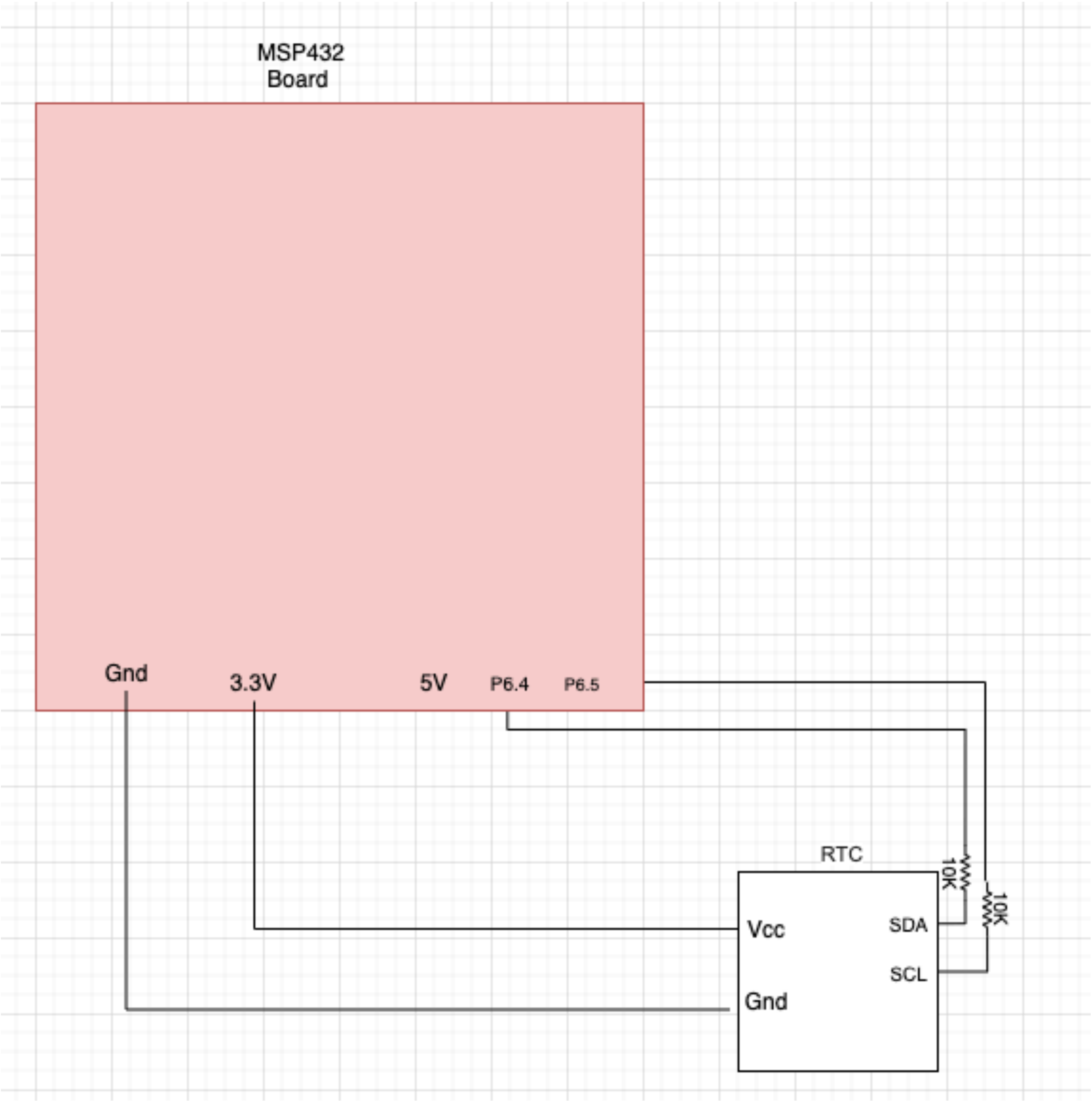


Figure 4: RTC Schematic

3.1.2.4 Left Blinker Schematic

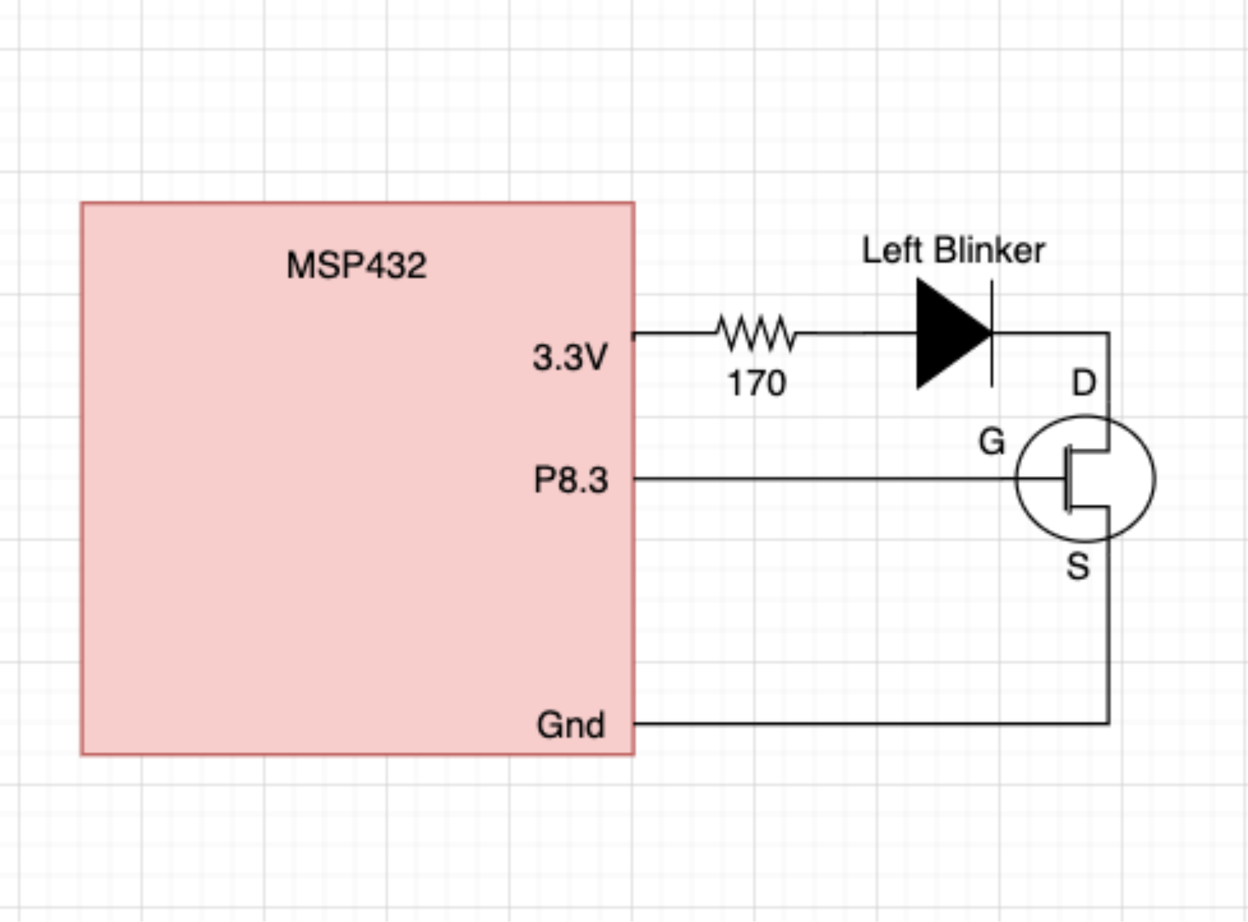


Figure 5: Left Blinker Schematic

3.1.2.5 Right Blinker Schematic

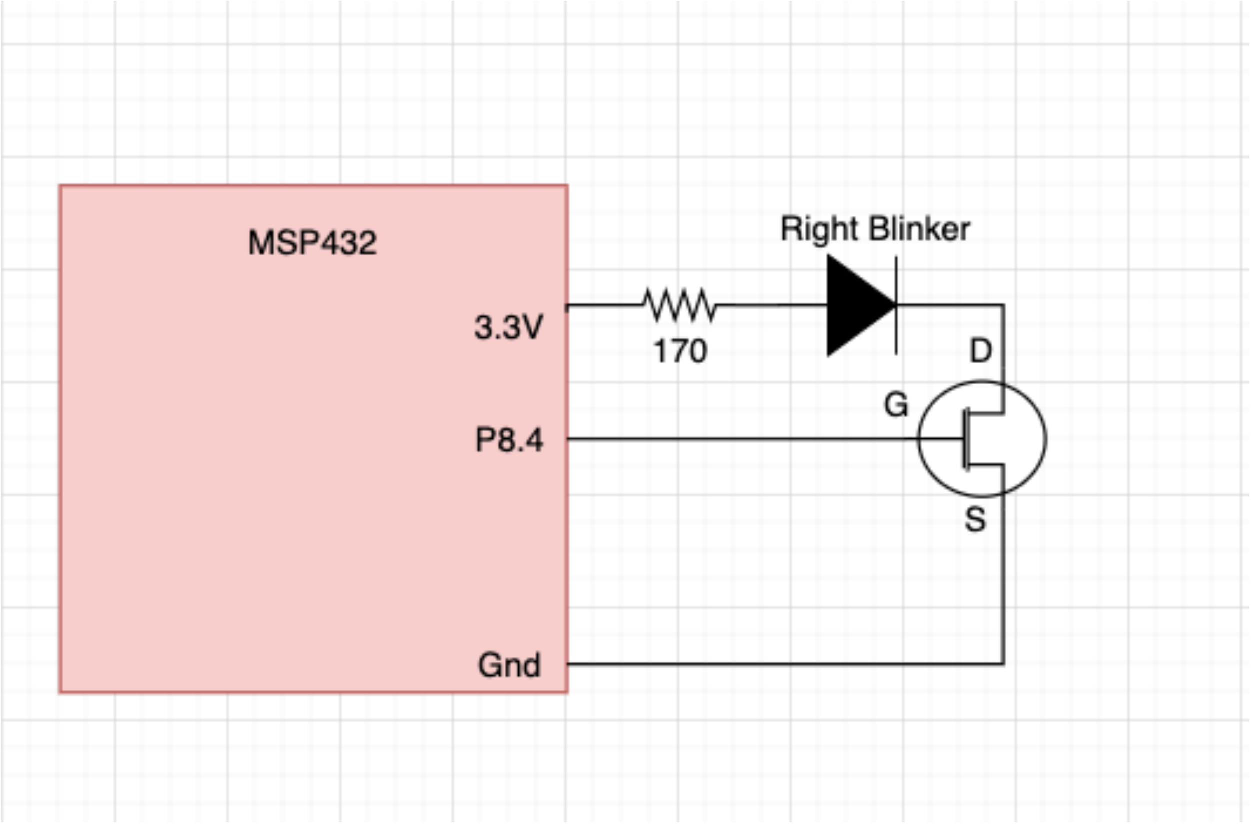


Figure 6: Right Blinker Schematic

3.1.2.6 Proximity LED Schematic

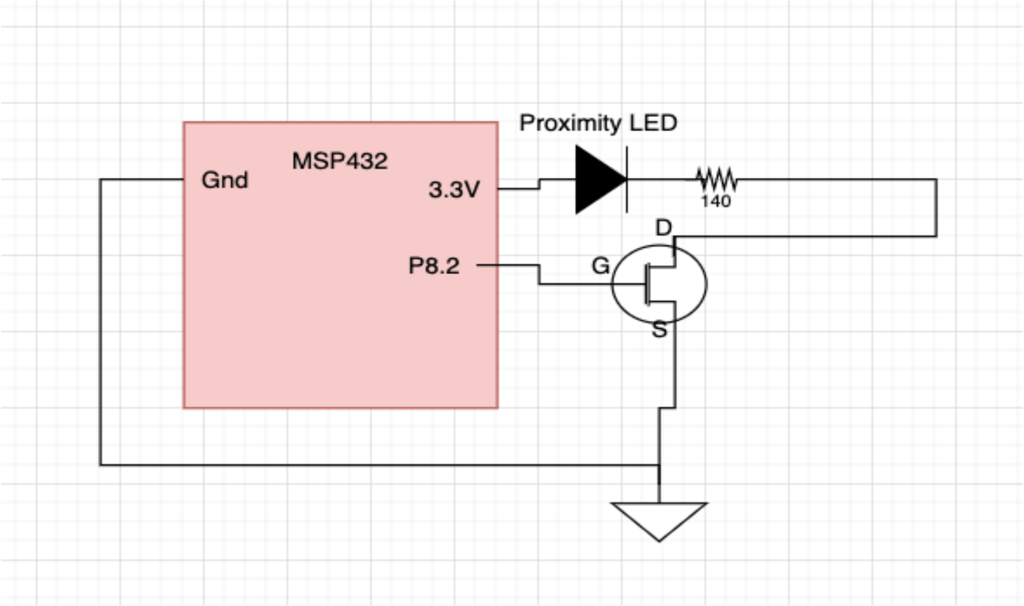


Figure 7: Proximity LED Schematic

3.1.2.7 Blinker Toggle Switch Schematic

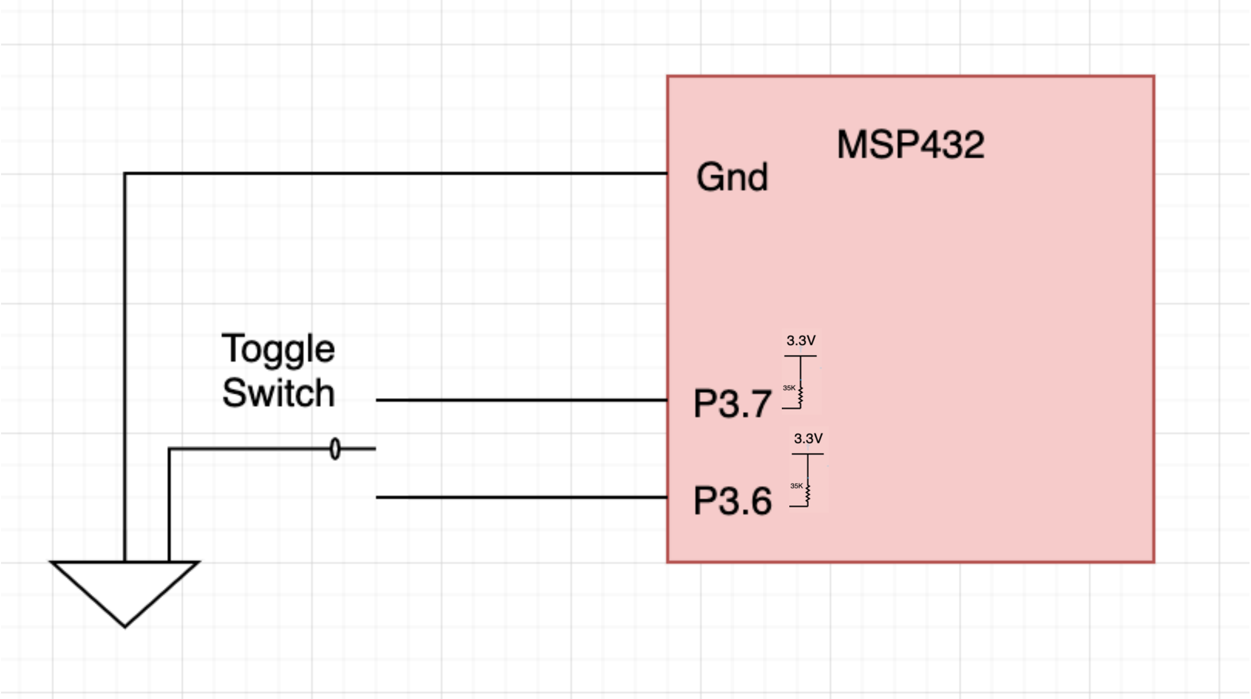


Figure 8: Blinker Toggle Switch Schematic

3.1.2.8 OH-137 Hall Effect Sensor Schematic

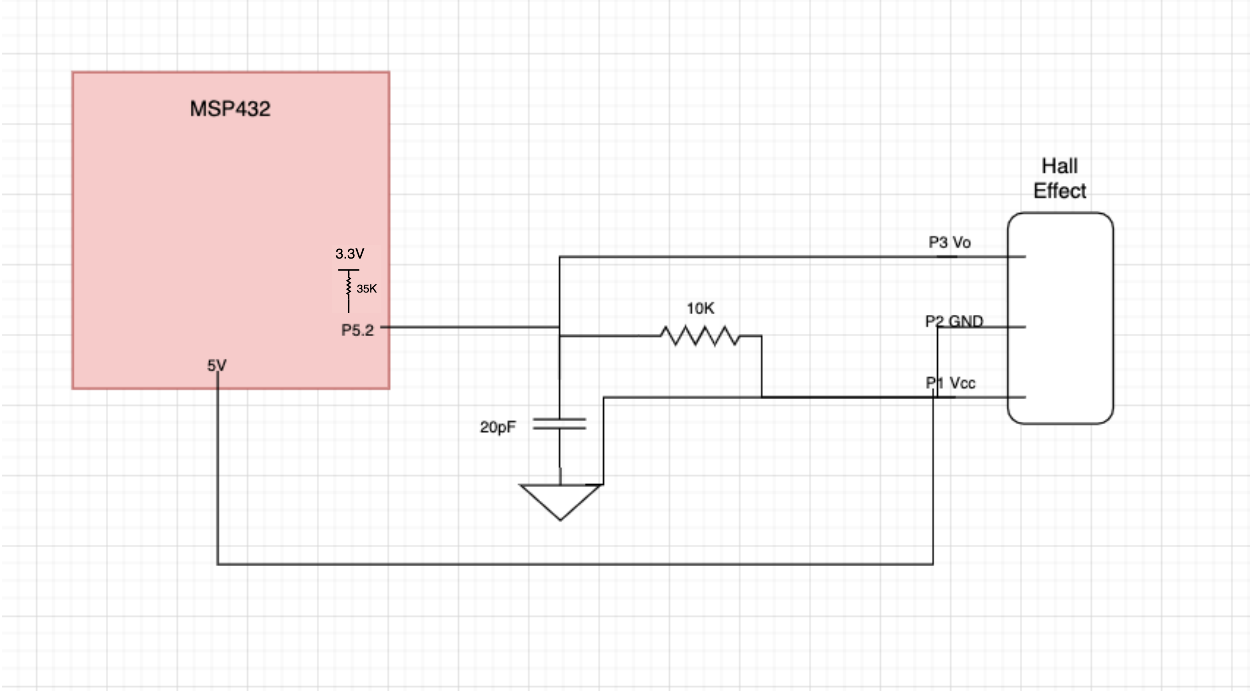


Figure 9: Hall Effect Schematic

3.1.2.9 MAX219 8 Character 7 Segment Digital Display Schematic

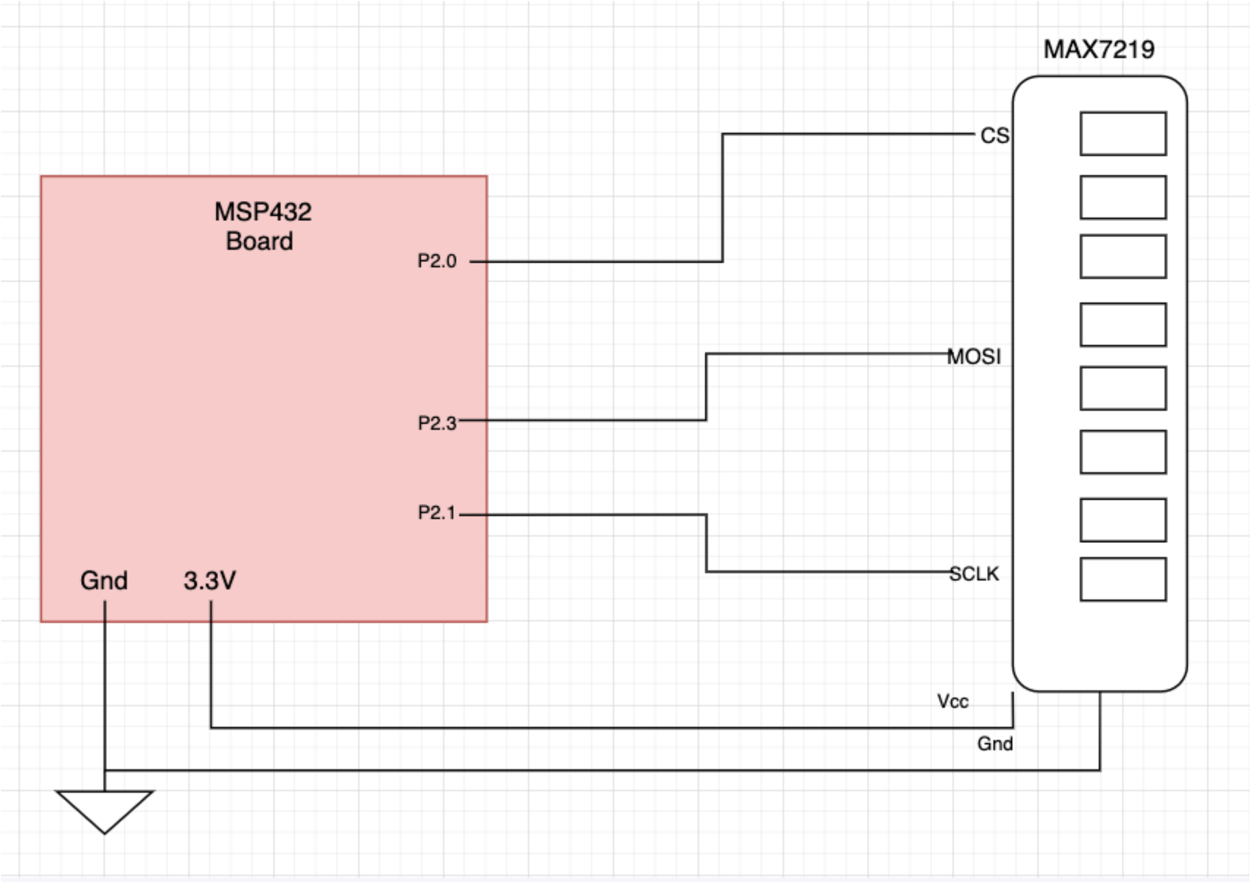


Figure 10: MAX219 8 Character 7 Segment Schematic

3.1.2.10 HC-SR04 Proximity Sensor Schematic

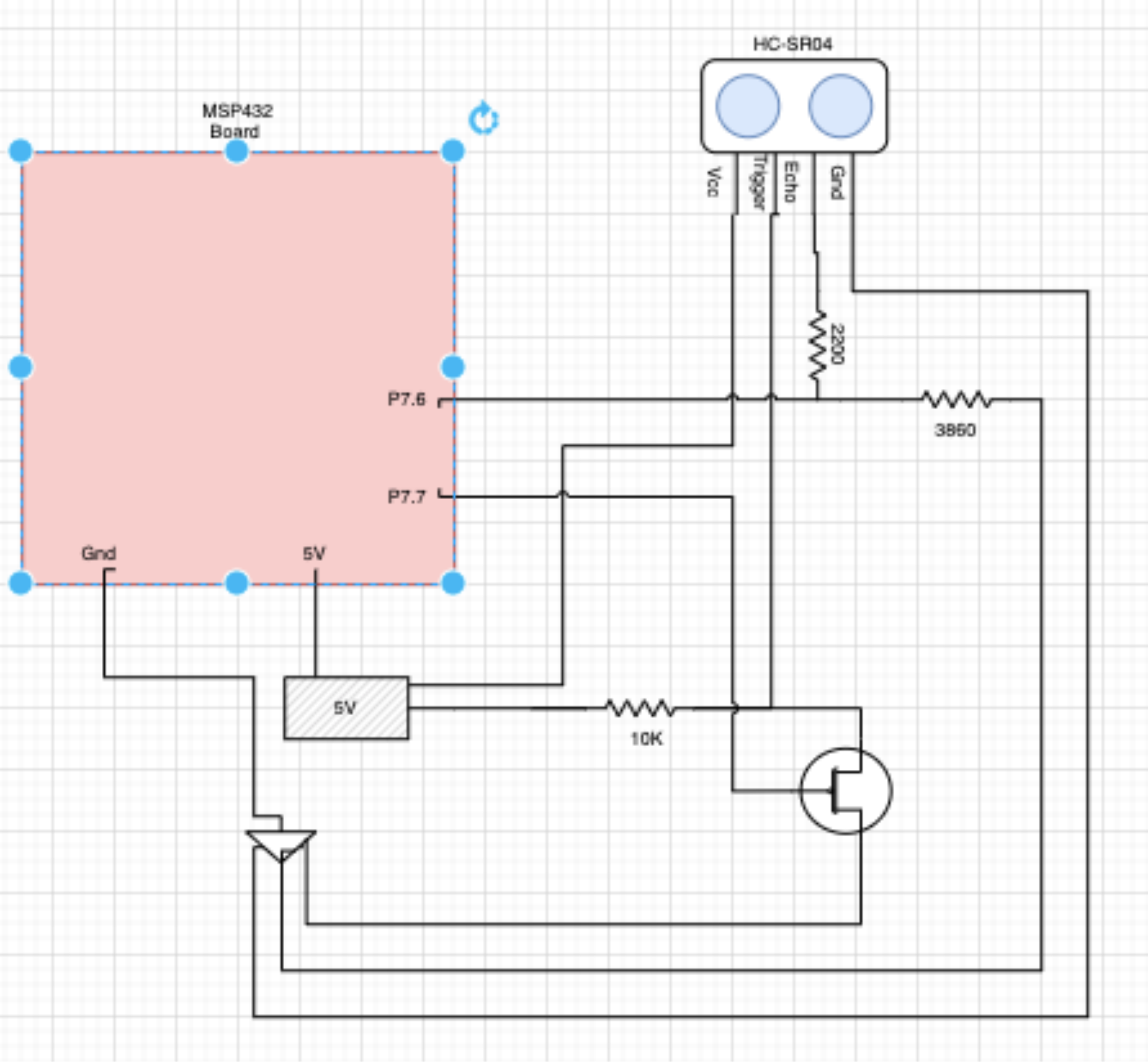


Figure 11: Proximity Sensor Schematic

3.1.2.11 24C02C EEPROM Flash Memory Chip Schematic

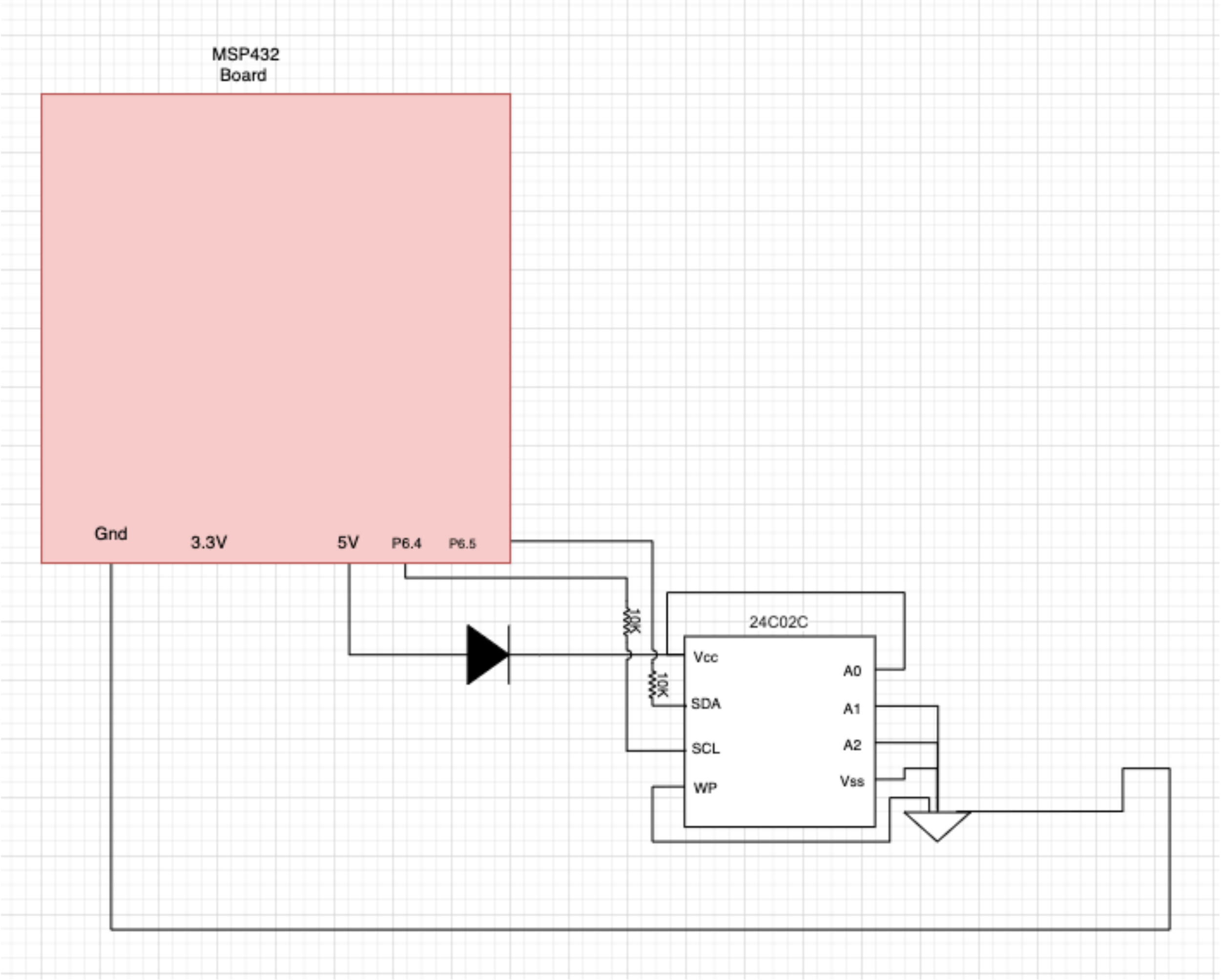


Figure 12: EEPROM Schematic

3.1.2.12 ST7735 LCD Schematic

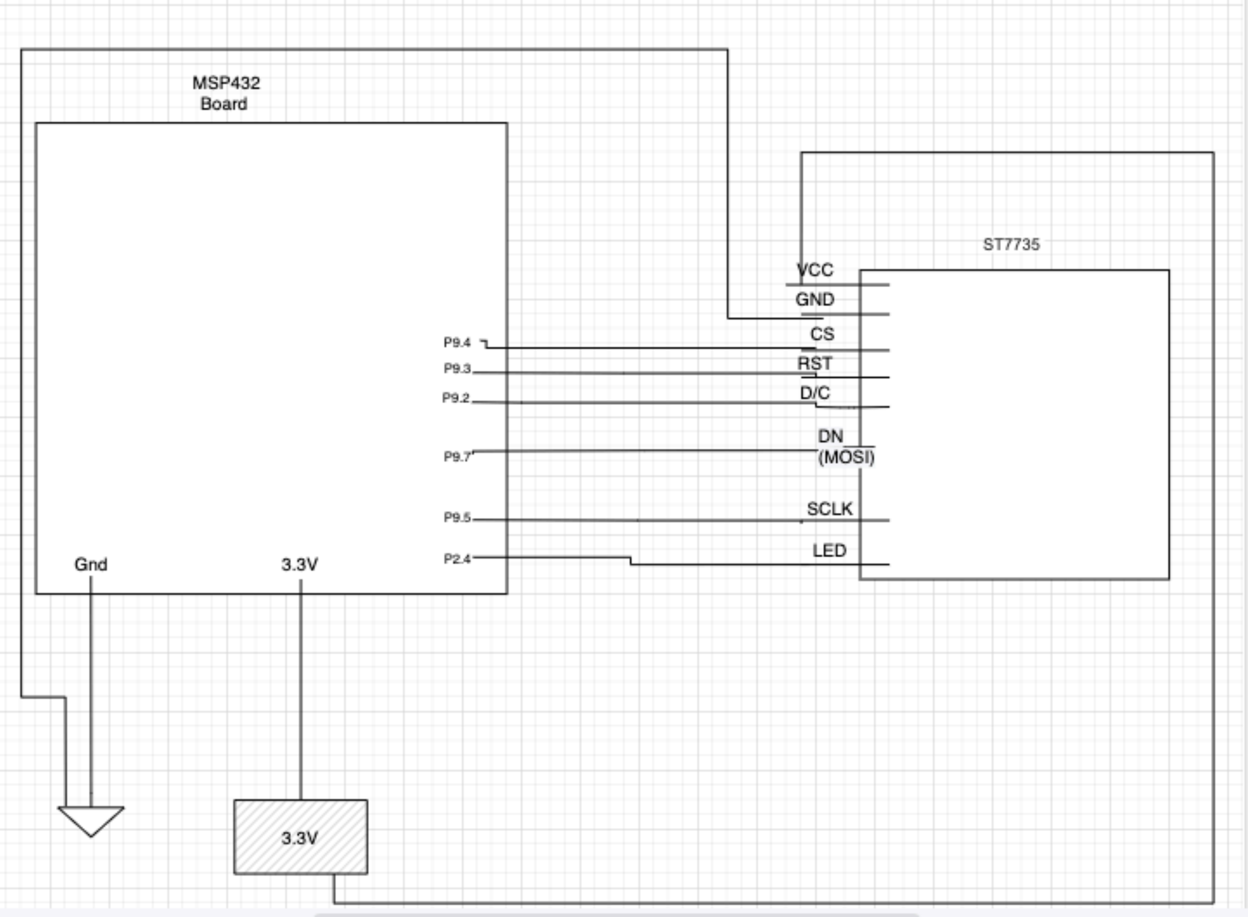


Figure 13: ST7735 LCD Schematic

3.1.2.13 Door Latch Schematic

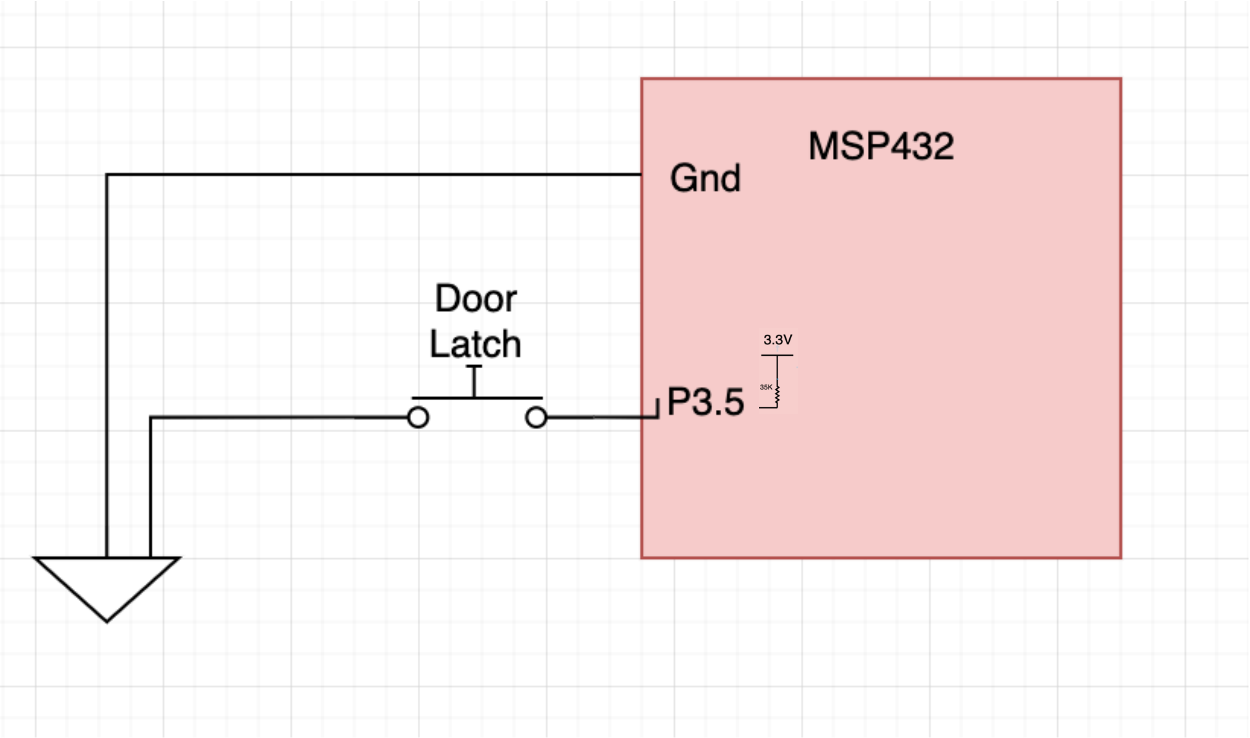


Figure 14: Door Latch Button Schematic

3.1.2.14 Encoder Schematic

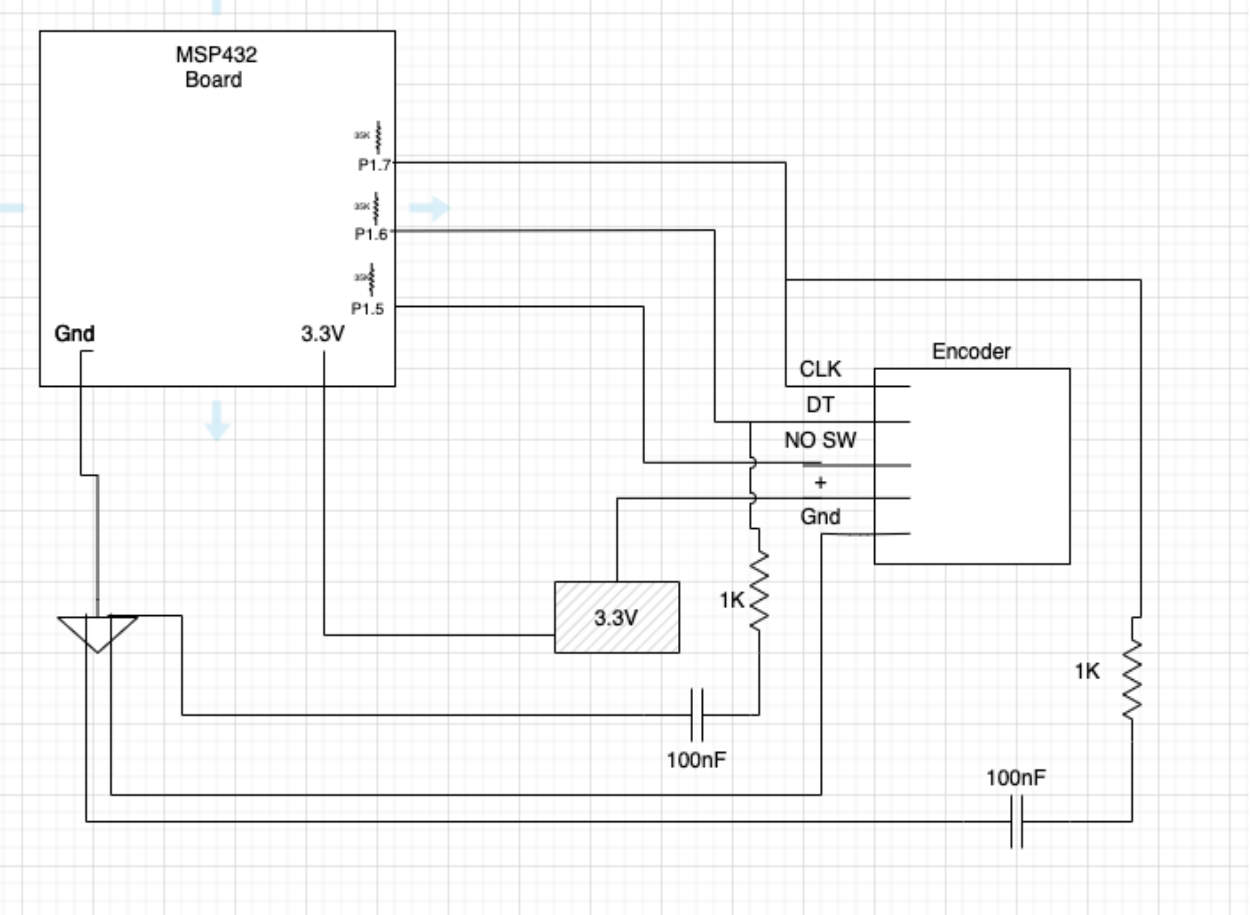


Figure 15: Encoder Schematic

3.1.2.15 Photoresistor Schematic

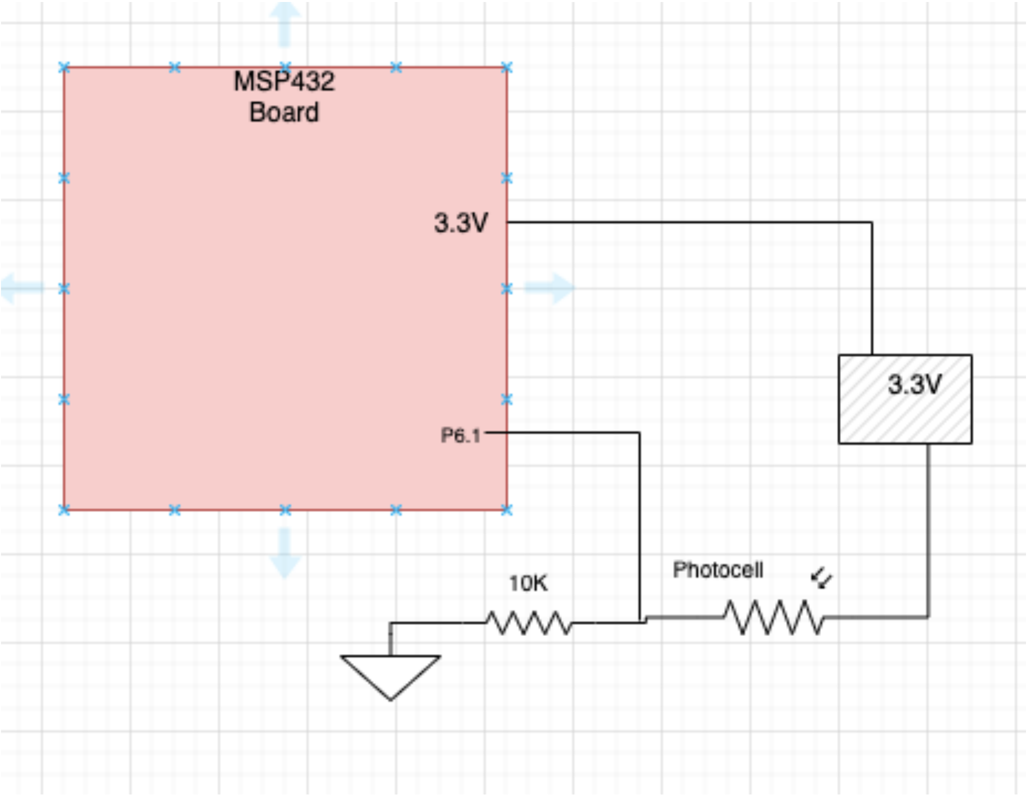


Figure 16: Photoresistor Schematic

3.1.2.16 Watchdog Reset Button Schematic

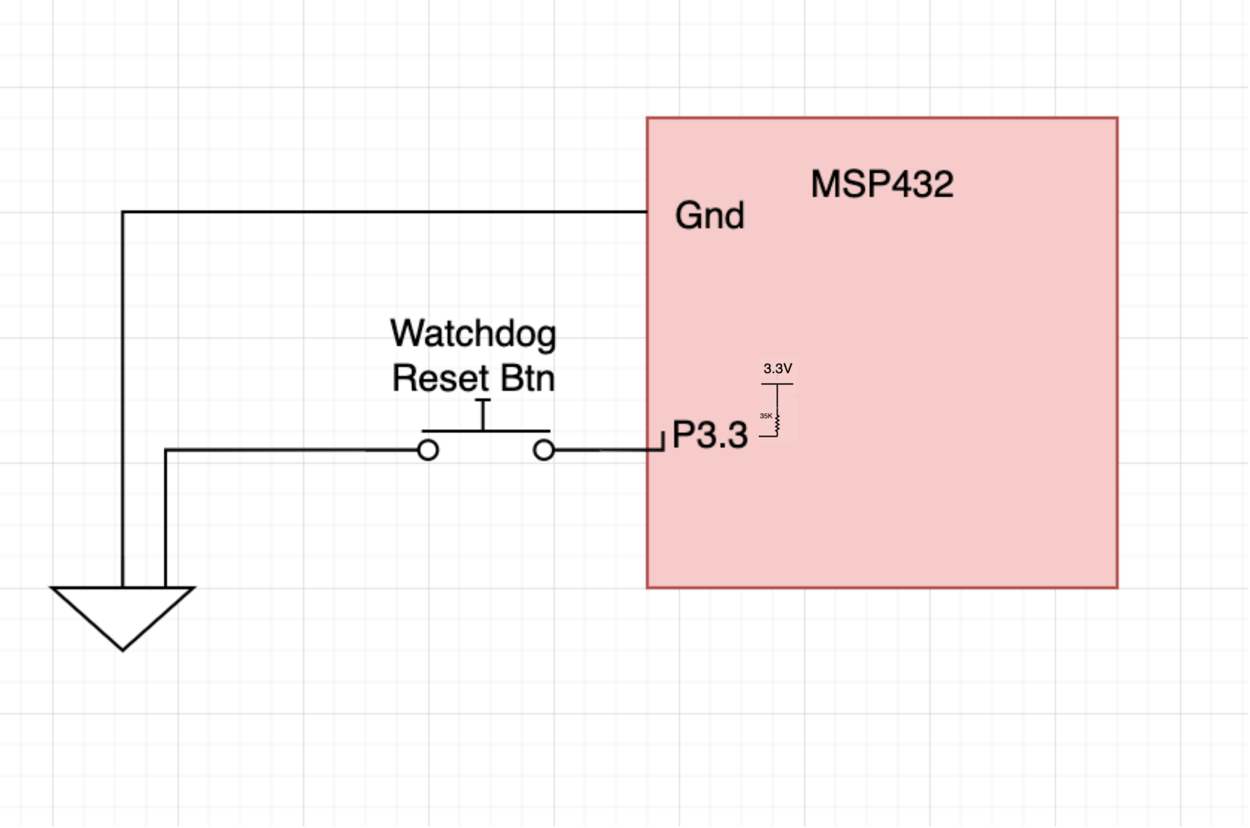


Figure 17: Watchdog Reset Schematic

3.1.2.17 New RTC Info Button Schematic

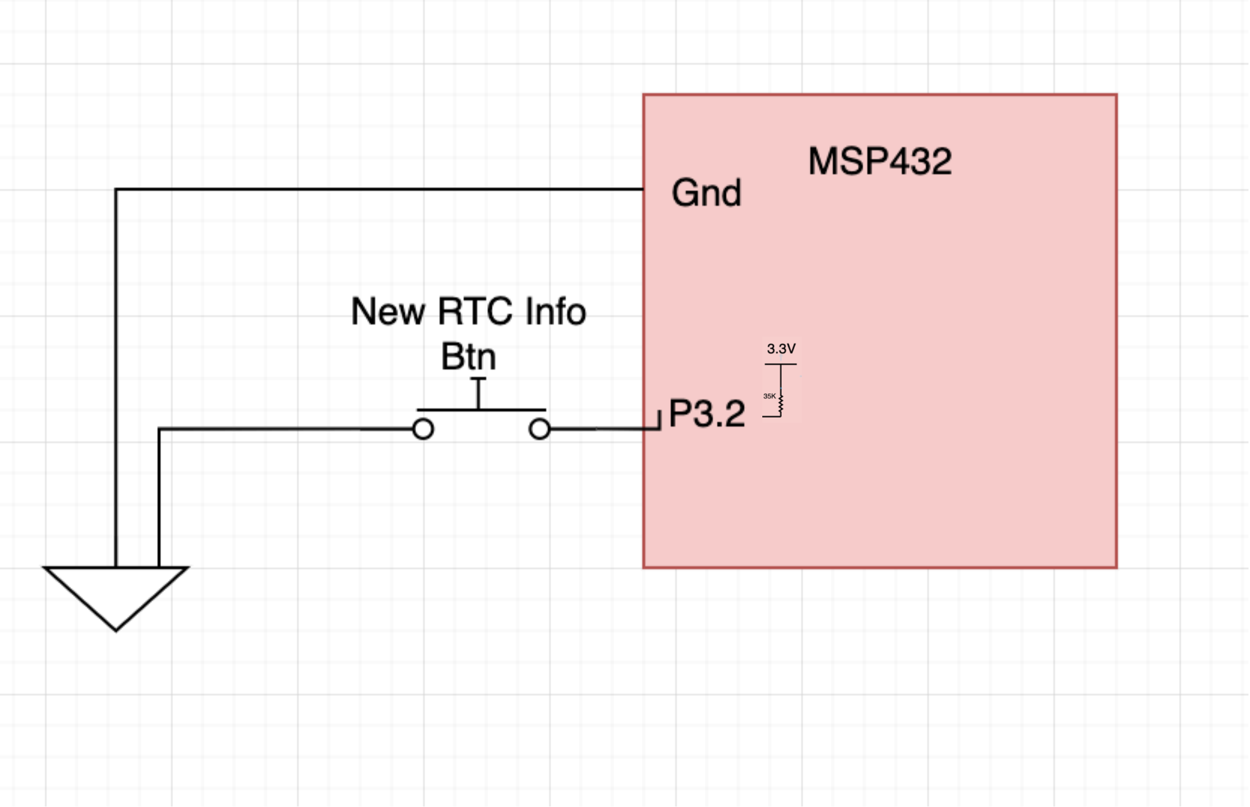


Figure 18: Watchdog Reset Schematic

3.2 Software Specifications

The UML diagram shown below shows the overall process of information flow throughout the program.

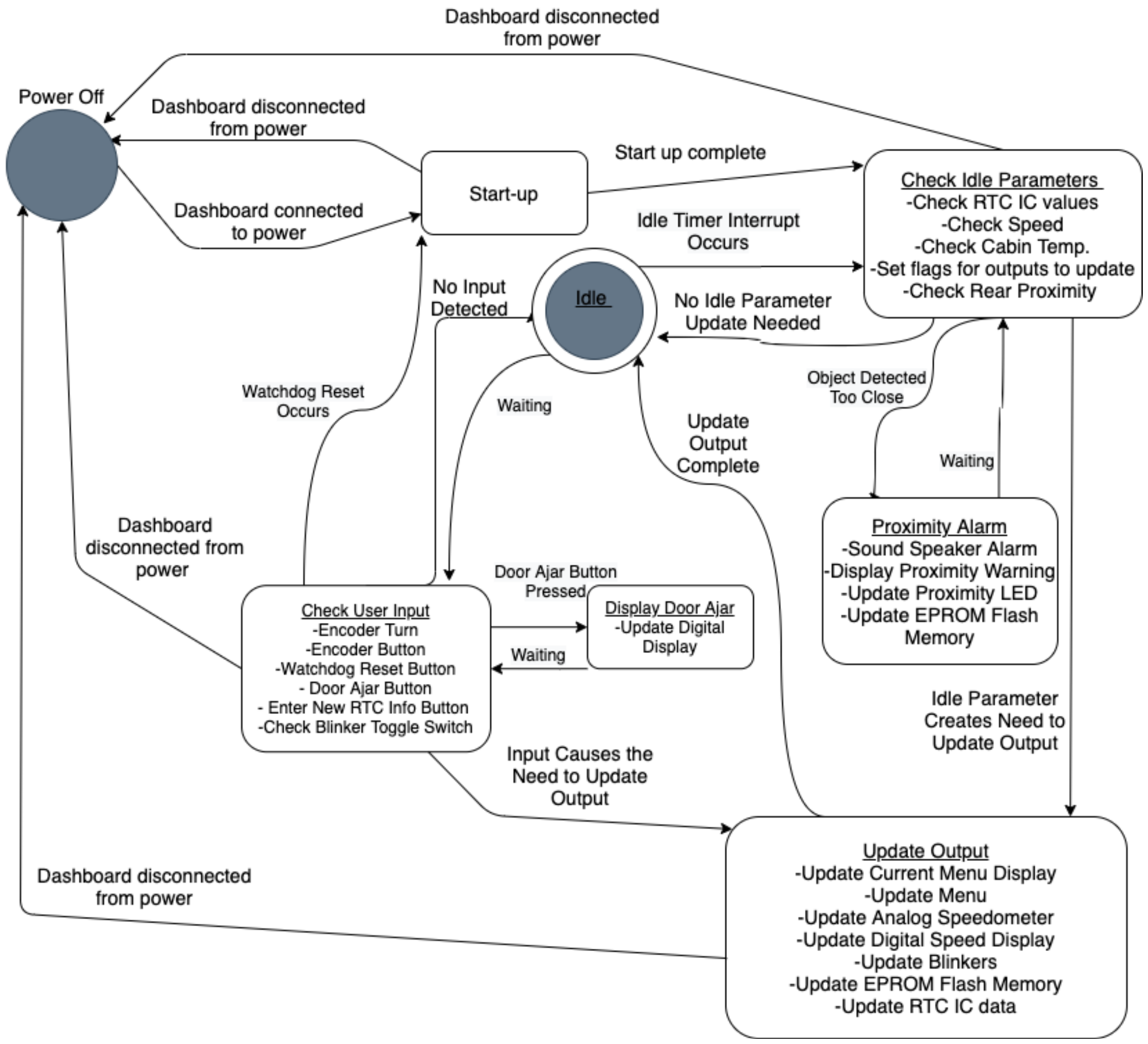

Dashboard disconnected from power
Power Off
Dashboard disconnected from power
Start up complete
Start-up
Dashboard connected to power
Check Idle Parameters
-Check RTC IC values
-Check Speed
-Check Cabin Temp.
-Set flags for outputs to update
-Check Rear Proximity
Idle Timer Interrupt Occurs
No Input Detected
Idle
No Idle Parameter Update Needed
Watchdog Reset Occurs
Waiting
Object Detected Too Close
Update Output Complete
Waiting
Dashboard disconnected from power
Proximity Alarm
-Sound Speaker Alarm
-Display Proximity Warning
-Update Proximity LED
-Update EPROM Flash Memory
Door Ajar Button Pressed
Check User Input
-Encoder Turn
-Encoder Button
-Watchdog Reset Button
- Door Ajar Button
- Enter New RTC Info Button
-Check Blinker Toggle Switch
Display Door Ajar
-Update Digital Display
Waiting
Idle Parameter Creates Need to Update Output
Input Causes the Need to Update Output
Dashboard disconnected from power
Update Output
-Update Current Menu Display
-Update Menu
-Update Analog Speedometer
-Update Digital Speed Display
-Update Blinkers
-Update EPROM Flash Memory
-Update RTC IC data

Figure 19: High Level UML Diagram

3.2.1 Software Design

Small snippets of code are provided to demonstrate some function of each section.

Power Off:

This is the start stage of the system. When the dashboard is connected to power, the system will move to start up. If at any time the system is disconnected from power, it wil return to this state. No modules are executed during this time.

Start Up:

The MSP432 will boot up and prepare the peripherals to run module functions and initialize all components, timer intervals, and global variables for correct function. When complete, the system will move to checking the parameters needed to begin displaying the startup menu.

```
Watchdog_halt();                              //halts watchdog

Project_init();                //initializedcomponents and timers for the project

Enable_Interrupts();                   //enable interrupts for project

Startup_Menu();                //calls a function to display startup menu on reset
```

Check Idle Parameters:

This block is entered whenever a timing interval has triggered an interrupt that causes the system to check the time, date, and temperature values in the RTC, as well as check rear proximity and set flags needed caused by change in system environment in the Idle stage. A change in time, date, or temperature caused by either user input or time passing will cause the module to move to Update outputs. If system status has not changed, the module will move to the idle stage. If an object has been detected too close, the module will move to the Proximity Alarm module. There are two vairables holding the values of the current and last calculated speeds. If these variables are not equal, the system will move to Update Output module, else, the system will move back to the Idle stage.

```
        ADC_Reading();                                          //read ADC value from photoresistor
        change_TIMERA0_1_dutyCycle();//change the LED backlight brightness depending on ADC value


if (Timer32_interruptFlag == 1) {//intrval interrupt for every second
                        RPMs = measure_RPMs(Revolution_count);          //measure RPMs

                        Timer32_interruptFlag = 0;                                      //reset flag
                        Revolution_count = 0;//reset revolution counts for the next second calculation

                        Current_Speed_mph = measure_Speed(RPMs, //calculate speed from RPMs
                                CIRCUMFERENCE_6ftDiameter);

switch (menu) {

                case 0: {

                        if (timerA3_0_interruptFlag == 1) {     //do this if startup menu is running
and a half second has elasped

                                Read_RTC_Print_Time_toLCD();            //read and print the time
                                Read_RTC_Temperature_Print_toLCD();     //read and print the
temperature in Fahrenheit

                                PROXIMITY();            //measure distance from rear of the vehicle
                                timerA3_0_interruptFlag = 0;//reset half second interval flag
                        }

                        break;
                        }
```

Check User Input:

This block will be entered every iteration of the program in the system. The status of dashboard buttons and the rotarty encoder knob will be evealuated. If the status of one of these components creates the need for a change in system output, the system will move to Update output stage. If no buttons are pressed the system will go back to the idle stage. If the door ajar button is pressed, the system wll move to Display Door Ajar module. The effect of a button press and encoder turns are constrained. The encoder turns and button counts are reset in the Update Output stage and will only cause change If the system is within the correct menu. If these conditions are not met, the system will move back to the Idle stage.

```
        leftBlinker_Switch_held = (!(P3IN & BIT6 ));//read tollge switch value for blinkers
        rightBlinker_Switch_held = (!(P3IN & BIT7 ));

        Open_Door_Btn_heldDown = (!(P3IN & 0x20));//sets global value to the state of the button for
door ajar

        for (i = 0; i < 15; i++) {
                Debounce_DoorOpen_Btn();        //global debounce for door open button
                Debounce_EncoderButton(); //global debounce flag for encoder button
                Debounce_NewRTC_Info_Btn();     //global debounce flag for button that enters new RTC
data
                Debounce_WatchodgReset_Btn();//global debounce flag for watchdog reset button button;
}
```

Update Output:

This module will be entered if any user input or idle parameter causes the need to hange the output status of the system. There are two global variables holding the values of the last and current calculated speeds. If the two variables are not equal, the system will calculate the difference. If the result is negative, the speedometer will be moved backward the absolute value of the difference in steps. If the result is positive, the speedometer will move forward the number of steps in difference. The digital speedometer display will alsobe updated with the current speed. If this current speed is above 85 mph, the EEPROM flash memory will be updated with a new alarm, saved in the byte of data corresponding the number of alarms that have been entered. If the number of alarms entered is 5, the variable for alarms entered is reset. If the toggle switch for the blinkers was read low, and the interval timer for toggling the blinkers has gone off, then a set flag will cause the blinkers to toggle. If the encoder button is pressed in the startup menu, the system will update the menu to the user interface options. If the encoder is turned in the user interface menu, the cursor will scroll to the next option. If the encoder button is pressed in the Inferface Options menu, the menu will be changed according to the number of encoder turns. If the encoder button is pressed in any other menu, the system will return to the Startup Menu. If the encoder is turned in the Change Date or Change Time menus, the number displayed will equal the number of encoder turns. If the encoder turns go out of bounds of valid information to store into the RTC, the encoder counts are reset. If the New RTC Info Button is pressed in one of these menus, the data displayed and highlighted by the cursor is saved into an array to be sent to the RTC. Once the button is pressed enough times to have entered all the digits necessary for the data being changed, the data is sent to the RTC and the menu is returned to the Startup Menu. If the menu selected is the Alarms menu, the alarms will be read from the EEPROM and the data is displayed on the LCD.

```
if (Current_Speed_mph < 100) {                            //display speed
                                MAX219_SPI_TransferData(0x05, Current_Speed_mph / 10);//tens place is 
int speed / 10
                                MAX219_SPI_TransferData(0x04, Current_Speed_mph % 10);//ones place is 
remainder of speed / 10
                                MAX219_SPI_TransferData(0x06, 0);//if less than 100, hundreds place 
must be zero
                        }

                        else if (Current_Speed_mph >= 100) {    //display greater than 100
                                MAX219_SPI_TransferData(0x05, (Current_Speed_mph / 10) % 10);
                                MAX219_SPI_TransferData(0x04, Current_Speed_mph % 10);
                                MAX219_SPI_TransferData(0x06, 1);//just print one in hundres place, 
speed will not go above 199
                        }
                        speed_Dif = (Current_Speed_mph - lastSpeed);//before setting speedometer 
position

        //calculate the difference of last and current speed
```

```
                        if (RPMs == 0) {                                //if the RPMs calculated are
zero
                                if (RPMs_areZero == 0) {//if this is not a repeated zero calculation
                                        Rotate_Stepper1_Backward_s_Steps(120);//turn speedometer to
zero position
                                        RPMs = 0;
                        //reset RPMs
                                        RPMs_areZero = 1;//set RPMs are zero flag so this is not
repeated continually while spped is zero
                                }

                                }



void control_Cursor(void) {

        char string_toPrint[20];
        uint16_t start_PositionX;
        uint16_t start_PositionY;

        uint16_t text_Color = 0xFFFF;           //white number when changing
        uint16_t bg_Color = 0;                          //black bg for option being selected

        int i = 0;

        if (cursor_moved == 0) {//only enters when the condition true after turning encoder is not yet
executed

                cursor_moved = 1;                                       //set flag variable

                total_turns = abs(CW_turns - CCW_turns);  //magnitude of pulse direction

                if ((total_turns == 0) && (menu == 1)) {                //highlight Set time

                        Interface_Options_Menu();                               //reprint the option
menu

                start_PositionX = 15;
                start_PositionY = 52;
                        string_toPrint[0] = 'S';//reprint the Set Time option with black background
                string_toPrint[1] = 'e';
                string_toPrint[2] = 't';
                string_toPrint[3] = ' ';
                string_toPrint[4] = 'T';
                string_toPrint[5] = 'i';
                string_toPrint[6] = 'm';
                string_toPrint[7] = 'e';
                string_toPrint[8] = NULL;

                for (i = 0; string_toPrint[i] != NULL; i++) {
                        ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                        ST7735_BLUE, bg_Color, 2);
                        start_PositionX += 12;
                }
        }

                else if ((total_turns == 1) && (menu == 1)) {
```

```
        Interface_Options_Menu();

        start_PositionX = 15;
                start_PositionY = 69;
        string_toPrint[0] = 'S';
        string_toPrint[1] = 'e';
        string_toPrint[2] = 't';
        string_toPrint[3] = ' ';
        string_toPrint[4] = 'D';
        string_toPrint[5] = 'a';
        string_toPrint[6] = 't';
        string_toPrint[7] = 'e';
        string_toPrint[8] = NULL;

        for (i = 0; string_toPrint[i] != NULL; i++) {
                ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                                0x04E0, bg_Color, 2);
                start_PositionX += 12;
        }
        }
```

Display Door Ajar:

In this module, the only function of the software is to display the digital door ajar warning as a safety feature of the vehicle. The status of the input pin will be constantly reevaluated be entering the Check User Input Module and will remain in this loop until the button is no longer pressed.

```
void DoorAjar_Btn_HeldDown(void)
{

        MAX219_SPI_init_NoDecode();     //set MAX219 to nodecode mode to diplay custom letters
        while (Open_Door_Btn_heldDown == 1)        //while the button is depressed
    {
      Open_Door_Btn_heldDown = (!(P3IN & 0x20));     //reevaluate
                Print_DoorAjar_MAX219();                                        //print door Ajar digitally
    }
        blank_MAX219();        //blank the screen (this was for testing mainly, but still works because it is reinitialized in decode mode)
        }
```

Proximity Alarm:

If an object has been detected within 15 inches of the rear of the dashboard, the system will enter this module. The audible speaker warning will be sound, and the visual warning will be displayed on the LCD. The proximity warning LED will be turned on, and an alarm is entered into the EEPROM flash memory only once per object detected. The system will move back to checking the rear proximity, and will continue in this loop until the object is no longer sensed. When it is no longer sensed, the proximity LED and both visual and audible warning will no longer be output to the user.

```
void PROXIMITY() {

        measure_Period();//measure the period of timerA capture for proximity sensor

        if (object_DistanceFromSensor < 15) {//if the distance measured is less than 15 inches

                while (object_DistanceFromSensor < 15) {//repeat this while the object is sensed

                        if (alarm_entered == 0) {//only want to enter one alarm each time, so this
evaluates a flag for that
                                Read_RTC_Store_Alarm_EEPROM(alarm_count);//store alarm in EEPROM
                                alarm_count++;
        //increment count
                                alarm_entered = 1;                      //set the flag after entering
alarm
                                }

                        P8->OUT |= BIT2;//blue LED on 8.2 is lit while an object is near the rear

                        displayParkingWarning();//displays a visual warning on screen each second
                        Bells(5);                                               //audible alarm from
piezo speaker
                        Output_Clear();
        //clear output

                        measure_Period();                       //reevaluate the distance fromthe
object
                                }

                alarm_entered = 0;//when the object is no longer sensed within 15, reset the alarm flag
                Startup_Menu();                                         //go back to the
startup menu

                } else
                P8->OUT &= ~BIT2;                                                               //else,
turn the LED off
}
```

3.2.2 Software Implementation

An oscilloscope was used to debug signals going to and from components and the development board. A DMM was used to ensure good electrical connections as well as measure the level of input, output, and resistance from components. The software environment used to develop this product was Code Composer Studio. Libraries were created for each component or type of component to organize functions to make debugging easier and more organized. The debugger of CCS was used to program and test the board after modifications were made. Separate “main” functions were created and saved to test each component individually and eventually integrate their functions. Some main testing function snippets are shown below, along with the libraries and functions they implement.

In Project.c

```
void main09965()     //main for testing door Ajar
{
  int i = 0;

  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

  Clock_Init48MHz();

  SysTickInit_NoInterrupts();

  openDoor_Btn_init();
  SPI_init();

          MAX219_SPI_init_Decode();

  Enable_Interrupts();

  while (1)
  {
    Open_Door_Btn_heldDown = (!(P3IN & 0x20));

    for (i = 0; i < 15; i++)
    {
      Debounce_DoorOpen_Btn();
    }

                          if (Open_Door_Btn_interruptFlag == 1)          //interrupt occurs
                                      {
                                  if (Open_Door_Btn_pressed == 1)     //confirms debounce successful
                                              {
                                          DoorAjar_Btn_HeldDown();
                                          Open_Door_Btn_pressed = 0;          //reset debounce flag
                                          Open_Door_Btn_interruptFlag = 0;
                                  }
                                  MAX219_SPI_init_Decode();
                          }

                          else
```

```
                        MAX219_SPI_TransferData(0x04, 0x07);        //display 74
                        MAX219_SPI_TransferData(0x04, 0x04);

  }
            }

void DoorAjar_Btn_HeldDown(void)
{

            MAX219_SPI_init_NoDecode();     //set MAX219 to nodecode mode to diplay custom letters
            while (Open_Door_Btn_heldDown == 1)        //while the button is depressed
       {
          Open_Door_Btn_heldDown = (!(P3IN & 0x20));     //reevaluate
                        Print_DoorAjar_MAX219();                                                //print door Ajar digitally
       }
            blank_MAX219();        //blank the screen (this was for testing mainly, but still works because it is reinitialized in decode mode)
            }
```

4.0 Verification

In order to meet the requirements and validate them as described below, timing intervals and communications were designed to work with the speed of the microcontroller clock. It was risky to implement PWM using the same timers as some timing and toggle intervals, as the initializations may affect or even overwrite others. To ensure correct function, the first (or 0th) instance of TimerA peripheral was used to create timing intervals and PWM was used on different instances. Another challenge was ensuring that incorrect data did was not entered into the components such as the RTC and EEPROM. Each condition for entering each digit of data for these components was carefully programmed and tested. Another risk was poor and messy electrical connections. Research was done on extra components such as the toggle switch to ensure their ease of use and manufacturing quality. Many electrical connections were soldered and heat shrunk to minimize stress on them, and a set of wires was purchased specifically for breadboard organization and function. Connection issues would also occur if components were connected to incorrect voltages. Voltage rails were labeled with marker on the sides of the breadboards. Putting the I2C lines on the same port can cause issues when trying to communicate with one device but not another. Pull up resistors were used to negate the chance of incorrect device communication. All requirements were verified successfully.

1. Inspection. It can be observed that the dashboard is built in a single unit.
2. Inspection. It can be observed that there is an LCD display on the dashboard unit.
3. Testing. Upon startup, the current time will be displayed on the LCD.
4. Testing. Upon startup, the current day of the week will be displayed on the LCD.
5. Testing. Upon startup, the current cabin temperature will be displayed on the LCD.
6. Testing. Upon startup, the current date will be displayed on the LCD.
7. Testing. A button will be pressed to change the menu screens on the LCD.
8. Testing. A button will be pressed to initiate the user interface, allowing the user to input the time, and the button will be pressed to enter the time.
9. Testing. A button will be pressed to initiate the user interface, allowing the user to input the date, and the button will be pressed to enter the date.
10. Testing. A function will be used to read the temperature from the RTC, that output will be compared to an external thermometer.
11. Testing. A heat gun will be applied to the RTC with a thermometer next to it and the warning will be visibly output when that temperature is 100 degrees Fahrenheit and above.
12. Testing. The interactive menu will be displayed, and the encoder will be used to scroll through options.
13. Analysis. A button will be pressed, and the interface menus will be displayed.
14. Testing. Motor rotations will be detected which will be converted to speed and displayed on characters of a 7-Segment.
15. Testing. Motor rotations will be detected which will be converted to speed and displayed in analog fashion on a speedometer
16. Testing. The speed of an external motor will be changed. The stepper motor displaying speed and the speed of the digital display will be the same.
17. Testing. The ambient light will be changed, and the backlight brightness of the display will be changed.
18. Testing. A pressed button simulates an open door; when it is no longer pressed, the door ajar warning will go away.
19. Testing. A button will be pressed to digitally output an open door warning.
20. Testing. An object will be brought within 15 inches of the rear, an audible sound will be output.
21. Testing. An object will be brought within 15 inches of the rear, a visual warning will be displayed on the LCD.
22. Testing. An object will be brought within 15 inches of the rear, an LED will light.
23. Testing. An object will be brought within 15 inches of the rear, the EEPROM will be read to analyze the times stored.

24. Testing. The design will be disconnected from Code Composer and powered up and will then run the program autonomously
25. Testing. The current time will be changed and updated on display
26. Testing. The current date will be changed and updated on the display.
27. Testing. A button will be pressed, and the MSP will run a soft reset after 4 seconds.
28. Testing. More than 5 alarms will be triggered from speeds over 85 mph with their times being recorded, the EEPROM will be read to analyze the times stored.
29. Testing. The speed of the motor will be changed to greater than 85 mph 5 times with their dates recorded, the EPROM will be read to analyze the dates stored.
30. Testing. An AC-DC wall plug must provide power to the dashboard. .
31. Testing. The unit will be picked up or moved, the components attached to the faceplace will be immoble.
32. Testing. Blinkers will be off. A button will be pressed, Blinkers will activate.
33. Testing. Blinkers will be on. A button will be pressed, Blinkers will deactivate.
34. Testing. Blinkers will be off. The user will toggle a switch, blinker will activate corresponding to user input, blinking in a specified sequence
35. Inspection. The I2C connections will be visible using a specific wire color
36. Analysis. The design will be closely monitored while operating, the enclosure should enclose moving components all of the time.
37. Testing. A one second timer will be used. A counter will increment every time the chime tone is outputted during the second. The counter will be greater than or equal to 3.
38. Analysis. A list of components and costs will be recorded and analyzed throughout the project.
39. Inspection. Ambient lighting will be dark, the speedometer needle will be visible in the dark.
40. Inspection. Ambient lighting will be dark, the displays on the faceplate will be visible in the dark.
41. Testing. A measuring tape will be used to measure the dimension

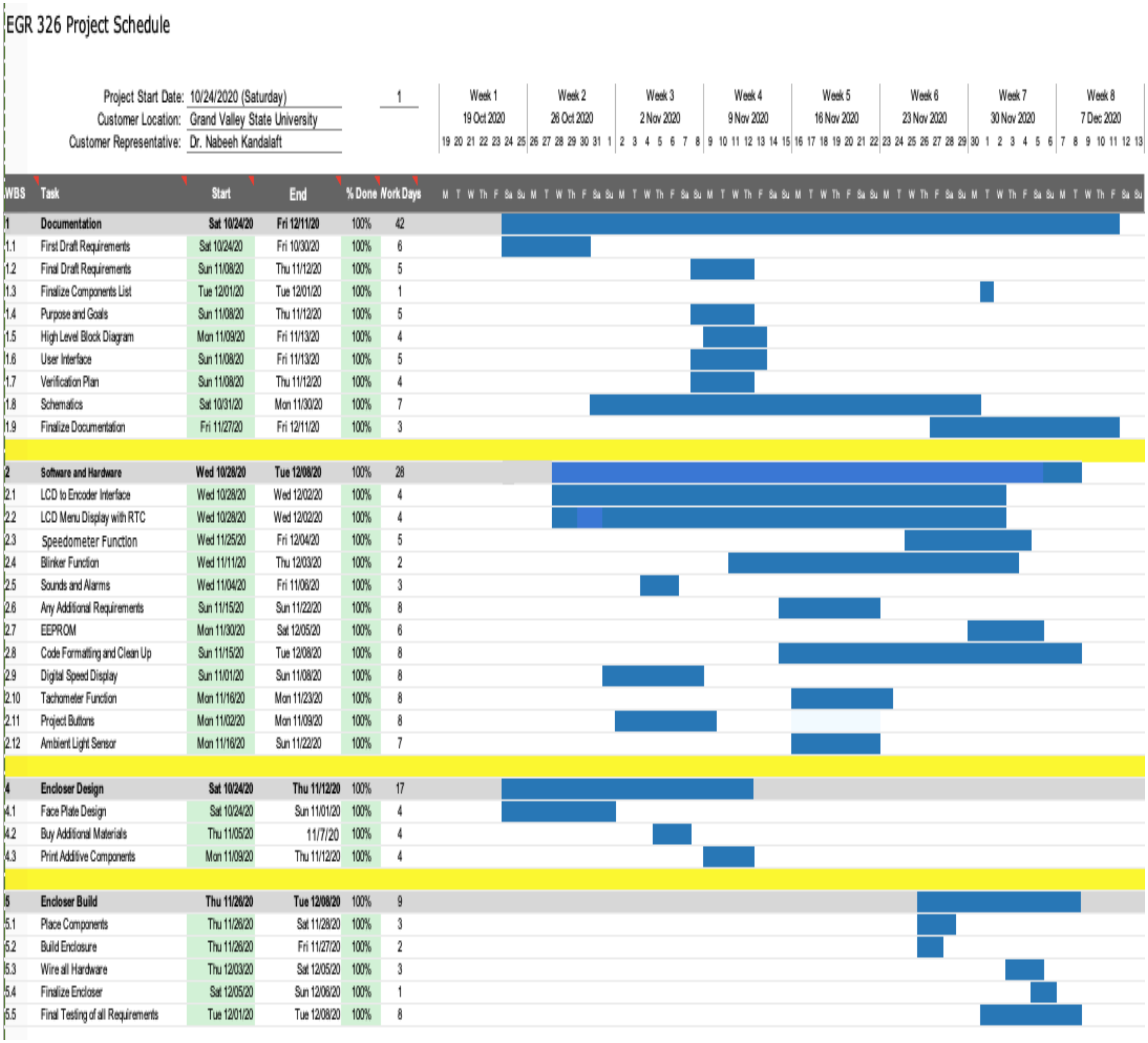
42. Testing. A measuring tape will be used to measure the dimension

5.0 Project Budget and Schedule

5.1 Budget

Component	Supplier	Part #	Price Per Unit	Units Used	Total
Toggle Switch	APIELE	APIELE-TOGGLE	$3.66	1	$3.66
Glow-In-The-Dark PLA 3D Printing Filament	Amolen	3D-GID-GN1	$0.026 / g	10	$0.27
Tactile Pushbutton	TE Connectivity	1825967-2	$0.24	1	$0..24
Dupont Wires	Elegoo	--	$0.058	48	$2.79
Straight Wires	AUSTOR	AMA-18-580	$0.019	22	$0.43
Heat Shrink	Eventronic	ET1002	$0.012	24	$0.30
Pine Wood	--	--	--	--	--
Screws	Menards	--	$0.19	24	$4.56
Washers	Menards	--	$0.08	6	$0.48
Nuts	Menards	--	$0.13	6	$0.78
Bolts	Menards	--	$0.13	6	$0.78
Spray Paint	Rustoleum	2006618774	$7.30	2	$14.60

5.2 Schedule



EGR 326 Project Schedule

Project Start Date: 10/24/2020 (Saturday)
Customer Location: Grand Valley State University
Customer Representative: Dr. Nabeeh Kandalaft

WBS	Task	Start	End	% Done	Work Days
1	**Documentation**	**Sat 10/24/20**	**Fri 12/11/20**	100%	42
1.1	First Draft Requirements	Sat 10/24/20	Fri 10/30/20	100%	6
1.2	Final Draft Requirements	Sun 11/08/20	Thu 11/12/20	100%	5
1.3	Finalize Components List	Tue 12/01/20	Tue 12/01/20	100%	1
1.4	Purpose and Goals	Sun 11/08/20	Thu 11/12/20	100%	5
1.5	High Level Block Diagram	Mon 11/09/20	Fri 11/13/20	100%	4
1.6	User Interface	Sun 11/08/20	Fri 11/13/20	100%	5
1.7	Verification Plan	Sun 11/08/20	Thu 11/12/20	100%	4
1.8	Schematics	Sat 10/31/20	Mon 11/30/20	100%	7
1.9	Finalize Documentation	Fri 11/27/20	Fri 12/11/20	100%	3
2	**Software and Hardware**	**Wed 10/28/20**	**Tue 12/08/20**	100%	28
2.1	LCD to Encoder Interface	Wed 10/28/20	Wed 12/02/20	100%	4
2.2	LCD Menu Display with RTC	Wed 10/28/20	Wed 12/02/20	100%	4
2.3	Speedometer Function	Wed 11/25/20	Fri 12/04/20	100%	5
2.4	Blinker Function	Wed 11/11/20	Thu 12/03/20	100%	2
2.5	Sounds and Alarms	Wed 11/04/20	Fri 11/06/20	100%	3
2.6	Any Additional Requirements	Sun 11/15/20	Sun 11/22/20	100%	8
2.7	EEPROM	Mon 11/30/20	Sat 12/05/20	100%	6
2.8	Code Formatting and Clean Up	Sun 11/15/20	Tue 12/08/20	100%	8
2.9	Digital Speed Display	Sun 11/01/20	Sun 11/08/20	100%	8
2.10	Tachometer Function	Mon 11/16/20	Mon 11/23/20	100%	8
2.11	Project Buttons	Mon 11/02/20	Mon 11/09/20	100%	8
2.12	Ambient Light Sensor	Mon 11/16/20	Sun 11/22/20	100%	7
4	**Encloser Design**	**Sat 10/24/20**	**Thu 11/12/20**	100%	17
4.1	Face Plate Design	Sat 10/24/20	Sun 11/01/20	100%	4
4.2	Buy Additional Materials	Thu 11/05/20	11/7/20	100%	4
4.3	Print Additive Components	Mon 11/09/20	Thu 11/12/20	100%	4
5	**Encloser Build**	**Thu 11/26/20**	**Tue 12/08/20**	100%	9
5.1	Place Components	Thu 11/26/20	Sat 11/28/20	100%	3
5.2	Build Enclosure	Thu 11/26/20	Fri 11/27/20	100%	2
5.3	Wire all Hardware	Thu 12/03/20	Sat 12/05/20	100%	3
5.4	Finalize Encloser	Sat 12/05/20	Sun 12/06/20	100%	1
5.5	Final Testing of all Requirements	Tue 12/01/20	Tue 12/08/20	100%	8

6.0 Revision History

Table

Revision Number	Date	Author	Changes
Requirements Doc Rev A	10/21/2020	Jordan Hayes Nick Biesbock	First draft of Design Specifications Document
Requirements Doc Rev B	11/13/2020	Jordan Hayes Nick Biesbock	Design Specification cut from Purpose and Goals Requirements no longer included user actions, but system "provided function" Validation labels corrected
Design Doc Rev A	12/14/2020	Jordan Hayes	Project goal changed to single unit design Customer Requirements and validations reassigned to conform with University guidelines UML diagram follows convention, with a clear beginning and idle stage Final Requirements link requirement to customer requirement number High Level Block Diagram does not include part numbers

Appendix A – Electrical Schematics

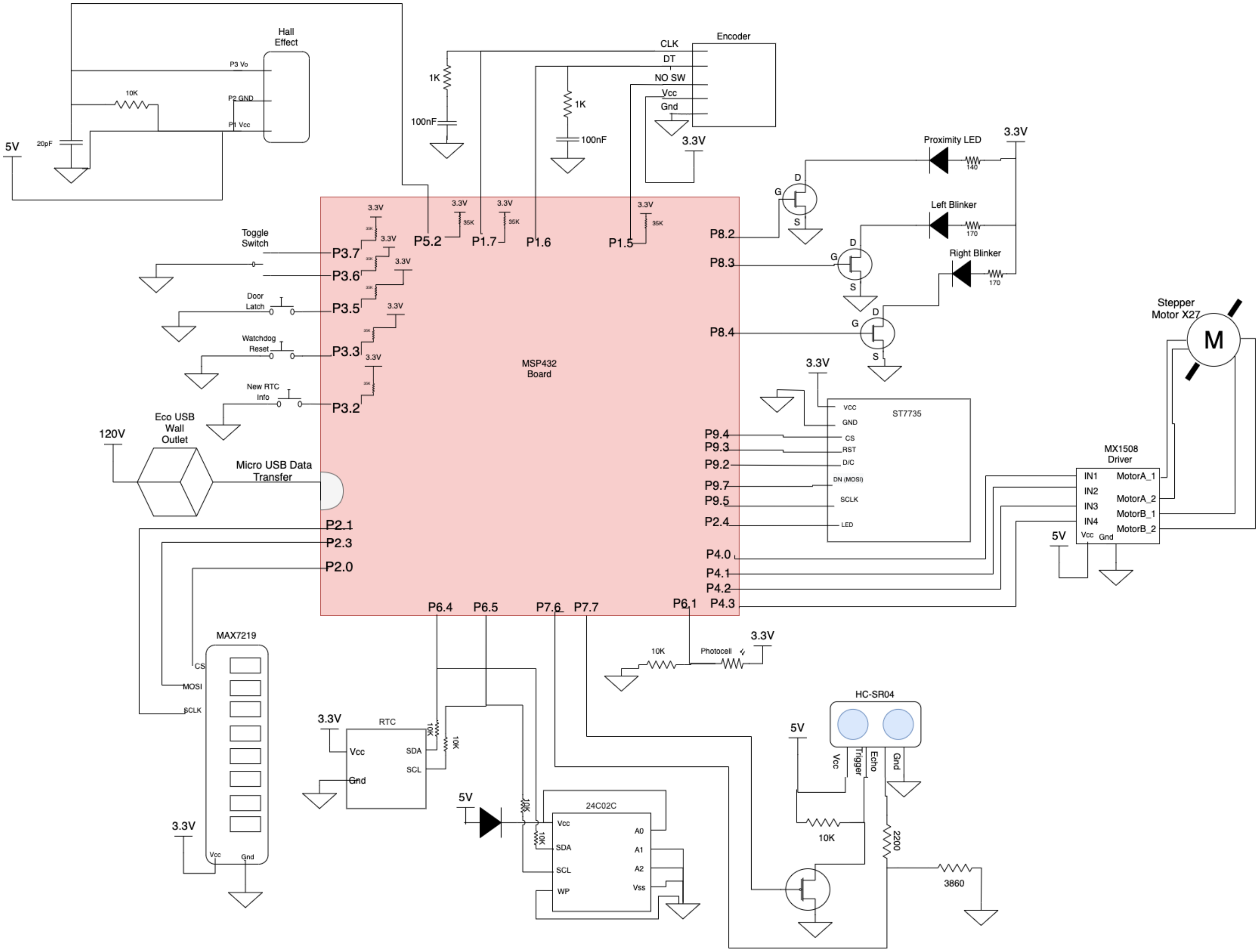


Figure 18: Complete Project Schematic

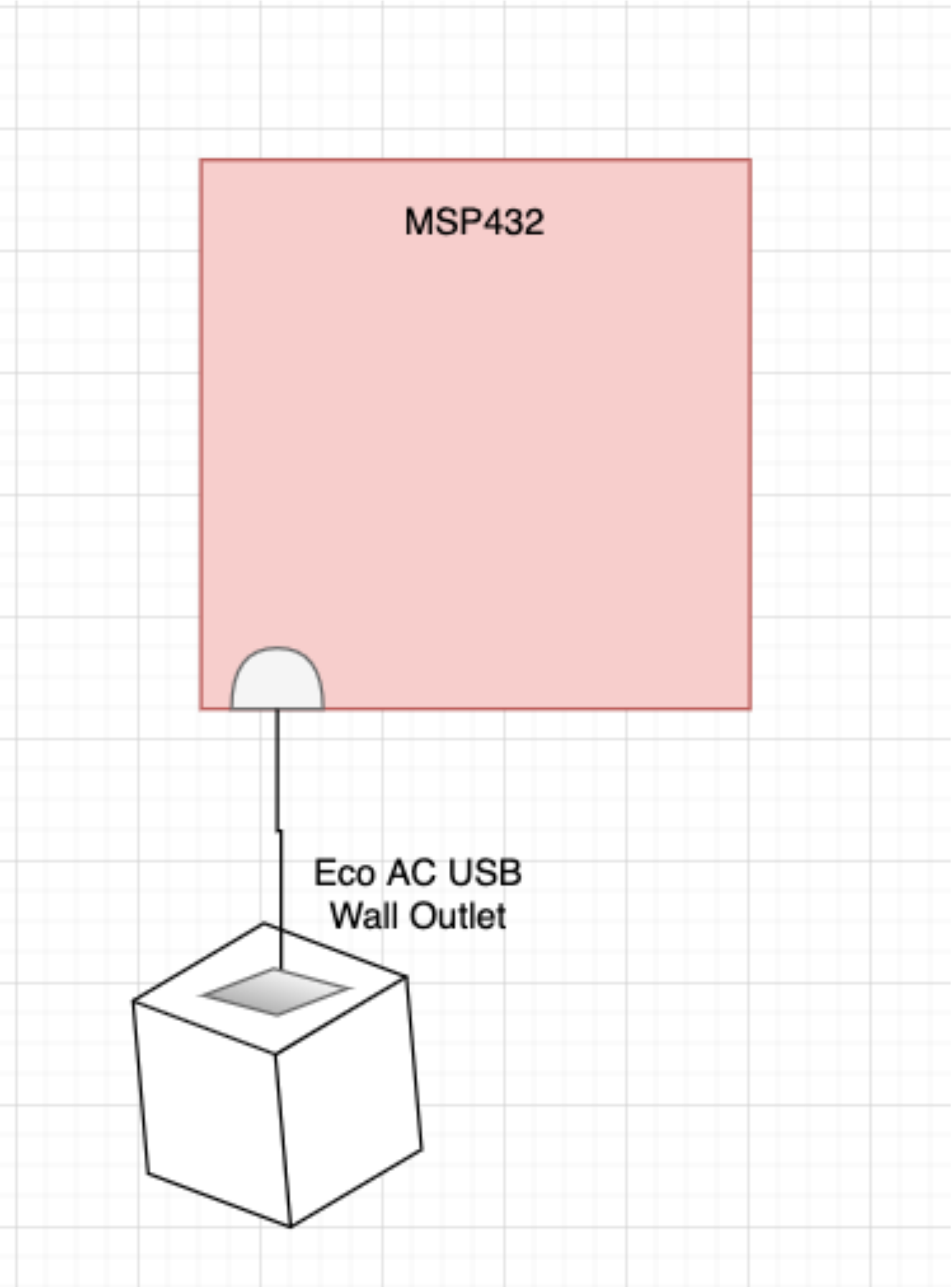


Figure 2: Power Supply Schematic

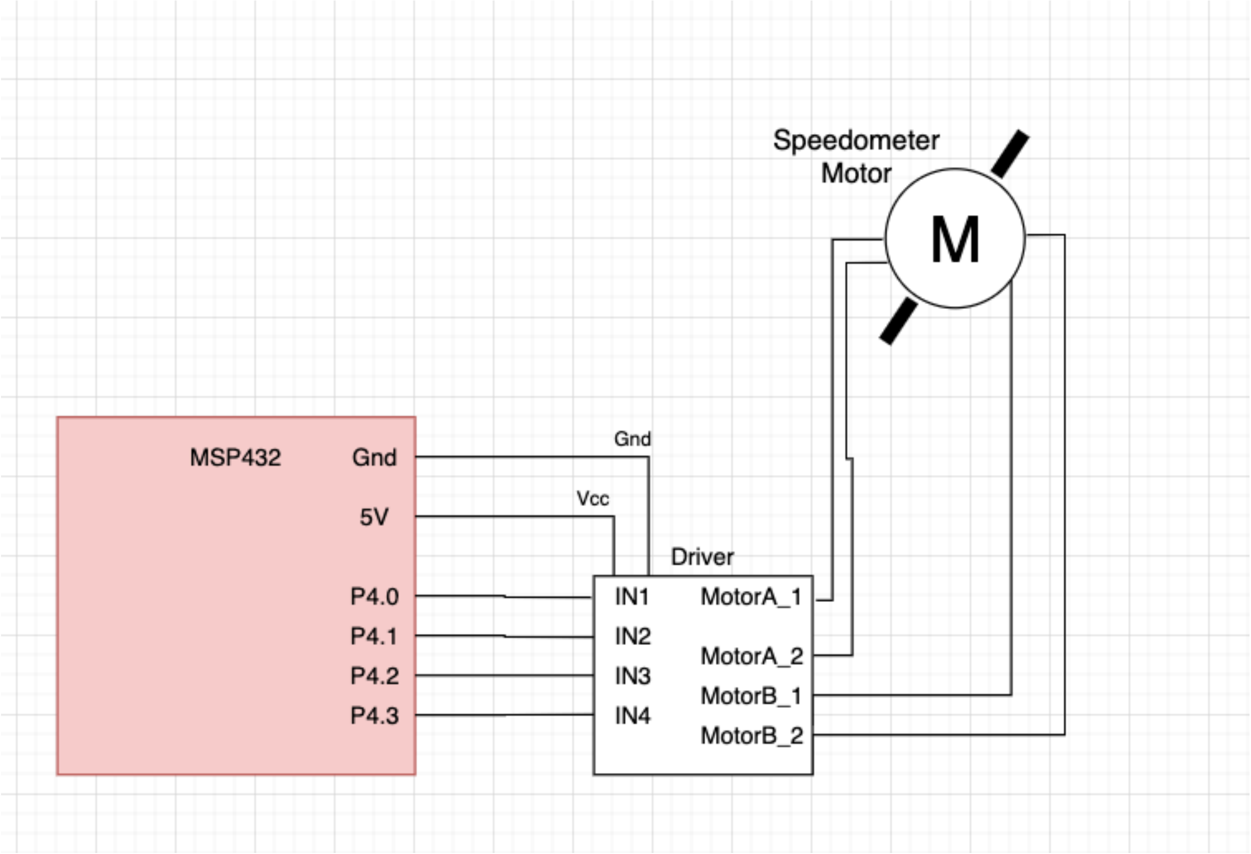


Figure 3: Stepper Motor Schematic

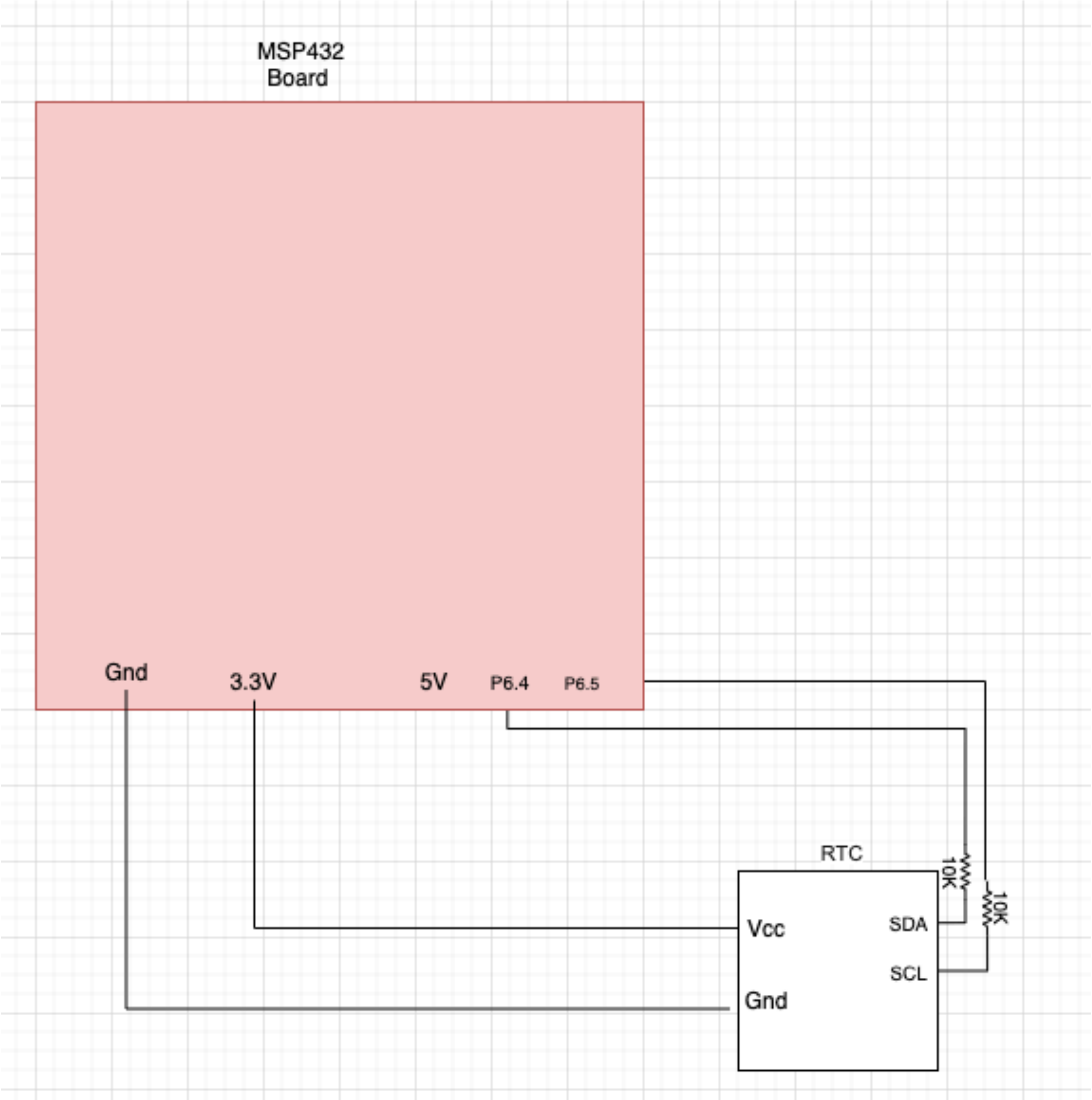


Figure 4: RTC Schematic

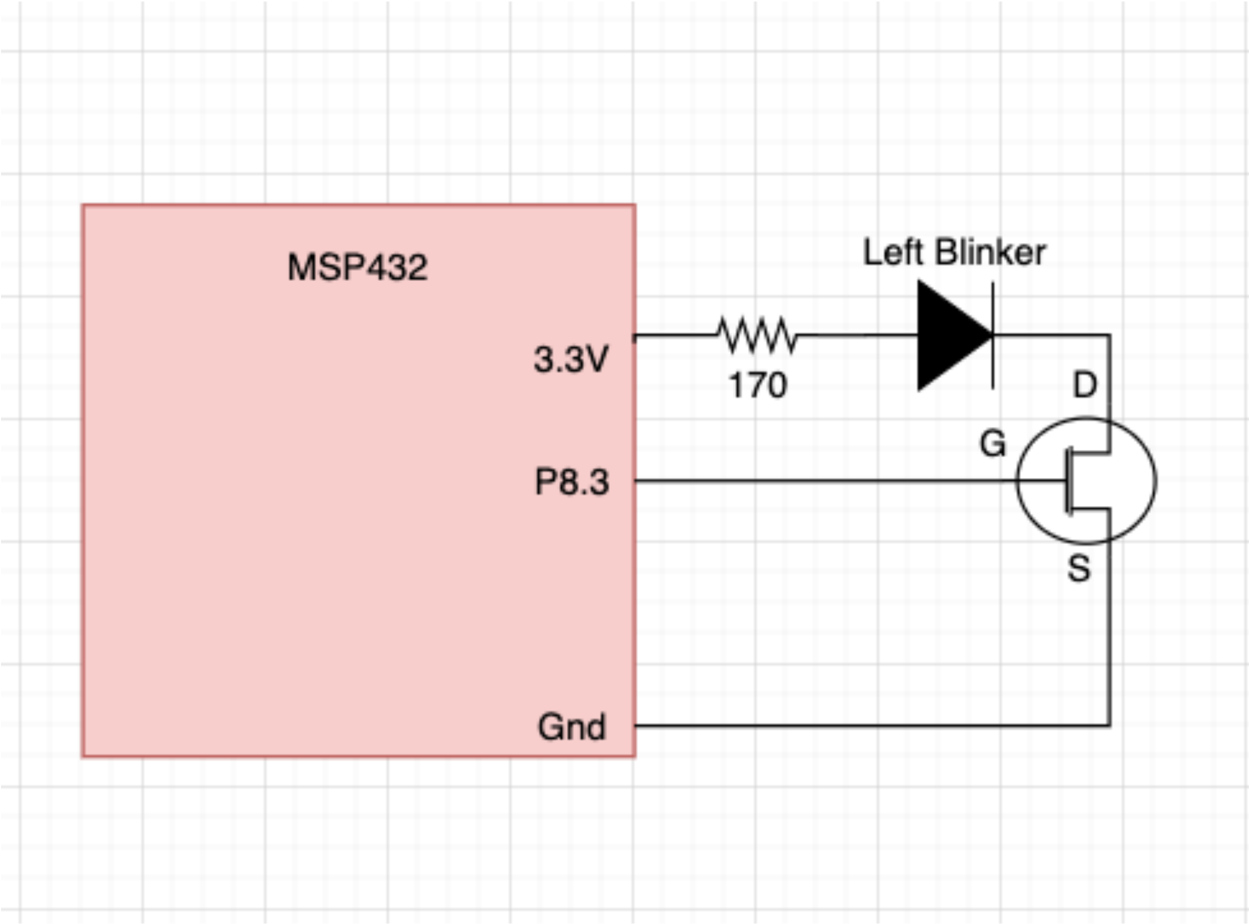


Figure 5: Left Blinker Schematic

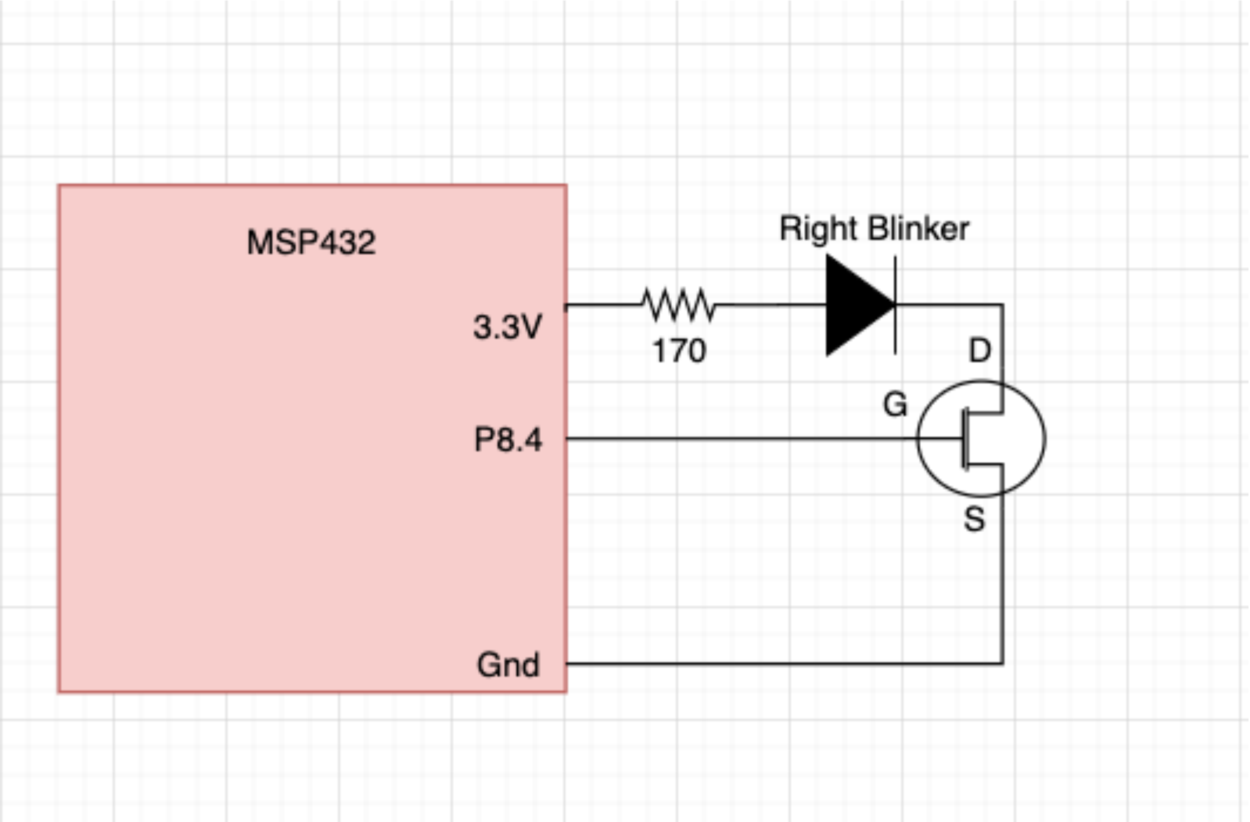


Figure 6: Right Blinker Schematic

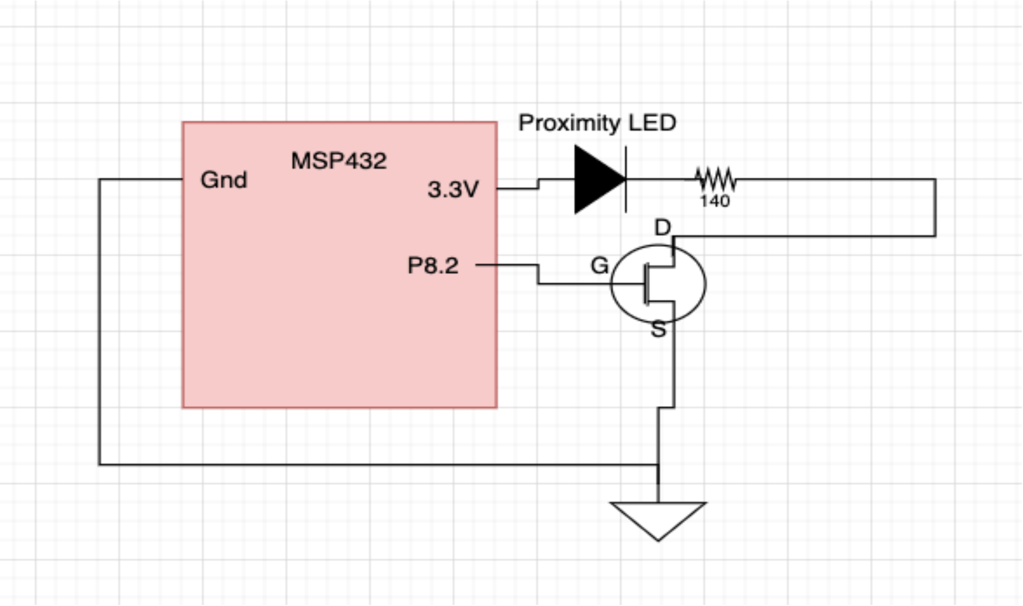


Figure 7: Proximity LED Schematic

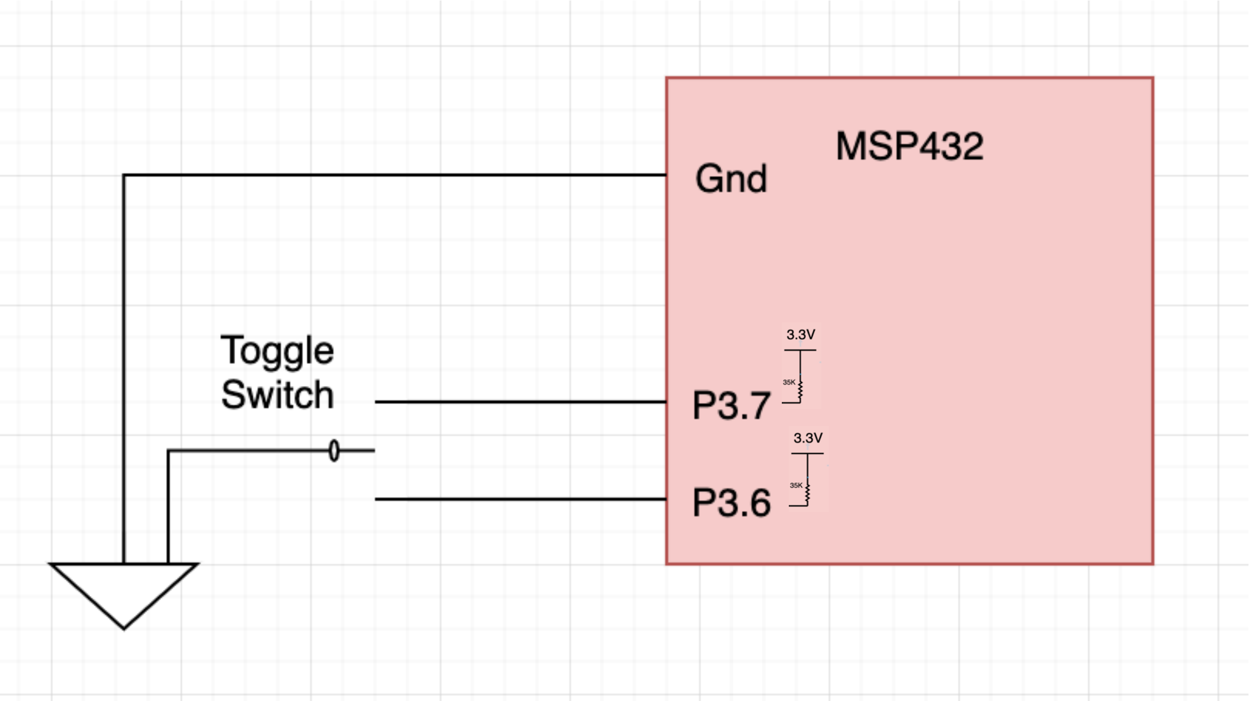


Figure 8: Blinker Toggle Switch Schematic

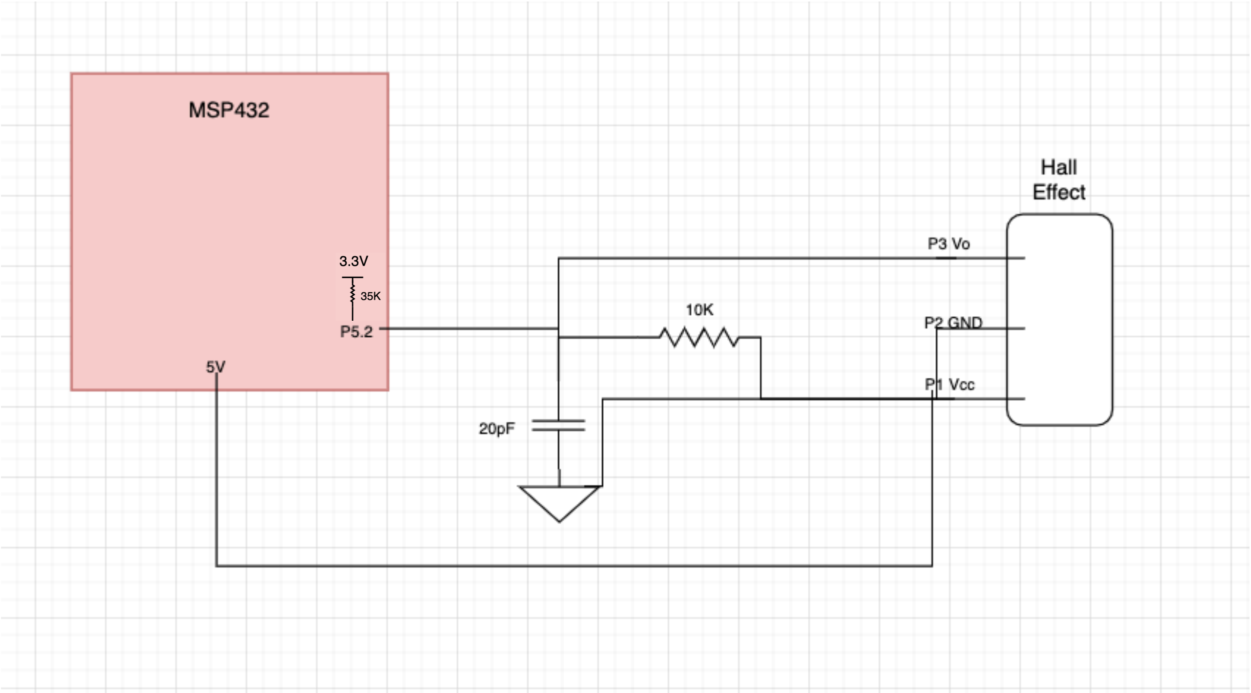


Figure 9: Hall Effect Schematic

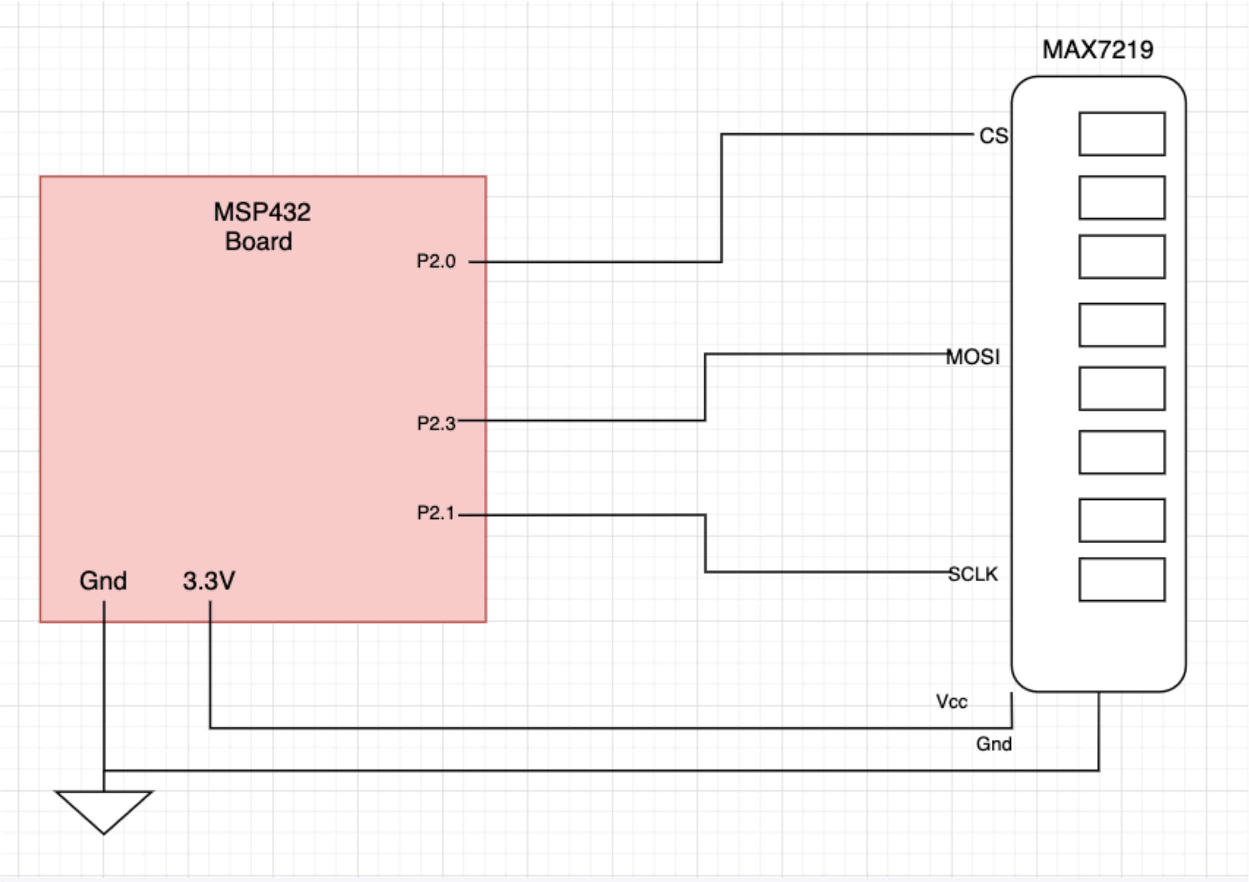


Figure 10: MAX219 8 Character 7 Segment Schematic

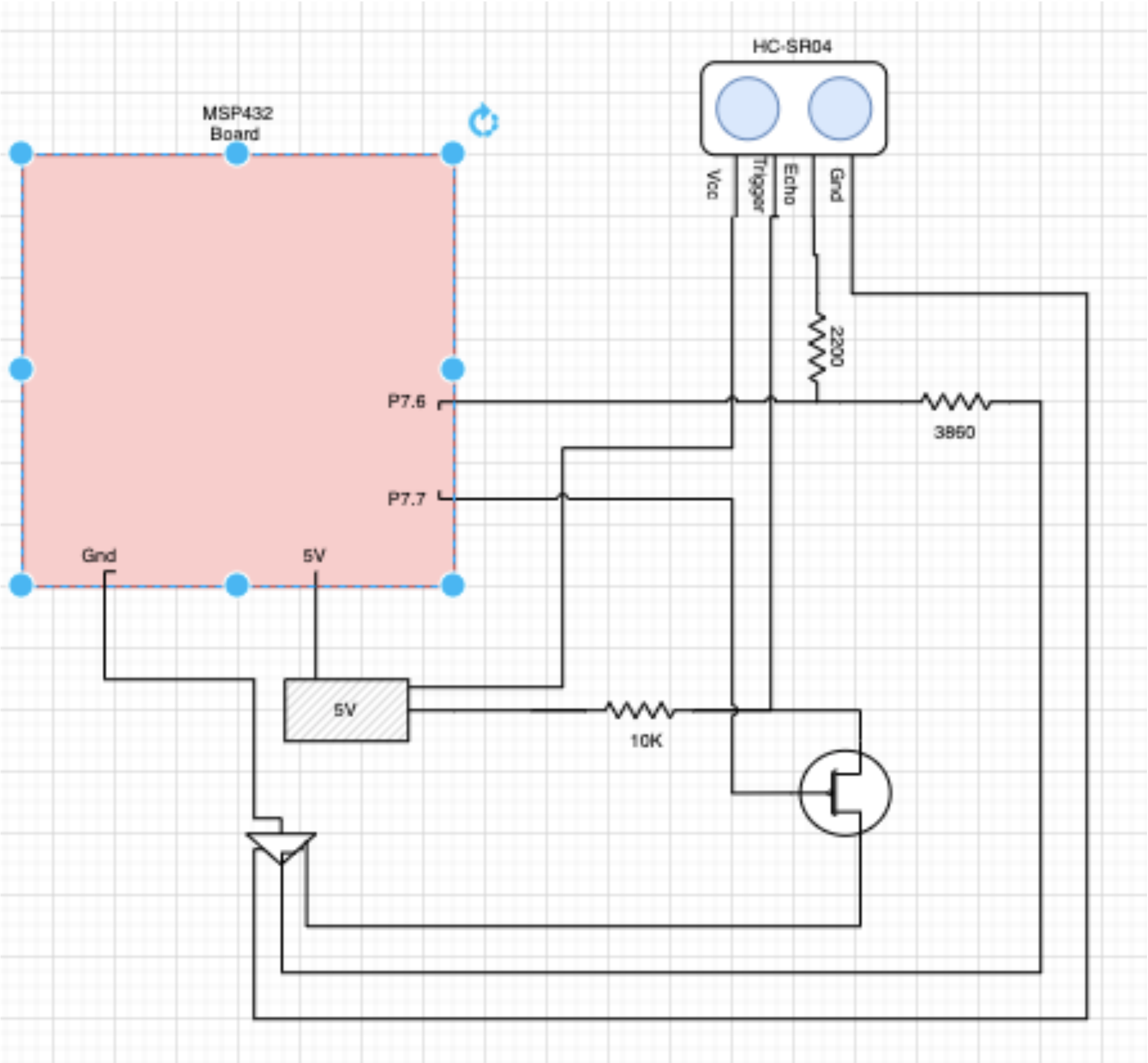


Figure 11: Proximity Sensor Schematic

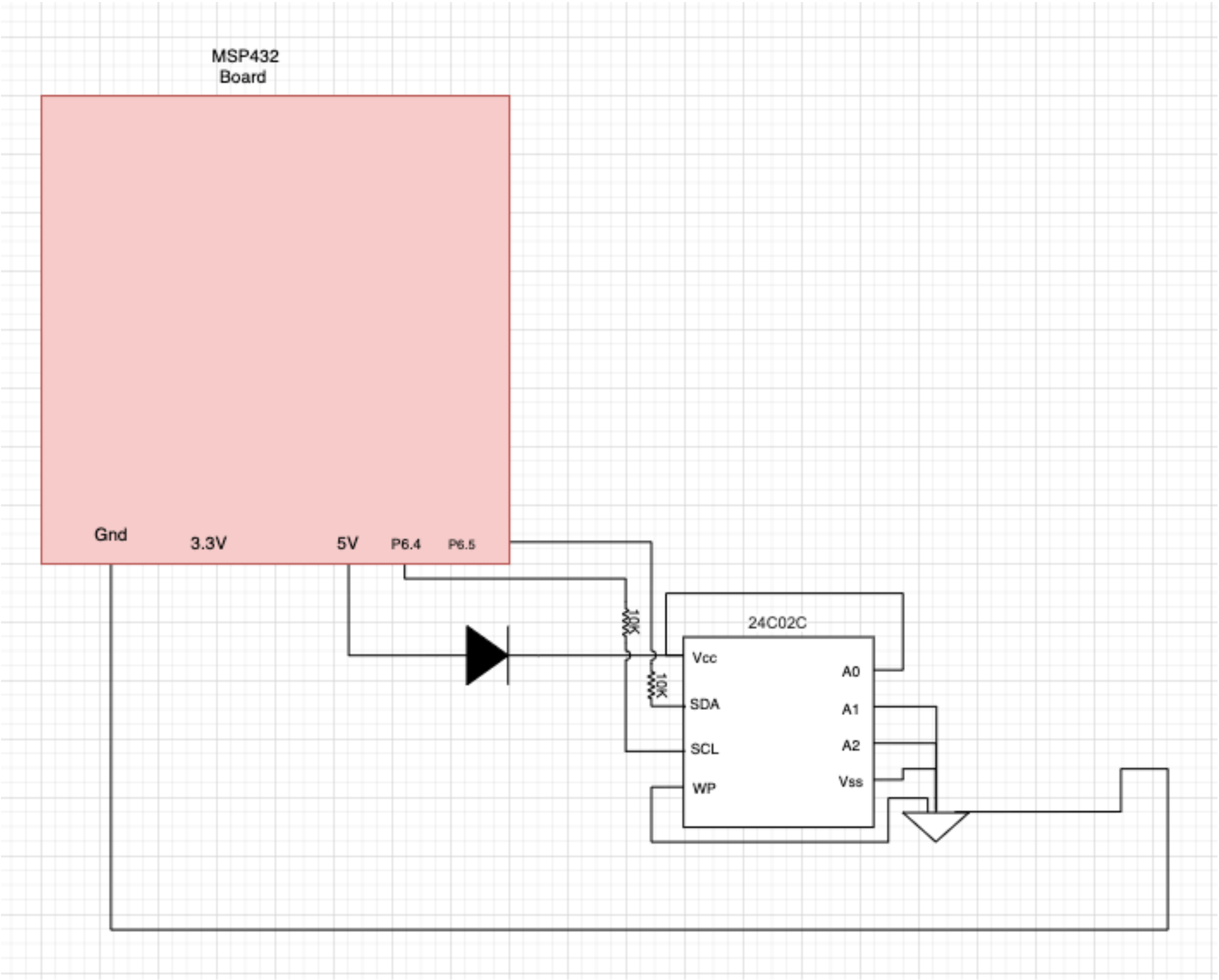


Figure 12: EEPROM Schematic

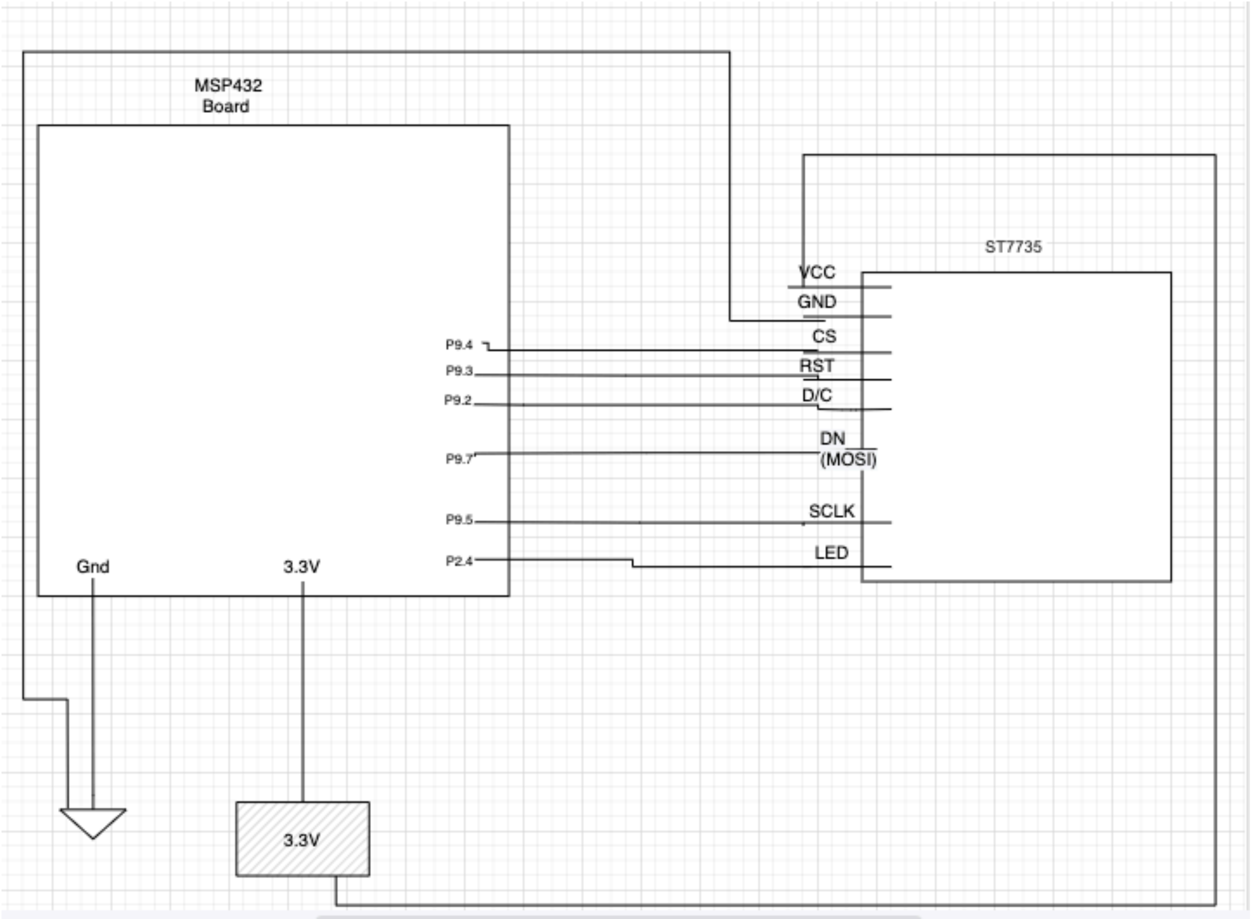


Figure 13: ST7735 LCD Schematic

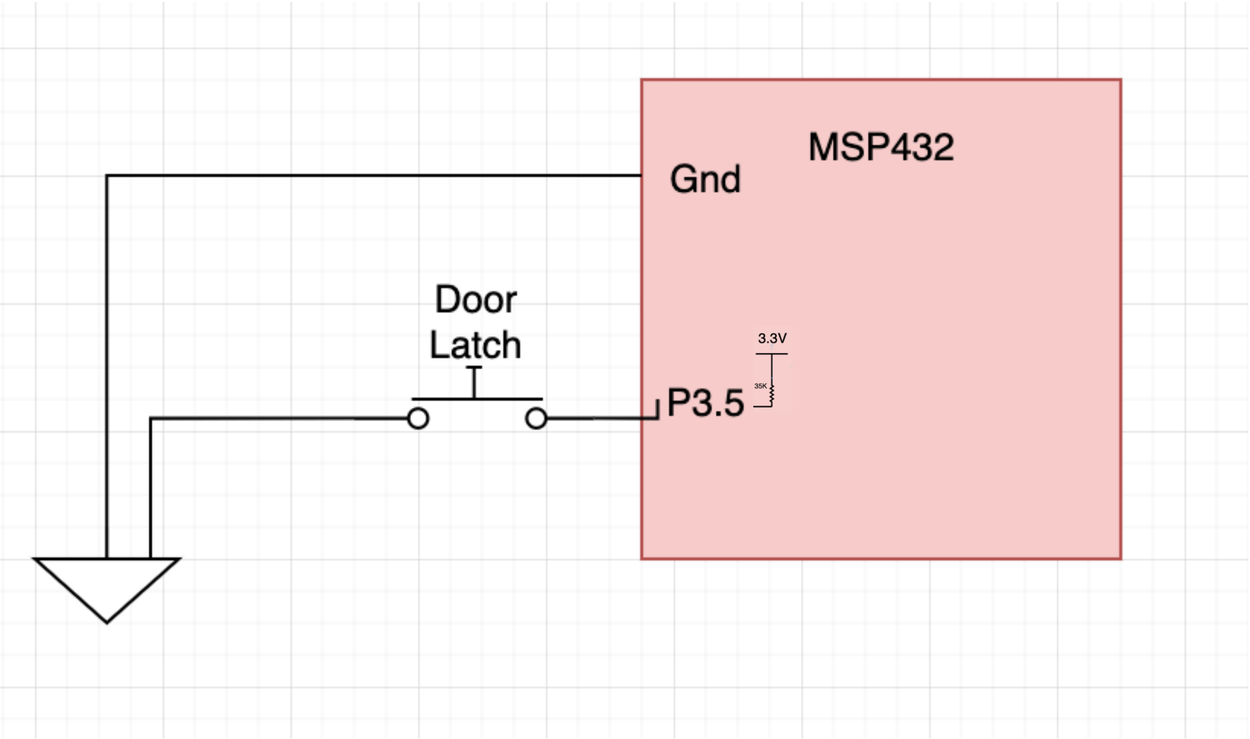


Figure 14: Door Latch Button Schematic

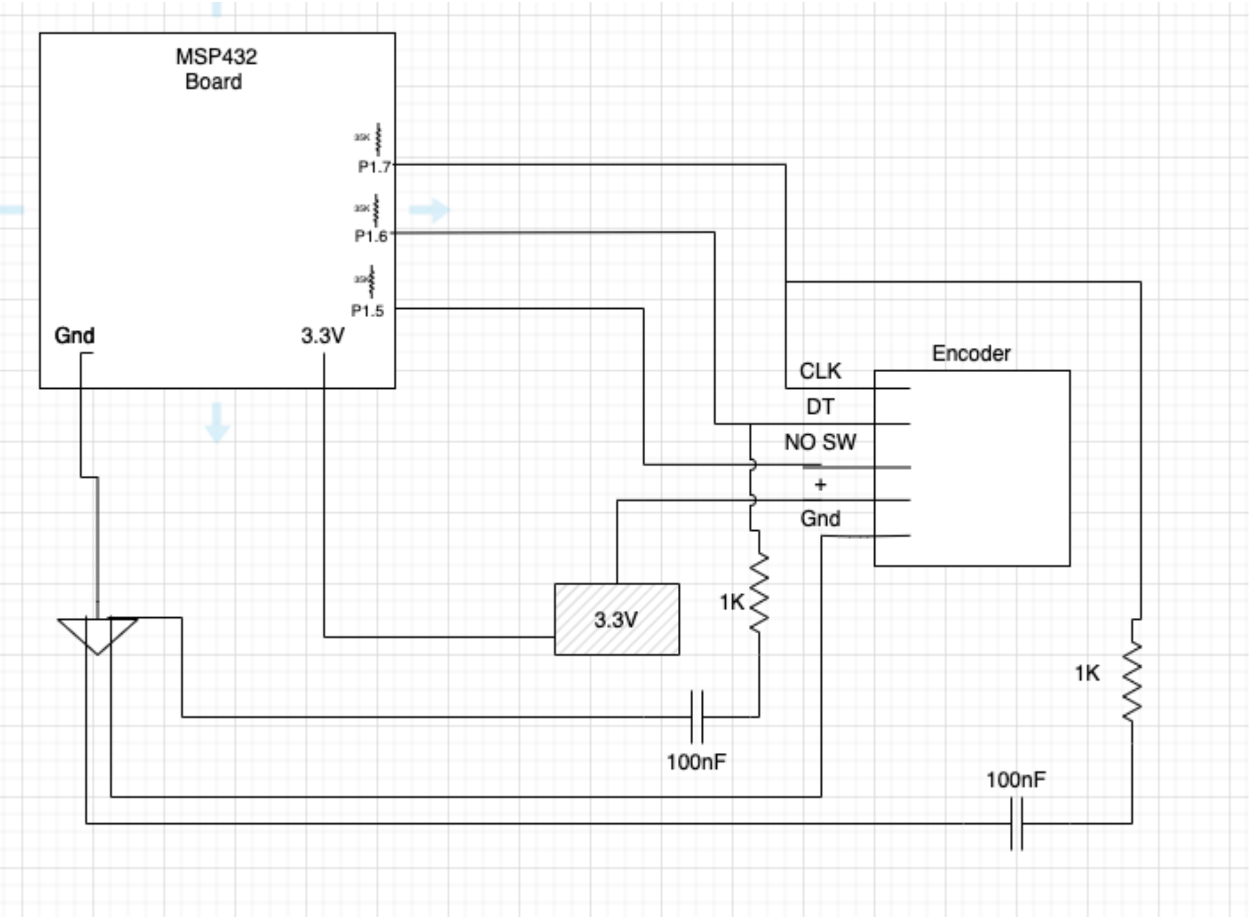


Figure 15: Encoder Schematic

6.1.1.1 Photoresistor Schematiic

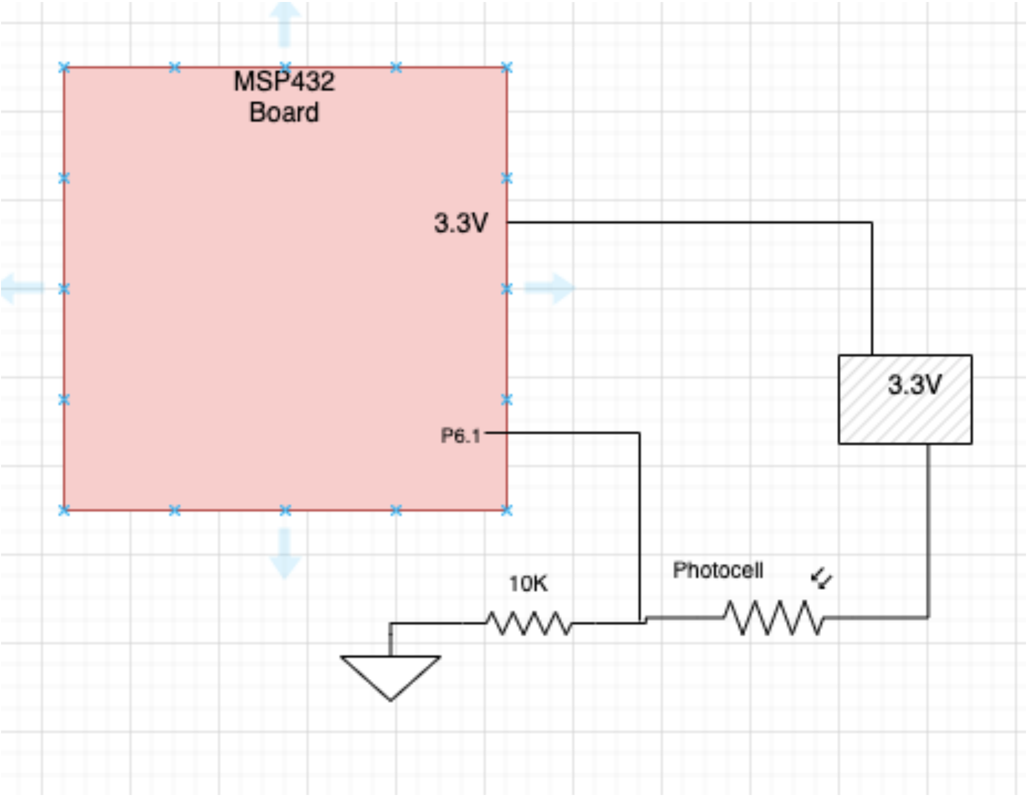


Figure 16: Photoresistor Schematic

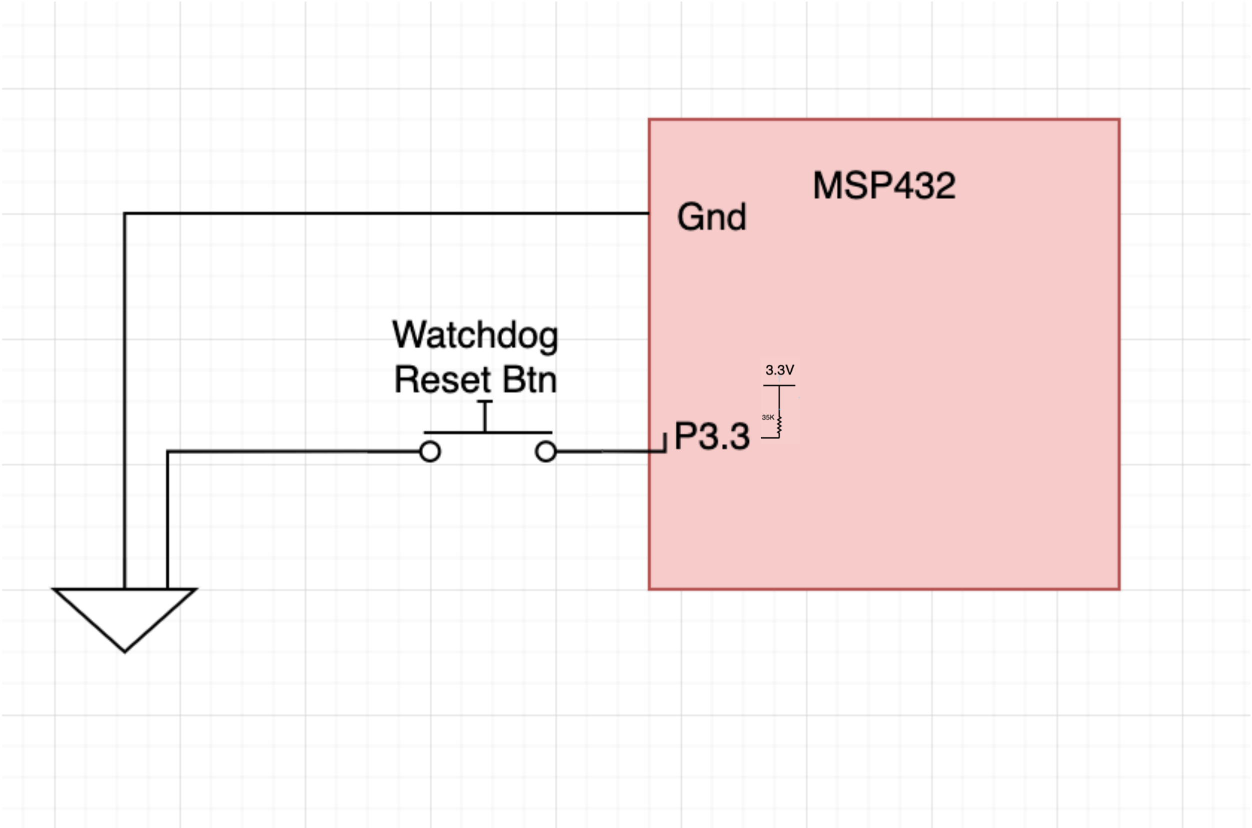


Figure 17: Watchdog Reset Schematic

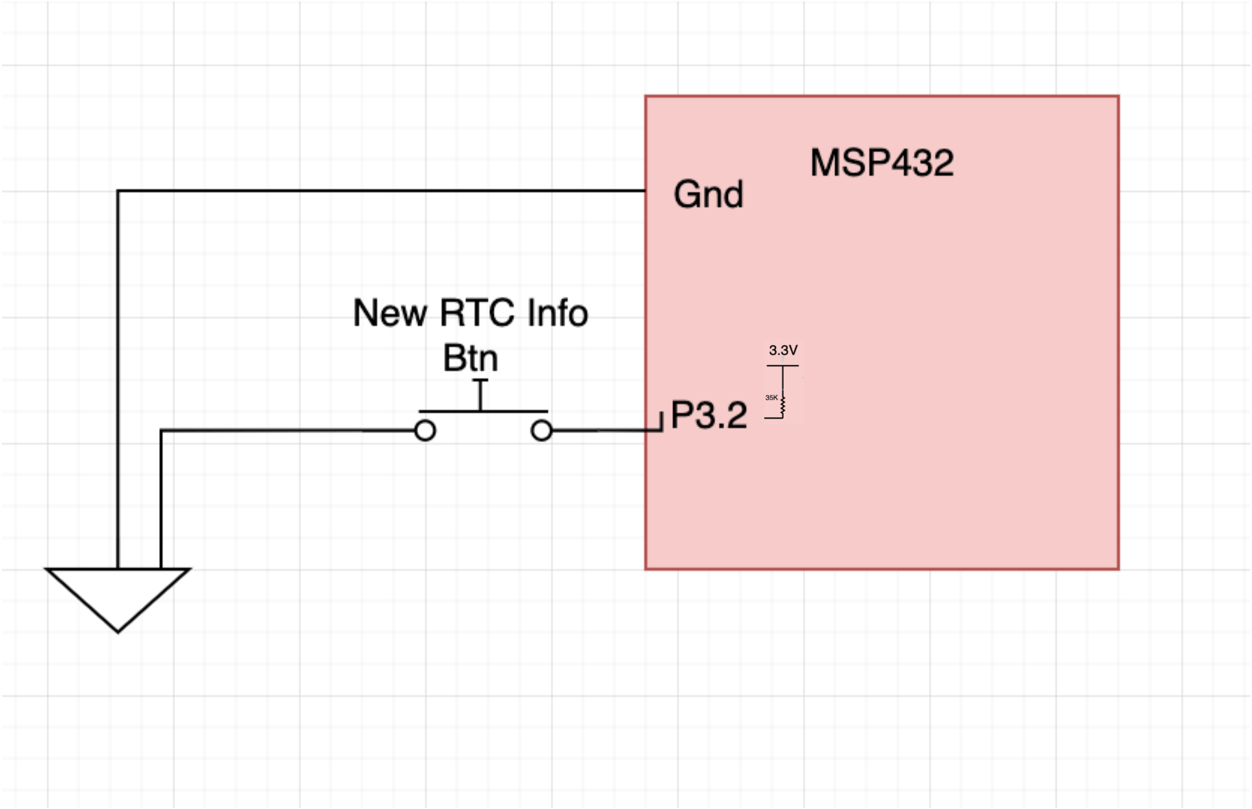


Figure 18: Watchdog Reset Schematic

Appendix B – Bill of Materials

Major Components Included in Kit/Lab

Component	Supplier	Part #	Price Per Unit	Units Used	Total
ST7735 LCD	Just4Fun	--	$11.95	1	--
LEDs	Just4Fun	--	$0.75	3	--
Rotary Encoder	Just4Fun	--	$1.95	1	--
Photoresistor	Just4Fun	--	&1.95	1	--
Tactile Switch	Just4Fun	--	$1.99	2	--
24C02C EEPROM	Just4Fun	--	$0.32	1	--
HC-S04 Proximity Sensor	Just4Fun	--	$5.71	1	--
MAX219 7-Seg	Just4Fun	--	$5.15	1	--
Stepper Motor X27	Just4Fun	--	$4.95	1	--
OH-137 Hall Effect Sensor	Just4Fun	--	$1.95	1	--
RTC IC	Just4Fun	--	$4.05	1	--

Summary of Minor Components

Component	Units Used	Total Cost
Transistors	4	--
Resistors	7	--
Capacitors	3	--

Extra Components

Component	Supplier	Part #	Price Per Unit	Units Used	Total
Toggle Switch	APIELE	APIELE-TOGGLE	$3.66	1	$3.66
Glow-In-The-Dark PLA 3D Printing Filament	Amolen	3D-GID-GN1	$0.026 / g	10	$0.27
Tactile Pushbutton	TE Connectivity	1825967-2	$0.24	1	$0.24
Dupont Wires	Elegoo	--	$0.058	48	$2.79
Straight Wires	AUSTOR	AMA-18-580	$0.019	22	$0.43
Heat Shrink	Eventronic	ET1002	$0.012	24	$0.30
Pine Wood	--	--	--	--	--
Screws	Menards	--	$0.19	24	$4.56
Washers	Menards	--	$0.08	6	$0.48
Nuts	Menards	--	$0.13	6	$0.78
Bolts	Menards	--	$0.13	6	$0.78
Spray Paint	Rustoleum	2006618774	$7.30	2	$14.60
Total					**$25.52**

Appendix D – Source Code

Project.h

```
/*
 * Project.h
 *
 *  Project Library
 *     Author: 7jorchay
 */

#ifndef PROJECT_H_
#define PROJECT_H_

#define TOGGLE_P20  P2 -> OUT ^= BIT0
#define TOGGLE_Left_Blinker  P8 -> OUT ^= BIT3
#define TOGGLE_Right_Blinker  P8 -> OUT ^= BIT4
#define BLINKERS_OFF P8 -> OUT &= ~(BIT3 | BIT4)

#define CIRCUMFERENCE_6ftDiameter .0072       // 6(ft) * pi
#define CIRCUMFERENCE_3inRad .0002974   //circumference in miles of 3 inch radius wheel

#define slaveAddrRTC 0x68
extern volatile int Port1_interruptFlag;
extern volatile int Enter_Date_Time_Btn_count;

extern volatile uint8_t New_RTC_Time_Hour_tens;
extern volatile uint8_t New_RTC_Time_Hour_ones;
extern volatile uint8_t New_RTC_Time_Minute_tens;
extern volatile uint8_t New_RTC_Time_Minute_ones;

extern unsigned char Time_toSend[4];
extern unsigned char Date_toSend[5];

extern volatile uint8_t New_RTC_Date_Day_ones;
extern volatile uint8_t New_RTC_Date_Date_tens;
extern volatile uint8_t New_RTC_Date_Date_ones;
extern volatile uint8_t New_RTC_Date_Month_tens;
extern volatile uint8_t New_RTC_Date_Month_ones;
extern volatile uint8_t New_RTC_Date_Year_tens;
extern volatile uint8_t New_RTC_Date_Year_ones;

void Clock_Init48MHz(void);
void Enable_Interrupts(void);
void PORT1_IRQHandler(void);
void PORT3_IRQHandler(void);
void PORT5_IRQHandler(void);
void TA3_0_IRQHandler(void);
void TA0_0_IRQHandler(void);
void TA1_0_IRQHandler(void);
void TA2_0_IRQHandler(void);
void Project_init(void);
void reset_Encoder_Counts(void);
void control_Cursor(void);
void DoorAjar_Btn_HeldDown(void);
void CONTROL_BLINKERS();
void PROXIMITY(void);
void main();
```

```
#endif /* PROJECT_H_ */
```

Project.c

```
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/*
 *              INCLUDES
 */
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "math.h"
#include "SysTick_Library.h"
#include "ST7735.h"
#include "TIMERA.h"
#include "Clocks.h"
#include "Menus.h"
#include "EncoderFunctions.h"
#include "ProjectButtons.h"
#include "RTC_ProjectFunctions.h"
#include "I2C_ProjectFunctions.h"
#include "Stepper.h"
#include "MAX219.h"
#include "Hall_Effect.h"
#include "HC-SR04.h"
#include "Photocell.h"
#include "Songs.h"
#include "EEPROM.h"
#include "Project.h"

////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/*
 *              GLOBAL VARIABLES
 */
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
volatile int timerA3_0_interruptFlag = 0;//gloabl interrupt flag for timerA3_0, 0 instance can only time
volatile int timerA1_0_interruptFlag = 0;//global interrupt flag for TIMERA1_0

volatile int Timer32_interruptFlag = 0;             //global interrupt flag for TIMER32, one second timer

volatile int leftBlinker_Switch_held = 0;//flag set when toggle switch position checked corresponding to left blinker
volatile int rightBlinker_Switch_held = 0;//flag set when toggle switch position checked corresponding to right blinker
//these are used to set the toggle flag in TIMERA3 interrupt
volatile int Open_Door_Btn_interruptFlag = 0;//global flag for open dorr button pin
volatile int Open_Door_Btn_heldDown = 0;//continuall reevaluted in while loop to determine if the open door button is still held down

volatile int NewRTC_Info_Btn_interruptFlag = 0;                 //global flag for New_RTC_info button
volatile int Enter_Date_Time_Btn_count = 0;     //Count for how many times the New_RTC_info button has been pressed before reset

volatile int WatchdogReset_Btn_interruptFlag = 0;//global flag for when the watchdog button is pressed

volatile uint8_t New_RTC_Time_Hour_tens = 0;//global variables holding the desired new values for time to send to RTC
volatile uint8_t New_RTC_Time_Hour_ones = 0;                                  //one digit at a time
volatile uint8_t New_RTC_Time_Minute_tens = 0;
volatile uint8_t New_RTC_Time_Minute_ones = 0;

volatile uint8_t New_RTC_Date_Day_ones = 0; //global variables holding byte data for desired new RTC Date
volatile uint8_t New_RTC_Date_Date_tens = 0;
volatile uint8_t New_RTC_Date_Date_ones = 0;
volatile uint8_t New_RTC_Date_Month_tens = 0;
volatile uint8_t New_RTC_Date_Month_ones = 0;
volatile uint8_t New_RTC_Date_Year_tens = 0;
volatile uint8_t New_RTC_Date_Year_ones = 0;


unsigned char Time_toSend[4] = { 0, 0, 0, 0 };//global array, initialized with zeros, set bytes to new desired time data
unsigned char Date_toSend[5] = { 0, 0, 0, 0, 0 };//global array, initialized with zeros, set bytes to new desired date data
volatile int alarm_count = 0;//global alarm count to determine which byte is written or read to EEPROM
volatile int alarm_entered = 0;         //global flag for whether an alarm was sent to EEPROM

volatile int TOGGLE_Left_Blinker_flag = 0;//global flag set when corresponding left blinker pin from toggle switch is read low
volatile int TOGGLE_Right_Blinker_flag = 0;       //global flag set when corresponding right blinker pin from toggle switch is read low
volatile int BLINKERS_OFF_flag = 0; //global flag for when toggle switch is not set either direction
//volatile int alarms_printed;

/////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////
/*
 *            FUNCTION PROTOTYPES in .h
 */
/////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////
void main()    //Project_main
{
            int i;
            Watchdog_halt();                                                //halts watchdog

            Project_init();                                     //initializedcomponents and timers for the project

            Enable_Interrupts();                                //enable interrupts for project

            Startup_Menu();                                     //calls a function to display startup menu on reset


  while (1)
  {
//read inputs

                        if (alarm_count == 5) {                 //if alarm count is equal to 5, reset it
                                    alarm_count = 0;                        //only 5 alarms are stored
                        }

                        ADC_Reading();                                                                          //read ADC value from photoresistor
                        change_TIMERA0_1_dutyCycle();//change the LED backlight brightness depending on ADC value

                        leftBlinker_Switch_held = (!(P3IN & BIT6 ));//read tollge switch value for blinkers
                        rightBlinker_Switch_held = (!(P3IN & BIT7 ));

                        Open_Door_Btn_heldDown = (!(P3IN & 0x20));//sets global value to the state of the button for door ajar

                        for (i = 0; i < 15; i++) {
                                    Debounce_DoorOpen_Btn();            //global debounce for door open button
                                    Debounce_EncoderButton(); //global debounce flag for encoder button
                                    Debounce_NewRTC_Info_Btn();     //global debounce flag for button that enters new RTC data
                                    Debounce_WatchodgReset_Btn();//global debounce flag for watchdog reset button button;
                        }

//respond to inputs
                        if (Open_Door_Btn_interruptFlag == 1) //interrupt occurs for door ajar button
                                                {
                                    if (Open_Door_Btn_pressed == 1)       //confirms debounce successful
                                                            {
                                                DoorAjar_Btn_HeldDown();                            //display door ajar on MAX219
                                                Open_Door_Btn_pressed = 0;             //reset debounce flag
                                                Open_Door_Btn_interruptFlag = 0;//reset interrupt flag once button is no longer held down
            }
            MAX219_SPI_init_Decode();//reset MAX219 to decode mode for speed display
        }

        if (Encoder_Btn_interruptFlag == 1) {//interrupt occurs on encoder button pin
            if (Encoder_Btn_pressed == 1) {                 //confirm encodr debounce
                MENU();                                             //the encoder button controls
MENUS
            }
        }

        if (NewRTC_Info_Btn_interruptFlag == 1) {//interrupt occurs on button for entering new RTC information
            if (NewRTC_Info_Btn_pressed == 1) {                     //confirm debounce

                if ((menu == 2) || (menu == 3)) {//will only have an effect if menu is currently
                    Enter_Date_Time_Btn_count++;            //change time or date

    //increment button count to control the digit changed
                    NewRTC_Info_Btn_interruptFlag = 0;      //clear flags
                    NewRTC_Info_Btn_pressed = 0;
                    reset_Encoder_Counts();     //reset encoder counts to begin counting again
                }
            }
        }

        if (WatchdogReset_Btn_interruptFlag == 1) {     //interrupt occurs on watchdog interrut flag
            if (WatchdogReset_Btn_pressed == 1) {           //debounce confirmed
                Watchdog_init();//initialize watchdog reset, 4 second timeout
                WatchdogReset_Btn_interruptFlag = 0;//technically never reached, but reset anyway in case
                WatchdogReset_Btn_pressed = 0;//i'd like to be able to interrupt reset

            }
        }

        CONTROL_BLINKERS();  //every half second, the timer will set a flag for the blinker
                                    //corresponding to toggle switch checked above
        if (Timer32_interruptFlag == 1) {//intrval interrupt for every second
            RPMs = measure_RPMs(Revolution_count);          //measure RPMs

            Timer32_interruptFlag = 0;                              //reset flag
            Revolution_count = 0;//reset revolution counts for the next second calculation

            Current_Speed_mph = measure_Speed(RPMs,  //calculate speed from RPMs
            CIRCUMFERENCE_6ftDiameter);

            if (Current_Speed_mph < 100) {                          //display speed
                MAX219_SPI_TransferData(0x05, Current_Speed_mph / 10);//tens place is int speed / 10
                MAX219_SPI_TransferData(0x04, Current_Speed_mph % 10);//ones place is remainder of speed / 10
                MAX219_SPI_TransferData(0x06, 0);//if less than 100, hundreds place must be zero
            }

            else if (Current_Speed_mph >= 100) {        //display greater than 100
                MAX219_SPI_TransferData(0x05, (Current_Speed_mph / 10) % 10);
                MAX219_SPI_TransferData(0x04, Current_Speed_mph % 10);
                MAX219_SPI_TransferData(0x06, 1);//just print one in hundres place, speed will not go above 199
            }
            speed_Dif = (Current_Speed_mph - lastSpeed);//before setting speedometer position

    //calculate the difference of last and current speed
            if (RPMs == 0) {                                //if the RPMs calculated are zero
                if (RPMs_areZero == 0) {//if this is not a repeated zero calculation

                    //TIMER_A2->CCR[2] = 6000;
                    //SysTick_delay_ms(10);
                    Rotate_Stepper1_Backward_s_Steps(120);//turn speedometer to zero position
                    RPMs = 0;
        //reset RPMs
                    RPMs_areZero = 1;//set RPMs are zero flag so this is not repeated continually while spped is
zero
                }

            }

            else if (RPMs > 2) {
                RPMs_areZero = 0;           //reset RPMs_are_zero when they are not
                if (firstTime_done == 0) {//if it is not the first time speed is being sent to speedmoeter
                    firstTime_done = 1;                     //set the first time flag

                    //TachometerSpeed(RPMs);
                    //TIMER_A2->CCR[2] = 3000;
                    //SysTick_delay_ms(10);
                    Rotate_Stepper1_Forward_s_Steps(Current_Speed_mph); //set analog speed by moving
motor number of mph in steps

                    RPMs = 0;                                   //reset RPMs after
using it

                    if (Current_Speed_mph > 85) {//if greater than 85, end alarm
                        Read_RTC_Store_Alarm_EEPROM(alarm_count);
                        alarm_count++;
    //increment alarm count
                    }
                } else {
                    if (speed_Dif == 0) {//if the same speed is measured twice ina row
                        RPMs = 0;                                       //do
nothing, reset RPMs
                    } else if (speed_Dif > 0) {     //if the current speed is greater than last
//                      TachometerSpeed(RPMs);
//                      //TIMER_A2->CCR[2] = 2500;
//                      SysTick_delay_ms(10);
                        Rotate_Stepper1_Forward_s_Steps(speed_Dif);     //stepper moves
forward the number of steps difference

                        RPMs = 0;                               //reset RPMs after
using

                        if ((speed_Dif + lastSpeed) > 85) {  //now if this calculation is greater than 85
                            Read_RTC_Store_Alarm_EEPROM(alarm_count);//send speed alarm
                            alarm_count++;

                        }
                    } else if (speed_Dif < 0) {     //if the current speed is less than the last

//                      TachometerSpeed(RPMs);
//                      SysTick_delay_ms(10);
                        Rotate_Stepper1_Backward_s_Steps(abs(speed_Dif));//rotate speedometer
number of steps backward

                        RPMs = 0;//equal to the absolute value of speed difference
                    }
                }
            }

            lastSpeed = Current_Speed_mph;//set last speed equal to current speed for next iteration
            //TIMER_A2->CCR[2] = 0;
        }
//menu display control
        switch (menu) {
```

```
                    case 0: {

                              if (timerA3_0_interruptFlag == 1) {  //do this if startup menu is running and a half second has elasped

                                        Read_RTC_Print_Time_toLCD();                        //read and print the time
                                        Read_RTC_Temperature_Print_toLCD();           //read and print the temperature in Fahrenheit

                                        PROXIMITY();                                  //measure distance from rear of the vehicle
                                        timerA3_0_interruptFlag = 0;//reset half second interval flag
                              }

                              break;
                    }
                    case 1: {                                                                              //menu 1: interface menu
                              control_Cursor();//control cursor using menu and encoder turn count conditions
                              break;
                    }
                    case 2: {                                                                              //menu 2: change time menu
                              control_Cursor();//control cursor using menu and encoder turn count conditions
                              break;
                    }
                    case 3: {                                                                              //menu 3: change date menu
                              control_Cursor();                                     //control cursor for this menu
                              break;
                    }
                    case 4: {                                                                              //menu 4: alarm menu
                              break;                                      //no cursor, an encoder press resets to startup menu
                    }

                    default: {
                              break;
                    }

                    }
  }
}

void Project_init()
{
  Clock_Init48MHz();                //MCLK 48 MHz | SMCLK 12MHz
  AClock_Divide4();                 //Divides ACLK by 4 for interval timers

  SysTickInit_NoInterrupts();        //SysTick for delays

          //TIMERA1_0_oneMSInterval_init();      //1 ms period timer        | TIMERA1_0
  TIMERA3_0_HalfSecondInterval_init(); //1 second interval timer   | TIMERA3_0

  openDoor_Btn_init();              //door latch button          | 3.5 GPIO
          Blinkers_Toggle_Switch_init(); //blinker switch                  | 3.6, 3.7 GPIO
          NewRTC_Info_Btn();

          Encoder_init();        //encoder                 | 1.5, 1.6, 1.7  | GPIO

  Left_Blinker();                   //left blinker            | 8.3 GPIO
  Right_Blinker();                  //right blinker           | 8.4 GPIO
  LED_closeProximity();              //object too close LED      | 8.2 GPIO

  TimerA0_1_PWM_init();      //Backlight LED              | 2.4 | TIMERA0_1
  ST7735_InitR(INITR_REDTAB);      //ST7735                 | 9.2, 9.3 GPIO
                    //                     | 9.4, 9.5, 9.7  | CS GPIO, SCL, SDA

  Stepper1_Motor();  //Stepper Motor 1          | 4.0, 4.1, 4.2, 4.3   | GPIO

          //Tachometer_Init();
		SPI_init();			//MAX219	| 2.1, 2.3 | CLK, SIMO
		MAX219_SPI_init_Decode();				//					| 2.0		| CS


		Hall_Effect_Out_init();		//hall effect output pin	| 5.2 | GPIO

		HCSR04_init();								//proximity sensor	| 7.7, 7.6 |

		Speaker_12MHz(0, 0);					//speaker	| 7.7		| TIMERA_
		//Tachometer_Init();
		ADC_pin_init();
		ADC_StartConversion_WithInterrupts();		//Photocell	| 6.1 | ADC

		Timer32_init();										//interval for each second
		WatchdogReset_Btn();							//initialize watchdogbutton | P3.3

		//not initialized because of I2C: RTC and EEPROM	| 6.4, 6.5	|
}

void DoorAjar_Btn_HeldDown(void)
{

		MAX219_SPI_init_NoDecode();	//set MAX219 to nodecode mode to diplay custom letters
		while (Open_Door_Btn_heldDown == 1)		//while the button is depressed
	{
	   Open_Door_Btn_heldDown = (!(P3IN & 0x20));	//reevaluate
			Print_DoorAjar_MAX219();								//print door Ajar digitally
	}
		blank_MAX219();		//blank the screen (this was for testing mainly, but still works because it is reinitialized in decode mode)
}

void MEASURE_SPEED() {//not used on accident, but could replace the entirety of the speedometer setting
		if (Timer32_interruptFlag == 1) {
			RPMs = measure_RPMs(Revolution_count);

			Timer32_interruptFlag = 0;
			Revolution_count = 0;
			Current_Speed_mph = measure_Speed(RPMs,
			CIRCUMFERENCE_6ftDiameter);
			speed_Dif = (Current_Speed_mph - lastSpeed);

			if (RPMs == 0) {
				if (RPMs_areZero == 0) {
					Rotate_Stepper1_Backward_s_Steps(120);
					RPMs = 0;
					RPMs_areZero = 1;
				}

			}

			else if (RPMs > 2) {
				RPMs_areZero = 0;
				if (firstTime_done == 0) {
					firstTime_done = 1;

					Rotate_Stepper1_Forward_s_Steps(Current_Speed_mph);
					RPMs = 0;

					if (Current_Speed_mph > 85) {
						Read_RTC_Store_Alarm_EEPROM(alarm_count);
						alarm_count++;
					}
				} else {
```

```
                if (speed_Dif == 0) {
                    RPMs = 0;
                } else if (speed_Dif > 0) {
                    Rotate_Stepper1_Forward_s_Steps(speed_Dif);
                    RPMs = 0;

                    if (speed_Dif > 85) {
                        Read_RTC_Store_Alarm_EEPROM(alarm_count);
                        alarm_count++;
                    }
                } else if (speed_Dif < 0) {
                    Rotate_Stepper1_Backward_s_Steps(abs(speed_Dif));
                    RPMs = 0;
                }
            }
        }

        lastSpeed = Current_Speed_mph;

    }
}

void CONTROL_BLINKERS() {
    if (TOGGLE_Left_Blinker_flag == 1) {//if the left blinker flag was set in TA3 handler because of the toggle switch evaluation
        TOGGLE_Left_Blinker;                                        //macro to toggle the LED
        TOGGLE_Left_Blinker_flag = 0;                       //reset flag
    }

    else if (TOGGLE_Right_Blinker_flag == 1) {      //repeat for right blinker LED
        TOGGLE_Right_Blinker;
        TOGGLE_Right_Blinker_flag = 0;
    }

    else if (BLINKERS_OFF_flag == 1) {//if neither toggle position is detected, blinker off flag is set
        BLINKERS_OFF;                                                           //macro to turn off both LEDs
        BLINKERS_OFF_flag = 0;                                                  //reset for next iteration
    }
}

void PROXIMITY() {

    measure_Period();//measure the period of timerA capture for proximity sensor

    if (object_DistanceFromSensor < 15) {//if the distance measured is less than 15 inches

        while (object_DistanceFromSensor < 15) {//repeat this while the object is sensed

            if (alarm_entered == 0) {//only want to enter one alarm each time, so this evaluates a flag for that
                Read_RTC_Store_Alarm_EEPROM(alarm_count);//store alarm in EEPROM
                alarm_count++;
    //increment count
                alarm_entered = 1;                      //set the flag after entering alarm
                }

            P8->OUT |= BIT2;//blue LED on 8.2 is lit while an object is near the rear

            displayParkingWarning();//displays a visual warning on screen each second
            Bells(5);                                               //audible alarm from piezo speaker
            Output_Clear();
    //clear output

            measure_Period();                       //reevaluate the distance fromthe object
                }
        alarm_entered = 0;//when the object is no longer sensed within 15, reset the alarm flag
```

```
                        Startup_Menu();                                                       //go back to the startup menu

                        } else
                        P8->OUT &= ~BIT2;                                                            //else, turn the LED off
}

/////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////
/*
 *          Interrupt Handlers
 */
///////////////////////////////////////////////////
//////////////////////////////////////////////////
void Enable_Interrupts(void)
{
            NVIC_EnableIRQ(PORT1_IRQn);     //enables interrupt register for Port 1 inputs
            NVIC_EnableIRQ(PORT3_IRQn);     //enables interrupt register for Port 3 inputs
            NVIC_EnableIRQ(PORT5_IRQn);     //enables interrupt register for Port 5 input
            NVIC_EnableIRQ(TA3_0_IRQn); //Enable TimerA3 interrupt for half second interval
  //NVIC_EnableIRQ(TA1_0_IRQn);   //for 1ms interval

  __enable_irq();             //enable global interrupts
}
void PORT5_IRQHandler(void)
{
            if (P5->IFG & BIT2)             //Checks for P5.5 flag bit
  {
                        Revolution_count++;    //increments global variable for revolutions of a wheel
                        //Clears interrupt flag
  }

            P5->IFG = 0;
            // P5->IFG &= ~BIT0;               //Clears interrupt flag
}
void TA0_0_IRQHandler(void)
{
//TOGGLE_P20;
  TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
//clear flag bit
}

void TA1_0_IRQHandler(void)
{
//TOGGLE_P20;
            timerA1_0_interruptFlag = 1;                                        //sets flag every half second
  TIMER_A1->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
  //clear flag bit
}

void TA2_0_IRQHandler(void)
{
//TOGGLE_P20;
  TIMER_A2->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
//clear flag bit
}

void TA3_0_IRQHandler(void)
{
  timerA3_0_interruptFlag = 1;          //set gloabl timer interrupt

            if (leftBlinker_Switch_held == 1) {    //sets blinker flags separately for evaluation separate from other calculations
                        TOGGLE_Left_Blinker_flag = 1;
```

```
          }

          else if (rightBlinker_Switch_held == 1) {
                    TOGGLE_Right_Blinker_flag = 1;
          }

          else if ((rightBlinker_Switch_held == 0)
                              && (leftBlinker_Switch_held == 0))
          {
                    BLINKERS_OFF_flag = 1;
          }

  TIMER_A3->CCTL[0] &= ~BIT0;
}

void T32_INT1_IRQHandler()
{
          Timer32_interruptFlag = 1;                                  //sets T32 flag each second
          TIMER32_1->INTCLR = 1;                                                //clear the internal register flag
}

void PORT3_IRQHandler(void)
{
          if ((P3->IFG & BIT5 ) >> 5)                       //checks P3.5
  {
    Open_Door_Btn_interruptFlag = 1;
  }

          if ((P3->IFG & BIT2 ) >> 2)                                                     //checks P3.2
                              {
                    NewRTC_Info_Btn_interruptFlag = 1;
          }

          if ((P3->IFG & BIT3 ) >> 3) {                                         //checks 3.3
                    WatchdogReset_Btn_interruptFlag = 1;
          }
          //P3->IFG &= ~BIT5;
          P3->IFG = 0;
}

void PORT1_IRQHandler(void)
{
  if (CLK == 0)                     //if clock was 1 prior to the interrupt
  {
                    cursor_moved = 0;
    DT = (P1->IN & BIT6 ) >> 6; //sets current value of encoder lines to these variables
    CLK = (P1->IN & BIT7 ) >> 7;       //shifted so I can compare them later

    if ((CLK == 1) && (DT == 0)) //if lines were not equal and the clk was zero before, turn is CW
    {
      CW_turns++;
    }
    else if (CLK == 1 && DT == 1)              //else, CCW
    {
      CCW_turns++;
    }
  }

  else if (CLK == 1)                 //if clock was 1 prior to the interrupt
  {
                    cursor_moved = 0;
    DT = (P1->IN & BIT6 ) >> 6; //sets current value of encoder lines to these variables
    CLK = (P1->IN & BIT7 ) >> 7;       //shifted so I can compare them later
    if (CLK == 0 && DT == 1)    //if clk was 1 prior and they are not equal
      CW_turns++;

    else if ((CLK == 0) && (DT == 0))         //else, CCW
      CCW_turns++;
  }

  if (CLK)          //toggle interrupt select because they only change once
    P1->IES |= BIT7;

  else
    P1->IES &= ~BIT7;

  if (((P1->IFG & BIT5 ) >> 5) == 1) //if interrupt is for an encoder button press
    Encoder_Btn_interruptFlag = 1;

  P1->IFG = 0;                //reset interrupt register flags on all pins
}

///////////////////////////////////////////////////
//////////////////////////////////////////////////
void main987465(void) //main for testing proximity            | works
{
  Watchdog_halt();

  Clock_Init48MHz();
  AClock_Divide4();
  TIMERA3_0_HalfSecondInterval_init();
  LED_closeProximity();
  Enable_Interrupts();
          ST7735_InitR(INITR_REDTAB);
          TimerA0_1_PWM_init();
          ADC_pin_init();
          ADC_StartConversion_WithInterrupts();           //Photocell        | 6.1 | ADC
          Enable_Interrupts();
  HCSR04_init();
  Speaker_12MHz(0, 0);

  Startup_Menu();
  while (1)
  {
                    ADC_Reading();
                    change_TIMERA0_1_dutyCycle();

                    if (alarm_count == 5) {
                              alarm_count = 0;
                    }

    if (timerA3_0_interruptFlag == 1)
    {
      measure_Period();

                              Read_RTC_Print_Time_toLCD();
                              Read_RTC_Temperature_Print_toLCD();

                              if (object_DistanceFromSensor < 15) {

                                        while (object_DistanceFromSensor < 15) {

                                                  if (alarm_entered == 0) {
                                                  Read_RTC_Store_Alarm_EEPROM(alarm_count);
                                                  alarm_count++;
                                                            alarm_entered = 1;
                                                  }
```

```
                                                    P8->OUT |= BIT2;

                                                    displayParkingWarning();
                                                    Bells(5);
                                                    Output_Clear();

                                                    measure_Period();
                                          }

                                          alarm_entered = 0;
                                          Startup_Menu();

                               } else
        P8->OUT &= ~BIT2;

      timerA3_0_interruptFlag = 0;
    }
  }

}

void main58394756()    //Servo_Test
{
          int rotation = 0;
          Clock_Init48MHz();
          AClock_Divide4();

          TIMERA3_0_HalfSecondInterval_init();
          Tachometer_Init();
          Enable_Interrupts();
          rotation = 0;
          while (1) {
                     if (timerA3_0_interruptFlag == 1) {
                               rotation += 50;
                               TachometerSpeed(rotation);
                               SysTick_delay_ms(10);
                               TIMER_A2->CCR[2] = 0;

                               timerA3_0_interruptFlag = 0;
                     }
          }
}

void main98475(void)   //menus_main | showing time and date and temperature
{
  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

  Clock_Init48MHz();
          AClock_Divide4();

          TIMERA3_0_HalfSecondInterval_init();

  ST7735_InitR(INITR_REDTAB);

  TimerA0_1_PWM_init();

          ADC_pin_init();
          ADC_StartConversion_WithInterrupts();          //Photocell        | 6.1 | ADC

          Enable_Interrupts();

  Startup_Menu();
  while (1)
  {
                    ADC_Reading();
                    change_TIMERA0_1_dutyCycle();

                    if (timerA3_0_interruptFlag == 1) {  //do this if startup menu is running

                              Read_RTC_Print_Time_toLCD();
                              Read_RTC_Temperature_Print_toLCD();

                              timerA3_0_interruptFlag = 0;
                    }

  }
}

void main09965()     //main for testing door Ajar  | works
{
  int i = 0;

  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

  Clock_Init48MHz();

  SysTickInit_NoInterrupts();

  openDoor_Btn_init();
  SPI_init();

          MAX219_SPI_init_Decode();

  Enable_Interrupts();

  while (1)
  {


    Open_Door_Btn_heldDown = (!(P3IN & 0x20));

    for (i = 0; i < 15; i++)
    {
      Debounce_DoorOpen_Btn();
    }

                    if (Open_Door_Btn_interruptFlag == 1)              //interrupt occurs
                                        {
                              if (Open_Door_Btn_pressed == 1)      //confirms debounce successful
                                                  {
                                        DoorAjar_Btn_HeldDown();
                                        Open_Door_Btn_pressed = 0;              //reset debounce flag
                                        Open_Door_Btn_interruptFlag = 0;
                              }
                              MAX219_SPI_init_Decode();
                    }

                    else


                    MAX219_SPI_TransferData(0x04, 0x07);        //display 74
                    MAX219_SPI_TransferData(0x04, 0x04);

  }
}
```

```
void main96847(void)      //Blinker testing main | works
{
   Watchdog_halt();

   Clock_Init48MHz();
   AClock_Divide4();

   Blinkers_Toggle_Switch_init();

   Left_Blinker();
   Right_Blinker();

   TIMERA3_0_HalfSecondInterval_init();
   Enable_Interrupts();

   while (1)
   {

      leftBlinker_Switch_held = (!(P3IN & BIT6 ));
      rightBlinker_Switch_held = (!(P3IN & BIT7 ));

      if ((timerA3_0_interruptFlag == 1) && (leftBlinker_Switch_held == 1))
      {
         TOGGLE_Left_Blinker;

         timerA3_0_interruptFlag = 0;
      }

      else if ((timerA3_0_interruptFlag == 1)
            && (rightBlinker_Switch_held == 1))
      {
         TOGGLE_Right_Blinker;

         timerA3_0_interruptFlag = 0;
      }
      else if ((leftBlinker_Switch_held == 0)
            && (rightBlinker_Switch_held == 0))
      {
         BLINKERS_OFF;
      }
   }
}

void main583847() {                                  //digital speed test

                int i;
                Watchdog_halt();

                Clock_Init48MHz();
                AClock_Divide4();

                ST7735_InitR(INITR_REDTAB);
                TimerA0_1_PWM_init();

                SysTickInit_NoInterrupts(); //must be done right before initializing stepper
                Stepper1_Motor();

                Hall_Effect_Out_init();

                Timer32_init();

                openDoor_Btn_init();
                SPI_init();
```

```
MAX219_SPI_init_Decode();

ADC_pin_init();
ADC_StartConversion_WithInterrupts();       //Photocell     | 6.1 | ADC

Enable_Interrupts();

while (1) {
    if (alarm_count == 5) {
        alarm_count = 0;
    }

    ADC_Reading();
    change_TIMERA0_1_dutyCycle();

    Open_Door_Btn_heldDown = (!(P3IN & 0x20));

    for (i = 0; i < 15; i++) {
        Debounce_DoorOpen_Btn();
    }

    DoorAjar_Btn_HeldDown();

    if (Timer32_interruptFlag == 1) {
        RPMs = measure_RPMs(Revolution_count);

        Timer32_interruptFlag = 0;
        Revolution_count = 0;

        Current_Speed_mph = measure_Speed(RPMs,
        CIRCUMFERENCE_6ftDiameter);

        speed_Dif = (Current_Speed_mph - lastSpeed);

        if (RPMs == 0) {
            if (RPMs_areZero == 0) {
                Rotate_Stepper1_Backward_s_Steps(120);
                RPMs = 0;
                RPMs_areZero = 1;
            }

        }

        else if (RPMs > 2) {
            RPMs_areZero = 0;
            if (firstTime_done == 0) {
                firstTime_done = 1;

                Rotate_Stepper1_Forward_s_Steps(Current_Speed_mph);
                RPMs = 0;

                if (Current_Speed_mph > 85) {
                    Read_RTC_Store_Alarm_EEPROM(alarm_count);
                    alarm_count++;
                }
            } else {
                if (speed_Dif == 0) {
                    RPMs = 0;
                } else if (speed_Dif > 0) {
                    Rotate_Stepper1_Forward_s_Steps(speed_Dif);
                    RPMs = 0;

                    if (speed_Dif > 85) {
                        Read_RTC_Store_Alarm_EEPROM(alarm_count);
```

```
                            alarm_count++;
                        }
                    } else if (speed_Dif < 0) {
                        Rotate_Stepper1_Backward_s_Steps(abs(speed_Dif));
                        RPMs = 0;
                    }
                }
            }

            lastSpeed = Current_Speed_mph;

        }
    }
}

void main94857() {    //watchdog test main
    Watchdog_halt();

    Clock_Init48MHz();
    AClock_Divide4();

    int i;

    while (1) {
        for (i = 0; i < 15; i++) {
            //debounce warchdog button
        }
    }
}

void main723454() //Motor_main | works
{
  Watchdog_halt();

  Clock_Init48MHz();
  AClock_Divide4();

    //ST7735_InitR(INITR_REDTAB);
    //TimerA0_1_PWM_init();

  SysTickInit_NoInterrupts(); //must be done right before initializing stepper
  Stepper1_Motor();

  Hall_Effect_Out_init();

    //MAX219_SPI_init_Decode();

  Timer32_init();

    //ADC_pin_init();
    //ADC_StartConversion_WithInterrupts();        //Photocell        | 6.1 | ADC

  Enable_Interrupts();

  while (1)
  {
        if (alarm_count == 5) {
            alarm_count = 0;
        }

        //ADC_Reading();
        //change_TIMERA0_1_dutyCycle();
```

```
if (Timer32_interruptFlag == 1)
{
  RPMs = measure_RPMs(Revolution_count);

  Timer32_interruptFlag = 0;
  Revolution_count = 0;

  Current_Speed_mph = measure_Speed(RPMs,
  CIRCUMFERENCE_6ftDiameter);

                            MAX219_SPI_TransferData(0x04, Current_Speed_mph / 10);  //display 74
                            MAX219_SPI_TransferData(0x04, Current_Speed_mph % 10);
  speed_Dif = (Current_Speed_mph - lastSpeed);

  if (RPMs == 0)
  {
    if (RPMs_areZero == 0)
    {
      Rotate_Stepper1_Backward_s_Steps(120);
      RPMs = 0;
      RPMs_areZero = 1;
    }

  }

  else if (RPMs > 2)
  {
    RPMs_areZero = 0;
    if (firstTime_done == 0)
    {
      firstTime_done = 1;

      Rotate_Stepper1_Forward_s_Steps(Current_Speed_mph);

      RPMs = 0;

                                                  if (Current_Speed_mph > 85) {
                                                             Read_RTC_Store_Alarm_EEPROM(alarm_count);
                                                             alarm_count++;
                                                  }
    }
    else
    {
      if (speed_Dif == 0)
      {
        RPMs = 0;
      }
      else if (speed_Dif > 0)
      {
        Rotate_Stepper1_Forward_s_Steps(speed_Dif);
        RPMs = 0;

                                                             if (speed_Dif > 85) {
                                                                        Read_RTC_Store_Alarm_EEPROM(alarm_count);
                                                                        alarm_count++;
                                                             }
      }
      else if (speed_Dif < 0)
      {
                                                             Rotate_Stepper1_Backward_s_Steps(abs(speed_Dif));

        RPMs = 0;
      }
    }
  }
```

```
      lastSpeed = Current_Speed_mph;

    }
  }
}

////////////////////////////////////////////
//component integration main
void main86938() {

          int i;
          Clock_Init48MHz();
          AClock_Divide4();

          TIMERA3_0_HalfSecondInterval_init();

          ST7735_InitR(INITR_REDTAB);

          TimerA0_1_PWM_init();

          ADC_pin_init();
          ADC_StartConversion_WithInterrupts();      //Photocell     | 6.1 | ADC

          Encoder_init();

          Blinkers_Toggle_Switch_init();

          Left_Blinker();
          Right_Blinker();

          openDoor_Btn_init();
          NewRTC_Info_Btn();

          SPI_init();

          MAX219_SPI_init_Decode();
          Print_Zeros_MAX219();

          Enable_Interrupts();

          Startup_Menu();

          while (1)
          {
                    ADC_Reading();
                    change_TIMERA0_1_dutyCycle();

                    leftBlinker_Switch_held = (!(P3IN & BIT6 ));
                    rightBlinker_Switch_held = (!(P3IN & BIT7 ));

                    Open_Door_Btn_heldDown = (!(P3IN & 0x20));

                    for (i = 0; i < 15; i++) {
                              Debounce_DoorOpen_Btn();
                              Debounce_EncoderButton(); //global debounce flag for encoder button;
                              Debounce_NewRTC_Info_Btn();

                    }

                    if (Open_Door_Btn_interruptFlag == 1)        //interrupt occurs
                                        {
                              if (Open_Door_Btn_pressed == 1)    //confirms debounce successful
                                                  {
```

```
        DoorAjar_Btn_HeldDown();
        Open_Door_Btn_pressed = 0;          //reset debounce flag
        Open_Door_Btn_interruptFlag = 0;
    }
    MAX219_SPI_init_Decode();
}

if (Encoder_Btn_interruptFlag == 1) {
    if (Encoder_Btn_pressed == 1) {
        MENU();
    }
}

if (NewRTC_Info_Btn_interruptFlag == 1) {
    if (NewRTC_Info_Btn_pressed == 1) {

        if ((menu == 2 || menu == 3)) {
            Enter_Date_Time_Btn_count++;

            NewRTC_Info_Btn_interruptFlag = 0;
            NewRTC_Info_Btn_pressed = 0;
            reset_Encoder_Counts();
        }
    }


if (TOGGLE_Left_Blinker_flag == 1) {
    TOGGLE_Left_Blinker;
    TOGGLE_Left_Blinker_flag = 0;
}

else if (TOGGLE_Right_Blinker_flag == 1) {
    TOGGLE_Right_Blinker;
    TOGGLE_Right_Blinker_flag = 0;
}

else if (BLINKERS_OFF_flag == 1) {
    BLINKERS_OFF;
    BLINKERS_OFF_flag = 0;
}
switch (menu) {

case 0: {

    if (timerA3_0_interruptFlag == 1) {  //do this if startup menu is running

        Read_RTC_Print_Time_toLCD();
        Read_RTC_Temperature_Print_toLCD();

        timerA3_0_interruptFlag = 0;
    }

    break;
}
case 1:
{
    control_Cursor();
    break;
}
case 2: {
    control_Cursor();
    break;
}
case 3: {
```

```
            control_Cursor();
            break;
        }
        case 4: {
            break;
        }

        default:
        {
            break;
        }
    }
    }
}
```

Clocks.h

```
/*
 * Clocks.h
 *
 *  Created on: Nov 25, 2020
 *      Author: 7jorchay
 */

#ifndef CLOCKS_H_
#define CLOCKS_H_


void Clocks_main();     //main for testing
////////////////////////////////////////////////////////
/*
 *          SYSTEM CLOCK CHANGES
 */
////////////////////////////////////////////////////////
void Clock_Init48MHz(void);

void AClock_Divide4(void);

////////////////////////////////////////////////////////
/*
 *          WATCHDOG
 */
////////////////////////////////////////////////////////
void Watchdog_init(void);

void Watchdog_resetCount(void);

void Watchdog_halt(void);

////////////////////////////////////////////////////////
/*
 *          SYSTICK
 */
////////////////////////////////////////////////////////
/*
void SysTick_Init_NoInterrupts();
void SysTick_delay_ms(uint16_t delay);
void SysTick_delay_us(int delay);
 */

////////////////////////////////////////////////////////
/*
 *          TIMER32
 */
////////////////////////////////////////////////////////
void Timer32_init(void);
```

```
#endif /* CLOCKS_H_ */
```

Clocks.c

```
/*
 * Clocks.c
 *
 *  Created on: Nov 25, 2020
 *      Author: 7jorchay
 */
#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "Clocks.h"

//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
/*
 *              MCLK | SMCLK
 */
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
void Clock_Init48MHz(void)
{
  // Configure Flash wait-state to 1 for both banks 0 & 1
  FLCTL->BANK0_RDCTL = (FLCTL->BANK0_RDCTL & ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) |
  FLCTL_BANK0_RDCTL_WAIT_1;
  FLCTL->BANK1_RDCTL = (FLCTL->BANK0_RDCTL & ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) |
  FLCTL_BANK1_RDCTL_WAIT_1;

  //Configure HFXT to use 48MHz crystal, source to MCLK & HSMCLK*
  PJ->SEL0 |= BIT2 | BIT3;           // Configure PJ.2/3 for HFXT function
  PJ->SEL1 &= ~(BIT2 | BIT3 );
  CS->KEY = CS_KEY_VAL;              // Unlock CS module for register access
  CS->CTL2 |= CS_CTL2_HFXT_EN | CS_CTL2_HFXTFREQ_6 | CS_CTL2_HFXTDRIVE;
  while (CS->IFG & CS_IFG_HFXTIFG)
    CS->CLRIFG |= CS_CLRIFG_CLR_HFXTIFG;

  /* Select MCLK & HSMCLK = HFXT, no divider */
  CS->CTL1 = CS->CTL1 & ~(CS_CTL1_SELM_MASK |
  CS_CTL1_DIVM_MASK |
  CS_CTL1_SELS_MASK |
  CS_CTL1_DIVHS_MASK) |
  CS_CTL1_SELM__HFXTCLK |
  CS_CTL1_SELS__HFXTCLK;

  CS->CTL1 = CS->CTL1 | CS_CTL1_DIVS_2; // change the SMCLK clock speed to 12 MHz.

  CS->KEY = 0;                  // Lock CS module from unintended accesses
}

//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
/*
 *              Watchdog
 */
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//initializes watchdog timer for watchdog mode, sending program into infinite while loop.
//after 4 seconds, the timer will time out and reset the program
void Watchdog_init()
{
  WDT_A->CTL = WDT_A_CTL_PW | 0x0024;        //00100100
                                //1s at 32.768 KHz
```

```
                                    //ACLK, watchdog mode
        while (1) {
                ;
        }
}

//function can reset the watchdog count value before program resets
void Watchdog_resetCount()
{
   CS->KEY = CS_KEY_VAL;                          //password needed

   WDT_A->CTL = WDT_A_CTL_PW | WDTCNTCL | 0x0024;      //resets watchdog count

   CS->KEY = 0;
}

//halts the watchdog timer
void Watchdog_halt()
{
   WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;    // stop watchdog timer
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/*
 *              ACLK
 */
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//didives the internal ACLK by 4 to reach longer intervals at 48MHz
void AClock_Divide4()
{
   CS->KEY = CS_KEY_VAL;                      //open register with password
   CS->CTL1 |= BIT(25) | BIT9;                //bit25 for ACLK / 4
                                     //bit9 for REFOCLK
   CS->KEY = 0;
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/*
 *              TIMER32
 */
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//initializes TIMER32 for a one second interval with interrupts
void Timer32_init(void)
{
        TIMER32_1->LOAD = (48000000) - 1;      //change this, is one second now
   TIMER32_1->CONTROL = 0b11100010;        //source to a different clock?
        NVIC_EnableIRQ(T32_INT1_IRQn);//might want to move this to enable interrupts
}
```

EEPROM.c

```
/*
 * EEPROM.c
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */


#include <stdio.h>
#include <stdint.h>
```

```
#include "msp.h"
#include "EEPROM.h"
#include "I2C_ProjectFunctions.h"
#include "ST7735.h"
#include "Project.h"

volatile int Alarm_position = 0;

//not used
void store_Date_and_Time_EEPROM_withType(unsigned char* Time_Date, int count,
                    int type)
{
          if (type == 0)                              //if type is zero it is a collision
                              {
                    ;
          }

          if (type == 1) {
                    count += 5;
          }

  uint8_t start_memAddr = (count * 8); //move to next "drawer", addresses are 7 bytes long so multiply by 8 to jump to next
                    //will write to 0, 8, 16, 24, 32
  I2C1_init();

  I2C1_burstWrite(slaveAddrA0, start_memAddr, 7, Time_Date);
}

//reads the RTC time and stores it in the EEPROM
//address stored determined by count
void Read_RTC_Store_Alarm_EEPROM(int count) {
          unsigned char Time_Date_toSend_EEPROM[7];

          I2C1_init();

          I2C1_burstRead(slaveAddrRTC, 0, 7, Time_Date_toSend_EEPROM); //read all 7 values from RTC

          store_Date_and_Time_EEPROM(Time_Date_toSend_EEPROM, count); //send to store in EEPROM, button count decides which address it
writes to
}

void store_Date_and_Time_EEPROM(unsigned char* Time_Date, int Alarm_count) {


          uint8_t start_memAddr = (Alarm_count * 8); //move to next "drawer", addresses are 7 bytes long so multiply by 8 to jump to next
          //will write to 0, 8, 16, 24, 32
          I2C1_init();

          I2C1_burstWrite(slaveAddrA0, start_memAddr, 7, Time_Date);
}
////////////////////////////////////////////////////////////////////////////////

//not used
void read_Date_and_Time_EEPROM_withType(int count, int type)     //edit
{
  unsigned char Time_Date_toRead[7];     //create array to store read values

  uint8_t start_memAddr = (count * 8);

  I2C1_init();

  I2C1_burstRead(slaveAddrA0, start_memAddr, 7, Time_Date_toRead); //put watch expression here

  print_Alarm_LCD(Time_Date_toRead, count);
```

```
}

//reads the times that alarms have been stored into the EEPROM
//reads 5 alarms
void read_Date_and_Time_EEPROM(int Alarm_count) {
            unsigned char Time_Date_toRead[7];      //create array to store read values

            uint8_t start_memAddr = (Alarm_count * 8);

            I2C1_init();

            I2C1_burstRead(slaveAddrA0, start_memAddr, 7, Time_Date_toRead); //put watch expression here

            print_Alarm_LCD(Time_Date_toRead, Alarm_count);
}

//prints alarms read from the EEPROM to the LCD in readable decimal numbers and format
void print_Alarm_LCD(unsigned char* Time_Date_Array, int Alarm_count)
{
  //uint8_t year = Time_Date_Array[6];     local variable to hold the year byte
  //uint8_t month = Time_Date_Array[5];    local variable to hold the month byte
  //uint8_t date = Time_Date_Array[4];     local variable to hold the date byte
  //uint8_t day = Time_Date_Array[3];      local variable to hold the day byte
  //hours = Time_Date_Array[2];
  //minutes = Time_Date_Array[1];
  //seconds = Time_Date_Array[0];

            uint16_t start_PositionY;
            uint16_t start_PositionX;

            if (Alarm_count == 0) {
                        start_PositionY = 30;
                        start_PositionX = 20;
            }
            if (Alarm_count == 1) {
                        start_PositionY = 42;
                        start_PositionX = 20;
            }
            if (Alarm_count == 2) {
                        start_PositionY = 54;
                        start_PositionX = 20;
            }
            if (Alarm_count == 3) {
                        start_PositionY = 66;
                        start_PositionX = 20;
            }
            if (Alarm_count == 4) {
                        start_PositionY = 78;
                        start_PositionX = 20;
            }

  uint16_t text_Color = 0xF81F;          //Purple
  uint16_t bg_Color = 0;               //black

  uint8_t num_toPrint_ones;             //variable printed to LCD ones place
  uint8_t num_toPrint_tens;             //variable printed to LCD tens place

  int i = 0;

  unsigned char Date_Array[5] = { Time_Date_Array[5], Time_Date_Array[4],
                         Time_Date_Array[2], Time_Date_Array[1],
                         Time_Date_Array[0] }; //fills array with desired values for lab

  for (i = 0; i < 5; i++)
  {
                     num_toPrint_tens = ((Date_Array[i] & 0xF0) >> 4) + '0'; //convert to decimal and print character value

    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_tens,
            text_Color, bg_Color, 1);

    start_PositionX += 6;

    num_toPrint_ones = (Date_Array[i] & 0x0F) + '0';

    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_ones,
            text_Color, bg_Color, 1);

    start_PositionX += 6;

                     if (i != 4) //print a semicolon after each number print until the last one
    {
      ST7735_DrawChar(start_PositionX, start_PositionY, ':', text_Color,
              bg_Color, 1);
      start_PositionX += 6;
    }
  }

}
```

EEPROM.h

```
/*
 * EEPROM.h
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#ifndef EEPROM_H_
#define EEPROM_H_

#define slaveAddrA0 0b1010001

extern volatile int alarm_number;

void store_Date_and_Time_EEPROM(unsigned char* Time_Date, int count); //stores  character array to EEPROM using I2C

void read_Date_and_Time_EEPROM(int count); //read charater array from EEPROM using I2C

void print_Alarm_LCD(unsigned char* Time_Date_Array, int Alarm_count); //prints the Alarm read from the EEPROM in the labs desired format
void Read_RTC_Store_Alarm_EEPROM(int count);

#endif /* EEPROM_H_ */
```

EncoderFunctions.c

```
/*
 * EncoderFunctions.c
 *
 *  Created on: Nov 26, 2020
 *      Author: 7jorchay
 */



#include "msp.h"
#include <stdio.h>
#include <math.h>
```

```
#include "EncoderFunctions.h"
#include "ST7735.h"
#include "menus.h"
#include "Project.h"
#include "RTC_ProjectFunctions.h"

volatile int Encoder_Btn_pressed = 0;
volatile int Encoder_Btn_interruptFlag = 0;        //interrupt flag variables
volatile int CCW_turns = 0;                        //variable for CCW pulses
volatile int CW_turns = 0;                         //variable for CW pulses
volatile int total_turns = 100; //number of turns in either direction and magnitude of difference
volatile int direction = 3;                  //if total is CW, = 0, if CCW, = 1

volatile int DT = 1;                              //DT variable, starts high
volatile int CLK = 1;                             //CLK variable, starts high

volatile int cursor_moved = 0;//global variable for whether the cursor has responded to
//a change in encoder turns so that it does not happen repeatedly

//initialize components of the encoder
void Encoder_init(void)
{
   Encoder_Btn_init();
   Encoder_CLK_init();
   Encoder_DT_init();
}

//initialize the encoder switch as an input, same as a button
void Encoder_Btn_init(void)
{
   P1->SEL0 &= ~BIT5;         //P1.5 set to GPIO
   P1->SEL1 &= ~BIT5;
   P1->DIR &= ~BIT5;          //P1.5 direction set to input
   P1->REN |= BIT5;           //activating the internal resistor
   P1->OUT |= BIT5;           //declaring the internal resistor as pull-up
   P1->IES |= BIT5;          //interrupt edge select on falling edge
   P1->IE |= BIT5;           //Interrupt enable
   P1->IFG &= ~BIT5;         //Clears the flag bit

}

//initializes encoder CLK line as input like a button
//handled by comparing the line to the DT line in the Port 1 handler
void Encoder_CLK_init(void)
{
   P1->SEL0 &= ~BIT7;         //P1.7 set to GPIO
   P1->SEL1 &= ~BIT7;
   P1->DIR &= ~BIT7;          //P1.7 direction set to input
   P1->REN |= BIT7;           //activating the internal resistor
   P1->OUT |= BIT7;           //declaring the internal resistor as pull-up
   P1->IES |= BIT7;          //interrupt edge select on falling edge
   P1->IE |= BIT7;           //Interrupt enable
   P1->IFG &= ~BIT7;         //Clears the flag bit
}

//initializes the DT line, no pull up resistor
//handled by comparing to CLK line
void Encoder_DT_init(void)      //data line whether the knob was last turned
{
   P1->SEL0 &= ~BIT6;         //P1.6 set to GPIO
   P1->SEL1 &= ~BIT6;
   P1->DIR &= ~BIT6;          //P1.6 direction set to input
   P1->REN |= BIT6;           //activating the internal resistor
   P1->OUT |= BIT6;          //declaring the internal resistor as pull-up
```

```
}

//debounce for the encoder button
int Debounce_EncoderButton()
{
   static uint16_t State_1 = 0;                 //Current debounce status

   State_1 = (State_1 << 1) | (P1IN & 0x20) >> 5 | 0xf800; //checks pin 1.5, don

   if (State_1 == 0xfc00)
   {
      Encoder_Btn_pressed = 1;
      return 1;                                //after 0 level is
                                          //stable for 10 consecutive calls
   }
   else
      return 0;
}

//resets all encoder counts: turns, and cursor moved flag
void reset_Encoder_Counts() {

        CW_turns = 0;                   //reset variable values
        CCW_turns = 0;
        total_turns = 0;
        cursor_moved = 0;
}

//this function holds all the conditions for what happens when the encoder knob is turned
//uses global variables for which menu is displayed and how many turns to determine which set of words is highlighted as an option
//if the encoder button is pressed while an option is highlight, the menu will be opened accordingly
//works by reprinting the menu, then reprinting the desired line with a black background in the same coordinates as the original menu
//changing individual digits in time and date are displayed and saved into RTC by using switch cases on total_turns % 10
//the currently highlighted and changing digit are set equal to total turns, and encoder values are reset if it goes out of bound for
//the current condition of the digit being set. The New_RTC_info button will also reset the counts and change the highlighted
//digit to the next one, changing the applicable conditions for encoder turn values
//many conditions for digits are dependent on the digit before. These conditions are nested within menu, then New_RTC_info button values


void control_Cursor(void) {

        char string_toPrint[20];
        uint16_t start_PositionX;
        uint16_t start_PositionY;

        uint16_t text_Color = 0xFFFF;        //white number when changing
        uint16_t bg_Color = 0;                                   //black bg for option being selected

        int i = 0;

        if (cursor_moved == 0) {//only enters when the condition true after turning encoder is not yet executed

                cursor_moved = 1;                                              //set flag variable

                total_turns = abs(CW_turns - CCW_turns);  //magnitude of pulse direction

                if ((total_turns == 0) && (menu == 1)) {                   //highlight Set time

                        Interface_Options_Menu();                                        //reprint the option menu

                start_PositionX = 15;
                start_PositionY = 52;
                        string_toPrint[0] = 'S';//reprint the Set Time option with black background
                string_toPrint[1] = 'e';
                string_toPrint[2] = 't';
                string_toPrint[3] = ' ';
                string_toPrint[4] = 'T';
                string_toPrint[5] = 'i';
                string_toPrint[6] = 'm';
                string_toPrint[7] = 'e';
                string_toPrint[8] = NULL;

                for (i = 0; string_toPrint[i] != NULL; i++) {
                        ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                        ST7735_BLUE, bg_Color, 2);
                        start_PositionX += 12;
                }
        }

                else if ((total_turns == 1) && (menu == 1)) {
                Interface_Options_Menu();

                start_PositionX = 15;
                        start_PositionY = 69;
                string_toPrint[0] = 'S';
                string_toPrint[1] = 'e';
                string_toPrint[2] = 't';
                string_toPrint[3] = ' ';
                string_toPrint[4] = 'D';
                string_toPrint[5] = 'a';
                string_toPrint[6] = 't';
                string_toPrint[7] = 'e';
                string_toPrint[8] = NULL;

                for (i = 0; string_toPrint[i] != NULL; i++) {
                        ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                                                0x04E0, bg_Color, 2);
                        start_PositionX += 12;
                }
        }

                else if ((total_turns == 2) && (menu == 1)) {
                Interface_Options_Menu();

                        start_PositionY = 86;
                start_PositionX = 30;

                string_toPrint[0] = 'A';
                string_toPrint[1] = 'l';
                string_toPrint[2] = 'a';
                string_toPrint[3] = 'r';
                string_toPrint[4] = 'm';
                string_toPrint[5] = 's';
                string_toPrint[6] = NULL;

                for (i = 0; string_toPrint[i] != NULL; i++) {
                        ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                        ST7735_RED, bg_Color, 2);
                        start_PositionX += 12;
                }
        }

                ////////////////////////////////////////////////////
                //
                //Starts conditions for changing time and date
                //
                ////////////////////////////////////////////////////
                else if (menu == 2) {                                               //if menu is 2, changing time
```

```
Set_Time_Menu();

start_PositionX = 20;
start_PositionY = 60;

if (Enter_Date_Time_Btn_count == 0) {// setting tens place os hours 0 - 2

    switch (total_turns % 10) {

    case 0: {

        New_RTC_Time_Hour_tens = 0x00;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Hour_tens + '0',
            text_Color, bg_Color, 2);
        break;
    }

    case 1: {

        New_RTC_Time_Hour_tens = 0x01;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Hour_tens + '0',
            text_Color, bg_Color, 2);
        break;
    }

    case 2: {

        New_RTC_Time_Hour_tens = 0x02;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Hour_tens + '0',
            text_Color, bg_Color, 2);
        break;
    }

    default:
    {
    reset_Encoder_Counts();
        break;
    }

    }
}

else if (Enter_Date_Time_Btn_count == 1) {//setting ones place of hours 0-9 unless tens was 2, then 0 - 3
    start_PositionX = 32;

    if ((New_RTC_Time_Hour_tens == 0)
            || (New_RTC_Time_Hour_tens == 1)) {

        switch (total_turns % 10) {

        case 0: {

            New_RTC_Time_Hour_ones = 0x00;

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Time_Hour_ones + '0', text_Color,
                    bg_Color, 2);
            break;
```

```
}

case 1: {

        New_RTC_Time_Hour_ones = 0x01;
        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}
case 2: {

        New_RTC_Time_Hour_ones = 0x02;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}

case 3: {
        New_RTC_Time_Hour_ones = 0x03;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}

case 4: {

        New_RTC_Time_Hour_ones = 0x04;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}
case 5: {

        New_RTC_Time_Hour_ones = 0x05;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}
case 6: {

        New_RTC_Time_Hour_ones = 0x06;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
}
case 7: {

        New_RTC_Time_Hour_ones = 0x07;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
        break;
        }
        case 8: {

                New_RTC_Time_Hour_ones = 0x08;

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
                break;
        }

        case 9: {

                New_RTC_Time_Hour_ones = 0x09;

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Time_Hour_ones + '0', text_Color,
                                bg_Color, 2);
                break;
        }
        default:
        {
                reset_Encoder_Counts();
                break;
        }

        }
}

if (New_RTC_Time_Hour_tens == 2) {

        switch (total_turns % 10) {

        case 0: {

                New_RTC_Time_Hour_ones = 0x00;

                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Hour_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }

        case 1: {

                New_RTC_Time_Hour_ones = 0x01;
                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Hour_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }
        case 2: {

                New_RTC_Time_Hour_ones = 0x02;

                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Hour_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }

        case 3: {
                New_RTC_Time_Hour_ones = 0x03;
```

```
            ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Hour_ones + '0', text_Color, bg_Color,
                2);
            break;
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }

    }
}
}

else if (Enter_Date_Time_Btn_count == 2)//setting minutes tens 0 - 5
        {
    Time_toSend[2] = (New_RTC_Time_Hour_tens << 4)
            | New_RTC_Time_Hour_ones;

    start_PositionX = 56;

    switch (total_turns % 10) {

    case 0: {

        New_RTC_Time_Minute_tens = 0x00;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }

    case 1: {

        New_RTC_Time_Minute_tens = 0x01;
        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }
    case 2: {

        New_RTC_Time_Minute_tens = 0x02;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }

    case 3: {
        New_RTC_Time_Minute_tens = 0x03;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }
    case 4: {

        New_RTC_Time_Minute_tens = 0x04;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }
    case 5: {

        New_RTC_Time_Minute_tens = 0x05;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_tens + '0', text_Color,
                bg_Color,
                2);
        break;
    }

    default:
    {
        reset_Encoder_Counts();
        break;
    }
    }

}

else if (Enter_Date_Time_Btn_count == 3) {//setting minutes ones 0 - 9
    start_PositionX = 68;

    switch (total_turns % 10) {

    case 0: {

        New_RTC_Time_Minute_ones = 0x00;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_ones + '0', text_Color,
                bg_Color,
                2);
        break;
    }

    case 1: {

        New_RTC_Time_Minute_ones = 0x01;
        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_ones + '0', text_Color,
                bg_Color,
                2);
        break;
    }
    case 2: {

        New_RTC_Time_Minute_ones = 0x02;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Time_Minute_ones + '0', text_Color,
                bg_Color,
                2);
        break;
}

case 3: {
        New_RTC_Time_Minute_ones = 0x03;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}

case 4: {

        New_RTC_Time_Minute_ones = 0x04;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}
case 5: {

        New_RTC_Time_Minute_ones = 0x05;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}
case 6: {

        New_RTC_Time_Minute_ones = 0x06;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}
case 7: {

        New_RTC_Time_Minute_ones = 0x07;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}
case 8: {

        New_RTC_Time_Minute_ones = 0x08;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Time_Minute_ones + '0', text_Color,
                        bg_Color,
                        2);
        break;
}
```

```
                case 9: {

                    New_RTC_Time_Minute_ones = 0x09;

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                            New_RTC_Time_Minute_ones + '0', text_Color,
                                            bg_Color,
                                            2);
                    break;
                }
                default:
                {
                    reset_Encoder_Counts();
                    break;
                }

                }
            }

            else if (Enter_Date_Time_Btn_count == 4) {//here we have all the values loaded into Time_toSend array


                Time_toSend[1] = (New_RTC_Time_Minute_tens << 4)
                                    | New_RTC_Time_Minute_ones;

                Write_New_Time_RTC(Time_toSend);//load data into RTC using I2C functions
                Enter_Date_Time_Btn_count = 0;//reset New_RTC_info counts as its function is complete
                MENU();     //call menu to go back to startup menu, reading the new time

            }
        }

        else if (menu == 3) {                                                   //changing the date

            Set_Date_Menu();                                                        //reprint menu

            start_PositionX = 5;
            start_PositionY = 60;

            if (Enter_Date_Time_Btn_count == 0) {// setting day of the week 1-7

                New_RTC_Date_Day_ones = total_turns + 1;//can't have a 0th day of the week, since the day is printed
using conditions 1-7

    //this digit is therefore the total turns + 1
                switch (total_turns % 10) {

                case 0: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                            New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                                            2);
                    break;
                }

                case 1: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                            New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                                            2);
                    break;
                }

                case 2: {
```

```
        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                2);
        break;
    }

    case 3: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                2);
        break;
    }
    case 4: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                2);
        break;
    }
    case 5: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                2);
        break;
    }

    case 6: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Day_ones + '0', text_Color, bg_Color,
                2);
        break;
    }

    default:
    {
        reset_Encoder_Counts();
        break;
    }

    }
}

else if (Enter_Date_Time_Btn_count == 1) {//setting tens place of month, 0-2

    start_PositionX = 29;

    New_RTC_Date_Month_tens = total_turns;

    switch (total_turns % 10) {

    case 0: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Month_tens + '0', text_Color, bg_Color,
                2);
        break;
    }

    case 1: {
```

```
                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Date_Month_tens + '0', text_Color, bg_Color,
                        2);
                break;
        }
        default:
        {
                reset_Encoder_Counts();
                break;
        }
        }

}

else if (Enter_Date_Time_Btn_count == 2) {// setting ones place of month, dependent on tens place

        start_PositionX = 41;

        if (New_RTC_Date_Month_tens == 0) {

                New_RTC_Date_Month_ones = total_turns + 1;

                switch (total_turns % 10) {

                case 0: {

                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                                2);
                        break;
                }

                case 1: {

                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                                2);
                        break;
                }

                case 2: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                                2);
                        break;
                }

                case 3: {


                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                                2);
                        break;
                }
                case 4: {

                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                                2);
                        break;
                }
                case 5: {
```

```
                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }

        case 6: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }
        case 7: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                        New_RTC_Date_Month_ones + '0', text_Color, bg_Color,
                        2);
                break;
        }

        case 8: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color,
                                bg_Color, 2);
                break;
        }

        default:
        {
                reset_Encoder_Counts();
                break;
        }

        }
}
else if (New_RTC_Date_Month_tens == 1) {

        New_RTC_Date_Month_ones = total_turns;

        switch (total_turns % 10) {

        case 0: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color,
                                bg_Color, 2);
                break;
        }

        case 1: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color,
                                bg_Color, 2);
                break;
        }

        case 2: {

                ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Month_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }

}

else if (Enter_Date_Time_Btn_count == 3)//date tens place, dependant on the month chosen
        {
    start_PositionX = 65;
    New_RTC_Date_Date_tens = total_turns;

    if ((New_RTC_Date_Month_tens == 1)
            || ((New_RTC_Date_Month_ones != 2))
                    && (New_RTC_Date_Month_tens != 1)) {

        switch (total_turns % 10) {

        case 0: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 1: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 2: {
            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 3: {
            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }
    else if ((New_RTC_Date_Month_tens != 1)//if the month was February
            && (New_RTC_Date_Month_ones == 2)) {

        switch (total_turns % 10) {
```

```
        case 0: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 1: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 2: {
            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_tens + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }

}

else if (Enter_Date_Time_Btn_count == 4) {    //ones place of months
    start_PositionX = 77;

    if (New_RTC_Date_Date_tens == 0)
        New_RTC_Date_Date_ones = (total_turns % 10) + 1;
    else
        New_RTC_Date_Date_ones = total_turns % 10;

    if ((New_RTC_Date_Date_tens == 0)
            || (New_RTC_Date_Date_tens == 1)) {

        switch (total_turns % 10) {

        case 0: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 1: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 2: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 3: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 4: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 5: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 6: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 7: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 8: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 9: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }
```

```
else if (New_RTC_Date_Date_tens == 2) {

    if (!((New_RTC_Date_Month_tens == 0)
            && (New_RTC_Date_Month_ones == 2))) {

        switch (total_turns % 10) {

        case 0: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 1: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 2: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 3: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 4: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 5: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 6: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 7: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 8: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 9: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        default:
        {
            New_RTC_Date_Date_ones = 0;
            reset_Encoder_Counts();
            break;
        }
        }
    }
    else if ((New_RTC_Date_Month_tens == 0)
            && (New_RTC_Date_Month_ones == 2)) {
        switch (total_turns % 10) {

        case 0: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 1: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 2: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 3: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 4: {
            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 5: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        case 6: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 7: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }
        case 8: {

            ST7735_DrawChar(start_PositionX, start_PositionY,
                    New_RTC_Date_Date_ones + '0', text_Color,
                    bg_Color, 2);
            break;
        }

        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }
}

else if (New_RTC_Date_Date_tens == 3) {


    if ((New_RTC_Date_Month_tens == 0)
        && ((New_RTC_Date_Month_ones == 1)
                || (New_RTC_Date_Month_ones == 3)
                || (New_RTC_Date_Month_ones == 5)
                || (New_RTC_Date_Month_ones == 7)
                || (New_RTC_Date_Month_ones == 8))) {//for date ones
place for 31 day months

    switch (total_turns % 10) {

    case 0: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Date_ones + '0', text_Color,
                bg_Color, 2);
        break;
    }
```

```
                    case 1: {

                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Date_ones + '0', text_Color,
                                bg_Color, 2);
                        break;
                    }
                        default:
                    {
                        reset_Encoder_Counts();
                            break;
                    }
                        }
                    } else if ((New_RTC_Date_Month_tens == 1)
                            && ((New_RTC_Date_Month_ones == 0)
                                    || (New_RTC_Date_Month_ones == 2))) {//31
day months

                        switch (total_turns % 10) {

                        case 0: {

                            ST7735_DrawChar(start_PositionX, start_PositionY,
                                    New_RTC_Date_Date_ones + '0', text_Color,
                                    bg_Color, 2);
                            break;
                        }

                        case 1: {

                            ST7735_DrawChar(start_PositionX, start_PositionY,
                                    New_RTC_Date_Date_ones + '0', text_Color,
                                    bg_Color, 2);
                            break;
                        }

                        default:
                        {
                            reset_Encoder_Counts();
                            break;
                        }
                        }
                    }

                }

                else if ((New_RTC_Date_Month_tens == 1)
                        && ((New_RTC_Date_Month_ones == 1)
                                && (New_RTC_Date_Date_tens != 3))) {


                    switch (total_turns % 10) {

                    case 0: {

                        ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Date_ones + '0', text_Color,
                                bg_Color, 2);
                        break;
                    }

                    case 1: {
```

```
                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}
case 2: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}

case 3: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}

case 4: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}
case 5: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}

case 6: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}
case 7: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}
case 8: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
}

case 9: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Date_ones + '0', text_Color,
                                        bg_Color, 2);
                    break;
```

```
        }
        default:
        {
            reset_Encoder_Counts();
            break;
        }
        }
    }
    else if ((New_RTC_Date_Month_tens == 1)
            && (New_RTC_Date_Month_ones == 1)) {        //30 day months

        New_RTC_Date_Date_ones = 0;
        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Date_ones + '0', text_Color, bg_Color,
                2);
        reset_Encoder_Counts();
    }
    else {
        New_RTC_Date_Date_ones = 0;

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Date_ones + '0', text_Color, bg_Color,
                2);
        reset_Encoder_Counts();
    }

}

else if (Enter_Date_Time_Btn_count == 5) {

    start_PositionX = 99;

    New_RTC_Date_Year_tens = total_turns % 10;

    switch (total_turns % 10) {

    case 0: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }

    case 1: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    case 2: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }

    case 3: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }

    case 4: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    case 5: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    case 6: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    case 7: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    case 8: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }

    case 9: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                New_RTC_Date_Year_tens + '0', text_Color,
                bg_Color, 2);
        break;
    }
    default:
    {
        reset_Encoder_Counts();
        break;
    }
    }
}

else if (Enter_Date_Time_Btn_count == 6) {//setting the year ones place. can be 0-9
    start_PositionX = 111;

    New_RTC_Date_Year_ones = total_turns % 10;

    switch (total_turns % 10) {

    case 0: {

        ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }

    case 1: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
    case 2: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }

    case 3: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }

    case 4: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
    case 5: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
    case 6: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
    case 7: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
    case 8: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                        New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                        2);
                    break;
    }
```

```
                case 9: {

                    ST7735_DrawChar(start_PositionX, start_PositionY,
                                New_RTC_Date_Year_ones + '0', text_Color, bg_Color,
                                2);
                    break;
                }
                default:
                {
                    reset_Encoder_Counts();
                    break;
                }
                }
            }

            else if (Enter_Date_Time_Btn_count == 7) {//we now have all the data for the date


                Date_toSend[0] = New_RTC_Date_Day_ones;   //setting the array to data received


                Date_toSend[1] = (New_RTC_Date_Date_tens << 4)//creating a byte of two bcd numbers to make a 2 digit
number readable to RTC
                        | New_RTC_Date_Date_ones;


                Date_toSend[2] = (New_RTC_Date_Month_tens << 4)
                        | New_RTC_Date_Month_ones;



                Date_toSend[3] = (New_RTC_Date_Year_tens << 4)
                        | New_RTC_Date_Year_ones;

                Write_New_Date_RTC(Date_toSend);//write the date to the correct positions in RTC
                Enter_Date_Time_Btn_count = 0;//reset button count after function is complete
                MENU();                                             //call menu to go back to startup

            }

        }

    else
        reset_Encoder_Counts();
    }

}
```

EncoderFunctions.h

```
/*
 * EncoderFunctions.h
 *
 *  Created on: Nov 26, 2020
 *      Author: 7jorchay
 */

#ifndef ENCODERFUNCTIONS_H_
#define ENCODERFUNCTIONS_H_

extern volatile int Encoder_Btn_pressed;
extern volatile int Encoder_Btn_interruptFlag;

extern volatile int CCW_turns;                  //variable for CCW pulses
extern volatile int CW_turns;                        //variable for CW pulses
extern volatile int total_turns; //number of turns in either direction and magnitude of difference
extern volatile int direction;              //if total is CW, = 0, if CCW, = 1

extern volatile int DT;                          //DT variable, starts high
extern volatile int CLK;                         //CLK variable, starts high

extern volatile int Encoder_Btn_count;
extern volatile int cursor_moved;
void Encoder_init(void);

void Encoder_Btn_init(void);
void Encoder_CLK_init(void);
void Encoder_DT_init(void);


void reset_Encoder_Counts(void);

void control_Cursor(void);

int Debounce_EncoderButton(void);

void reset_Encoder_Counts(void);

void PORT1_IRQHandler(void);


#endif /* ENCODERFUNCTIONS_H_ */
```

Hall_Effect.c

```
/*
 * Hall_Effect.c
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */


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

volatile int Current_Speed_mph = 0;
volatile int RPMs = 0;
volatile int Revolution_count = 0;
volatile int speed_Dif = 0;
volatile int lastSpeed = 0;
volatile int firstTime_done = 0;
volatile int RPMs_areZero = 0;

volatile int last_SpeedingTicket_Speed = 0;

//initilaize hall effect output pin as input with interrupt enable
void Hall_Effect_Out_init(void)
{
        P5->SEL0 &= ~BIT2;
        P5->SEL1 &= ~BIT2;
        P5->DIR &= ~BIT2;      //Set as input
        P5->IES |= BIT2;       //Set to trigger interrupt from high to low
        P5->IE |= BIT2;        //Enables interrupt
        P5->IFG &= ~BIT2;      //Clears interrupt flag
}
```

```
//function returns the calculated RPMs from revolutions sensed by the hall effect
int measure_RPMs(int Revolution_count) //checks every 10 seconds, SysTick Interrupt == 5
{
   int RPMs;

                RPMs = Revolution_count / 2 * 60; //60 seconds per minute divided by 1 second = 60 intervals / min
                                                                                                                                             //^^^ this will have to be changed depending
on how long I check for it

   return RPMs;
}

//fucntion returns speed using the RPMs measured and the circumference of the wheel from which we measure
float measure_Speed(int RPMs, float circumference)
{
   int speed = 0;

   speed = RPMs * circumference * 60; //revs per minute * circumference in miles * 60 (min / hour)
   return speed;
}
```

Hall_Effect.h

```
/*
 * Hall_Effect.h
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#ifndef HALL_EFFECT_H_
#define HALL_EFFECT_H_

extern volatile int Current_Speed_mph;
extern volatile int RPMs;
extern volatile int Revolution_count;
extern volatile int speed_Dif;
extern volatile int lastSpeed;
extern volatile int firstTime_done;
extern volatile int RPMs_areZero;
extern volatile int last_SpeedingTicket_Speed;

void Hall_Effect_Out_init(void);

int measure_RPMs(int Revolution_count);
float measure_Speed(int RPMs, float circumference);

#endif /* HALL_EFFECT_H_ */
```

HC-SR04.c

```
/*
 * HC-SR04.c
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "HC-SR04.h"

volatile int object_DistanceFromSensor = 0;       //distance variable
volatile float conversion_ToInches = 100.0;        //echo signal to inches
```

```
//initializes the Proximity sensors echo and trigger pins  using TIMERA capture and compare
void HCSR04_init(void)
{
            P7->SEL0 |= (BIT6 | BIT7 );
   P7->SEL1 &= ~(BIT6 | BIT7 );
            P7->DIR |= BIT7;                                                            //pulse pin
            P7->DIR &= ~BIT6;                                                           //echo

            TIMER_A1->CTL = 0x02E4;        //SMCLK, CLK /4, Up Mode, Clear TAR to start
   TIMER_A1->CCR[0] = 65535 - 1;              //87ms period
   TIMER_A1->CCTL[1] = TIMER_A_CCTLN_OUTMOD_6;     //Toggle/set

            TIMER_A1->CCR[1] = 8;                  //PWM duty cycle 10us toggles
            TIMER_A1->CCTL[2] = 0xC900; //rising and falling edge, CCI2A, SCS, capture mode
            TIMER_A1->EX0 = 0x0001;                       //Divide by 4
}

//check at an interval
void measure_Period()
{
   unsigned int last, current;

   float period;

   TIMER_A1->CCTL[2] &= ~1;                  //Clears interrupt flag

   while ((TIMER_A1->CCTL[2] & 1) == 0)
      ;
   //Wait until CCIFG is set

   current = TIMER_A1->CCR[2];                //Save first time stamp

   TIMER_A1->CCTL[2] &= ~1;                   //Clears interrupt flag

   while ((TIMER_A1->CCTL[2] & 1) == 0)
      ;           //Wait until CCIFG is set
   last = current;
   current = TIMER_A1->CCR[2];                //Save second time stamp

   period = current - last; //Calculating difference between rising and falling edge

   object_DistanceFromSensor = period / conversion_ToInches; //Converting cycle time to inches

   //printf("\nDistance = %.1f\n", object_DistanceFromSensor);
}
```

HC-SR04.h

```
/*
 * HC-SR04.h
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#ifndef HC_SR04_H_
#define HC_SR04_H_

extern volatile int object_DistanceFromSensor;       //distance variable
extern volatile float conversion_ToInches;      //echo signal to inches = 148.0;

void HCSR04_init();           //P7.6 | P7.7
void measure_Period();
#endif /* HC_SR04_H_ */
```

I2C_ProjectFunctions.c

```
                                              /*
 * I2C_ProjectFunctions.c
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */

#include "msp.h"
#include "stdio.h"

#include "I2C_ProjectFunctions.h"

#define slaveAddrRTC 0x68


//function only good for one burst of commuication
void I2C1_init(void)
{

   EUSCI_B1->CTLW0 |= 1;            //reset UCB0 to configure

                                              EUSCI_B1->CTLW0 = 0x0F81; //7-bit slave addr, master, I2C, synch mode, use SMCLK

                                              //EUSCI_B1->BRW = 30;
                                              EUSCI_B1->BRW = 120;           //set clock prescalar 12MHz / 120 = 100 kHz

   P6->SEL0 |= 0x30;                //P6.5 CLK and P6.4 DATA for UCB1

   P6->SEL1 &= ~0x30;

   EUSCI_B1->CTLW0 &= ~1;           //enable UCB0 after configuration

}

/***********************

 * BURST READ (MAZIDI)

 *********************/

int I2C1_burstRead(int slaveAdd, unsigned char memAddr, int byteCount,
               unsigned char* data)
{

   if (byteCount <= 0)

      return -1;                //no write was preformed

   EUSCI_B1->I2CSA = slaveAdd;       //setup slave adress

   EUSCI_B1->CTLW0 |= 0x0010;        //enable transmitter

   EUSCI_B1->CTLW0 |= 0x0002;        //generate START and send slave address

   while (!(EUSCI_B1->IFG & 2))
      ;      //wait till ack for transmission

   EUSCI_B1->TXBUF = memAddr;        //send memory address to slave

   while (!(EUSCI_B1->IFG & 2))
      ;      //wait till last transmission

   EUSCI_B1->CTLW0 &= ~0x0010;       //enable reciever
```

```
  EUSCI_B1->CTLW0 |= 0x0002;        //generate RESTART and send slave address

  while (EUSCI_B1->CTLW0 & 2)
    ;       //wait till RESTART is finished

  do
  {
    if (byteCount == 1)            //when only one byte of data is left
      EUSCI_B1->CTLW0 |= 0x004; //setup to send STOP after the last byte is recieved

    while (!(EUSCI_B1->IFG & 1))
      ;    //wait till data is recieved

    *data++ = EUSCI_B1->RXBUF;     //read the recieved data

    byteCount--;

  }
  while (byteCount);

  while (EUSCI_B1->CTLW0 & 4)
    ;       //wait until the STOP is sent

  return 0;                      //no error

}

/***********************

 * BURST WRITE (MAZIDI)

 **********************/

int I2C1_burstWrite(int slaveAdd, unsigned char memAddr, int byteCount,
          unsigned char* data)

{

  if (byteCount <= 0)

    return -1;                  //no write was preformed

  EUSCI_B1->I2CSA = slaveAdd;        //setup slave adress

  EUSCI_B1->CTLW0 |= 0x0010;         //enable transmitter

  EUSCI_B1->CTLW0 |= 0x0002;         //generate START and send slave address

  while (!(EUSCI_B1->IFG & 2))
    ;      //wait till ack for transmission

  EUSCI_B1->TXBUF = memAddr;         //send memory address to slave

  //send data one byte at a time

  do
  {
    while (!(EUSCI_B1->IFG & 2))
      ;    //wait till ready for transmission

    EUSCI_B1->TXBUF = *data++;     //send data to slave

    byteCount--;
```

```
   }
   while (byteCount > 0);

   while (!(EUSCI_B1->IFG & 2))
      ;        //wait till last transmission

   EUSCI_B1->CTLW0 |= 0x0004;         //send STOP

   while (EUSCI_B1->CTLW0 & 4)
      ;        //wait until STOP is sent back

   return 0;
}
```

I2C_ProjectFunctions.h

```
/*
 * I2C_ProjectFunctions.h
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */

#ifndef I2C_PROJECTFUNCTIONS_H_
#define I2C_PROJECTFUNCTIONS_H_
void I2C1_init();
int I2C1_burstWrite(int slaveAdd, unsigned char memAddr, int byteCount,
                   unsigned char* data);
int I2C1_burstRead(int slaveAdd, unsigned char memAddr, int byteCount,
                  unsigned char* data);


#endif /* I2C_PROJECTFUNCTIONS_H_ */
```

MAX219.c

```
/*
 * MAX219.c
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */



#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "MAX219.h"
#include "SysTick_Library.h"

void SPI_init()
{
   //P2.1
   //P2.3
   //

   EUSCI_A1->CTLW0 = 0x0001;          //disable during config
   EUSCI_A1->CTLW0 = 0xA9C1; //CLK polarity: 11, MSB first, 8 bit, master, 3- pin SPI
                             //Synchronous, SMCLK, no STE
                             //0x1010_1001_1000_0001 ::
            EUSCI_A1->BRW = 4;                 //'divide to 128KHz
   EUSCI_A1->CTLW0 &= ~0x0001;

   P2->SEL1 &= ~(BIT3 | BIT1 );       //2.1  CLK
   P2->SEL0 |= (BIT3 | BIT1 );        //2.3 SIMO
   P2->REN |= (BIT3 | BIT1 );
```

```
  P2->OUT |= (BIT3 | BIT1 );

          MAX219_CS_init();                                              //initialize Chip select pin, GPIO
}

//initializes the CS pin of the MAX219, idles high
void MAX219_CS_init()
{
  P2->SEL1 &= ~BIT0;
  P2->SEL0 &= ~BIT0;         //GPIO CS 2.0
  P2->DIR |= BIT0;          //DIR = output
  P2->OUT |= BIT0;           //Idle high
}

void MAX219_SPI_init_NoDecode()
{
  uint8_t i;

  char seg7_setupSequence_CMD[6] = { NODECODEMODECMD, INTENSITYCMD,
                   BIT8MODECMD,
                   DISABLESHUTDOWNCMD, TESTCMD };
  char seg7_setupSequence_DT[6] = { NODECODEMODEDT, INTENSITYDT, BIT8MODEDT,
                  DISABLESHUTDOWNDT, TESTDT };



  for (i = 0; i < 6; i++)         //5 commands, last must be nuls???
  {
    CS_LOW;                     //set CS low (active low)

                     SysTick_delay_ms(1);             //wait 1 millisecond

    while (!(EUSCI_A1->IFG & 2))
      ; //wait for EUSCI flag bit to acknowledge communication is successful

    EUSCI_A1->TXBUF = seg7_setupSequence_CMD[i];       //Send Command

    while (!(EUSCI_A1->IFG & 2))
      ;             //wait again

    EUSCI_A1->TXBUF = seg7_setupSequence_DT[i];       //Send Data

    while (!(EUSCI_A1->IFG & 2))
      ;             //wait for flag

                     SysTick_delay_ms(1);

    CS_HIGH;           //set CS after successful data transfer
  }

  CS_LOW;                             //?????

          SysTick_delay_ms(1);

  while (!(EUSCI_A1->IFG & 2))
    ;

  EUSCI_A1->TXBUF = 0x0F;                 //Command Decode Mode

  while (!(EUSCI_A1->IFG & 2))
    ;

  EUSCI_A1->TXBUF = 0x00;                 //Data Decode Mode
  while (!(EUSCI_A1->IFG & 2))
    ;

          SysTick_delay_ms(1);

  CS_HIGH;
}

void MAX219_SPI_init_Decode() {
          uint8_t i;

          char seg7_setupSequence_CMD[6] = { DECODEMODECMD, INTENSITYCMD, BIT8MODECMD,
                              DISABLESHUTDOWNCMD, TESTCMD };
          char seg7_setupSequence_DT[6] = { DECODEMODEDT, INTENSITYDT, BIT8MODEDT,
                              DISABLESHUTDOWNDT, TESTDT };

          for (i = 0; i < 6; i++)         //5 commands, last must be nuls???
                              {
                    CS_LOW;                   //set CS low (active low)

                    SysTick_delay_ms(1);              //wait 1 millisecond

                    while (!(EUSCI_A1->IFG & 2))
                              ; //wait for EUSCI flag bit to acknowledge communication is successful

                    EUSCI_A1->TXBUF = seg7_setupSequence_CMD[i];      //Send Command

                    while (!(EUSCI_A1->IFG & 2))
                              ;             //wait again

                    EUSCI_A1->TXBUF = seg7_setupSequence_DT[i];      //Send Data

                    while (!(EUSCI_A1->IFG & 2))
                              ;             //wait for flag

                    SysTick_delay_ms(1);

                    CS_HIGH;            //set CS after successful data transfer
          }

          CS_LOW;                         //?????

          SysTick_delay_ms(1);

          while (!(EUSCI_A1->IFG & 2))
                    ;

          EUSCI_A1->TXBUF = 0x0F;                  //Command Decode Mode

          while (!(EUSCI_A1->IFG & 2))
                    ;

          EUSCI_A1->TXBUF = 0x00;                   //Data Decode Mode

          while (!(EUSCI_A1->IFG & 2))
                    ;

          SysTick_delay_ms(1);

          CS_HIGH;

          Print_Zeros_MAX219();

}
```

```
void MAX219_SPI_TransferData(uint8_t cmd, uint8_t data)
{

   CS_LOW;            //set  CS low to initiate communication with this device

                SysTick_delay_ms(1);        //wait 1 ms for good measure

   while (!(EUSCI_A1->IFG & 2))
      ;    //wait

   EUSCI_A1->TXBUF = cmd;         //Send Command

   while (!(EUSCI_A1->IFG & 2))
      ;    //wait

   EUSCI_A1->TXBUF = data;        //Send Data

   while (!(EUSCI_A1->IFG & 2))
      ;    //wait

                SysTick_delay_ms(1);

   CS_HIGH;                      //set CS after communication

}

void Print_DoorAjar_MAX219(void)
{

   uint8_t i;
   uint8_t j = 0;

   //fill array with Data commands for no decode mode
   char seg7_doorAJAr_DT[8] = { MAX219_D_DT, MAX219_o_DT, MAX219_o_DT,
                            MAX219_r_DT, MAX219_A_DT, MAX219_J_DT,
                            MAX219_A_DT,
                            MAX219_r_DT };

                //i go from 8 to 1
                //j goes 0-7
   for (i = 8; i > 0; i--)
   {
      MAX219_SPI_TransferData(i, seg7_doorAJAr_DT[j]);
      j++;
   }

}

//function to turn intensity to zero on digits,
//blanking the MAX219 digits
void blank_MAX219()
{
   MAX219_SPI_TransferData(0x0C, 0x00);
}

//prints zeros to each character of the segment
void Print_Zeros_MAX219(void)
{
   uint8_t i;         //counter to sequence through each character on the MAX

   for (i = 1; i < 9; i++)            //for each character

   {
      CS_LOW;

                              SysTick_delay_ms(1);        //wait 1 ms for good measure

      while (!(EUSCI_A1->IFG & 2))
         ;   //wait for flag

      EUSCI_A1->TXBUF = i;          //will be 1-8

      while (!(EUSCI_A1->IFG & 2))
         ;   //wait

      EUSCI_A1->TXBUF = 0x00;        //Send zeros

      while (!(EUSCI_A1->IFG & 2))
         ;   //wait

                              SysTick_delay_ms(1);

      CS_HIGH;                    //set CS bit

   }
}
```

MAX219.h

```
/*
 * MAX219.h
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#ifndef MAX219_H_
#define MAX219_H_


#define DECODEMODECMD         0x09
#define DECODEMODEDT          0xFF

#define NODECODEMODECMD       0x09
#define NODECODEMODEDT        0x00

#define INTENSITYCMD          0x0A
#define INTENSITYDT           0x03

#define BIT8MODECMD           0x0B
#define BIT8MODEDT            0x07

#define DISABLESHUTDOWNCMD    0x0C
#define DISABLESHUTDOWNDT     0x01

#define TESTCMD               0x0F
#define TESTDT                0x01

#define CS_LOW                P2 -> OUT &= ~BIT0
#define CS_HIGH               P2 -> OUT |= BIT0

//door_ajar values. put in loop going 8 to 0 to print on each character of MAX219
#define MAX219_D_DT           0x3D
#define MAX219_o_DT           0x1D
#define MAX219_r_DT           0x05
#define MAX219_A_DT           0x77
#define MAX219_J_DT           0x38
```

```
void SPI_init(void);
void MAX219_CS_init(void);


void MAX219_SPI_init_Decode(void);

void MAX219_SPI_init_NoDecode(void);

void MAX219_CS_init(void);

void MAX219_SPI_TransferData(uint8_t cmd, uint8_t data);

void Print_Zeros_MAX219(void);

void Print_DoorAjar_MAX219(void);

void blank_MAX219(void);



#endif /* MAX219_H_ */
```

Menus.c

```
/*
 * Menus.c
 *
 *  Created on: Nov 25, 2020
 *      Author: 7jorchay
 */



#include <stdio.h>
#include <stdint.h>
#include "Menus.h"
#include "msp.h"
#include "ST7735.h"
#include "RTC_ProjectFunctions.h"
#include "EncoderFunctions.h"
#include "EEPROM.h"
#include "Project.h"

volatile int menu = 0;
//unsigned char CustomString_ToPrint[20];
//int i = 0;


//Menus
//top left corner = 0,0
//bottom right corner = for font 4, it's 100, 120
//middle x = 32 (size 4)
//middle y = 76 (size 4)
//
//int delay = 1000 * 16;        //delay for splash screen stuff

//Startup menu for LCD
void Startup_Menu()
{
        ST7735_FillScreen(0);                           //set screen to black
        Read_RTC_Print_Date_toLCD();            //read and print date to LCD
        Read_RTC_Print_Time_toLCD();            //read and print Time
```

```
                Read_RTC_Temperature_Print_toLCD();            //read and print the temperature

   char string_toPrint[20];
   uint16_t start_PositionX = 15;
   uint16_t start_PositionY = 110;

   int i = 0;

                //prints Settings highlighted in White
   string_toPrint[0] = 'S';
   string_toPrint[1] = 'e';
   string_toPrint[2] = 't';
   string_toPrint[3] = 't';
   string_toPrint[4] = 'i';
   string_toPrint[5] = 'n';
   string_toPrint[6] = 'g';
   string_toPrint[7] = 's';

                uint16_t text_Color = 0;                    //
   uint16_t bg_Color = 0xFFFF;         //


      for (i = 0; i <= 7; i++)
   {
      ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                  text_Color, bg_Color, 2);
      start_PositionX += 12;
   }
}



//prints the options of the interface menu
void Interface_Options_Menu()
{
                //Output_Clear();
   ST7735_FillScreen(0xFFFF);
   char string_toPrint[20];
   uint16_t start_PositionX = 15;
   uint16_t start_PositionY = 52;

   int i = 0;

   string_toPrint[0] = 'S';
   string_toPrint[1] = 'e';
   string_toPrint[2] = 't';
   string_toPrint[3] = ' ';
   string_toPrint[4] = 'T';
   string_toPrint[5] = 'i';
   string_toPrint[6] = 'm';
   string_toPrint[7] = 'e';
                string_toPrint[8] = NULL;

                //uint16_t text_Color = 0; //white, highlighted will be blue, maybe light blue
   uint16_t bg_Color = 0xFFFF;        //white
   //int delay = 1000 * 16;       //delay for splash screen stuff

   for (i = 0; string_toPrint[i] != NULL; i++)
   {
      ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                             ST7735_BLUE, bg_Color, 2);
      start_PositionX += 12;
   }
```

```
        start_PositionY += 17;
  start_PositionX = 15;

  string_toPrint[0] = 'S';
  string_toPrint[1] = 'e';
  string_toPrint[2] = 't';
  string_toPrint[3] = ' ';
  string_toPrint[4] = 'D';
  string_toPrint[5] = 'a';
  string_toPrint[6] = 't';
  string_toPrint[7] = 'e';
        string_toPrint[8] = NULL;

  for (i = 0; string_toPrint[i] != NULL; i++)
  {
    ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                                0x04E0, bg_Color, 2);
    start_PositionX += 12;
  }

        start_PositionY += 17;
        start_PositionX = 30;


        string_toPrint[0] = 'A';
        string_toPrint[1] = 'l';
        string_toPrint[2] = 'a';
        string_toPrint[3] = 'r';
        string_toPrint[4] = 'm';
        string_toPrint[5] = 's';
        string_toPrint[6] = NULL;

        for (i = 0; string_toPrint[i] != NULL; i++) {
                ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                ST7735_RED, bg_Color, 2);
                start_PositionX += 12;
        }

}

//prints a time format in all zeros, but will reprint with the new value entered when a new digit is being changed
void Set_Time_Menu() {

        ST7735_FillScreen(0xFFFF);
        uint16_t start_PositionX = 20;
        uint16_t start_PositionY = 60;
        uint16_t text_Color = 0xF800;       //blue
        uint16_t bg_Color = 0xFFFF;         //white

        char New_Time[6];

        New_Time[0] = New_RTC_Time_Hour_tens;
        New_Time[1] = New_RTC_Time_Hour_ones;
        New_Time[2] = New_RTC_Time_Minute_tens;
        New_Time[3] = New_RTC_Time_Minute_ones;

        int i = 0;

        for (i = 0; i < 4; i++) {
                ST7735_DrawChar(start_PositionX, start_PositionY, New_Time[i] + '0',
                                text_Color, bg_Color, 2);

                start_PositionX += 12;
        if (i == 1) {
            ST7735_DrawChar(start_PositionX, start_PositionY, ':', text_Color,
                    bg_Color, 2);
            start_PositionX += 12;
        }
    }
}

//same function as the Set time menu
//reprints with newly set Date digits
void Set_Date_Menu() {
    ST7735_FillScreen(0xFFFF);
    uint16_t start_PositionX = 5;
    uint16_t start_PositionY = 60;
    uint16_t text_Color = ST7735_MAGENTA;       //
    uint16_t bg_Color = 0xFFFF;       //white

    char New_Date[7];

    New_Date[0] = New_RTC_Date_Day_ones;
    New_Date[1] = New_RTC_Date_Month_tens;
    New_Date[2] = New_RTC_Date_Month_ones;
    New_Date[3] = New_RTC_Date_Date_tens;
    New_Date[4] = New_RTC_Date_Date_ones;
    New_Date[5] = New_RTC_Date_Year_tens;
    New_Date[6] = New_RTC_Date_Year_ones;

    int i = 0;

    for (i = 0; i < 7; i++) {
        ST7735_DrawChar(start_PositionX, start_PositionY, New_Date[i] + '0',
                text_Color, bg_Color, 2);

        text_Color = ST7735_RED;
        start_PositionX += 12;

        if (i == 0) {
            ST7735_DrawChar(start_PositionX, start_PositionY, ' ', text_Color,
                    bg_Color, 2);
            start_PositionX += 12;
        }

        if ((i == 2) || (i == 4)) {
            ST7735_DrawChar(start_PositionX, start_PositionY, '/', text_Color,
                    bg_Color, 2);
            start_PositionX += 12;
        }
    }


}

//menu display for the alarms

void Alarms_Menu() {

    //Output_Clear();
    ST7735_FillScreen(0);
    char string_toPrint[20];
    uint16_t start_PositionX = 15;
    uint16_t start_PositionY = 10;
    uint16_t text_Color = 0x0FF; //orange , highlighted will be blue, maybe light blue
    uint16_t bg_Color = 0;
```

```
            int i = 0;

//          string_toPrint[0] = 'C';
//          string_toPrint[1] = 'o';
//          string_toPrint[2] = 'l';
//          string_toPrint[3] = 'l';
//          string_toPrint[4] = 'i';
// string_toPrint[5] = 's';
//          string_toPrint[6] = 'i';
//          string_toPrint[7] = 'o';
//          string_toPrint[8] = 'n';
//          string_toPrint[9] = 's';
//          string_toPrint[10] = NULL;

            string_toPrint[0] = 'A';
            string_toPrint[1] = 'L';
            string_toPrint[2] = 'A';
            string_toPrint[3] = 'R';
            string_toPrint[4] = 'M';
            string_toPrint[5] = 'S';
            string_toPrint[6] = NULL;

            ST7735_FillRect(0, 0, 140, 3, ST7735_YELLOW);

  for (i = 0; string_toPrint[i] != NULL; i++)
  {
    ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
                                                text_Color, bg_Color, 2);
                        start_PositionX += 12;
  }

            ST7735_FillRect(0, 24, 140, 2, ST7735_MAGENTA);

//          start_PositionY += 65;
//          start_PositionX = 15;

//   string_toPrint[0] = 'S';
//   string_toPrint[1] = 'p';
//   string_toPrint[2] = 'e';
//   string_toPrint[3] = 'e';
//   string_toPrint[4] = 'd';
//   string_toPrint[5] = 'i';
//   string_toPrint[6] = 'n';
//   string_toPrint[7] = 'g';
//          string_toPrint[8] = 'T';
//          string_toPrint[9] = 'i';
//          string_toPrint[10] = 'c';
//          string_toPrint[11] = 'k';
//          string_toPrint[12] = 'e';
//          string_toPrint[13] = 't';
//          string_toPrint[14] = 's';
//          string_toPrint[15] = NULL;

//   for (i = 0; string_toPrint[i] != NULL; i++)
//   {
//     ST7735_DrawChar(start_PositionX, start_PositionY, string_toPrint[i],
//                                              0xF7E0, bg_Color, 1);
//                      start_PositionX += 7;
//
//                      if (i == 7) {
//                                  start_PositionY += 10;
//                                  start_PositionX = 15;
//                      }
//	}

	//ST7735_FillRect(0, 95, 140, 2, ST7735_MAGENTA);

	for (i = 0; i < 5; i++) {
		read_Date_and_Time_EEPROM(i);
	}

	ST7735_FillRect(0, 157, 140, 3, ST7735_YELLOW);
}


//function to decide which menu is next
//dependent on the current menu and total turns
void MENU() {

	//startup = 0
	//interface options = 1
	//Set time menu = 2
	//set date menu = 3
	//alarms menu = 4
	if (menu == 0) {
		menu = 1;
		Interface_Options_Menu();
	}

	else if ((menu == 1) && (total_turns == 0)) {//interface menu, cursor on Set_
		menu = 2;
		Set_Time_Menu();
	}

	else if ((menu == 1) && (total_turns == 1)) {//interface menu, cursor on Set_
		menu = 3;
		Set_Date_Menu();
	}

	else if ((menu == 1) && (total_turns == 2)) {//interface menu, cursor on Set_
		menu = 4;
		Alarms_Menu();
	}
	else if (menu == 2) {//if menu is greater than 1, function will reset display to startup menu
		menu = 0;
		//function to set new Time
		Startup_Menu();

	} else if (menu == 3) {
		menu = 0;
		//function to set new Date
		Startup_Menu();

	} else if (menu == 4) {
		menu = 0;
		Startup_Menu();
	}

	reset_Encoder_Counts();					//at new menu, reset encoder counts
	Encoder_Btn_pressed = 0;				//reset interrupt and debounce global flags
	Encoder_Btn_interruptFlag = 0;
}
```

Menus.h

```
/*
 * Menus.h
 *
 *  Created on: Nov 25, 2020
 *     Author: 7jorchay
 */

#ifndef MENUS_H_
#define MENUS_H_
extern volatile int menu;
void Menus_main(void);
void Startup_Menu(void);

void Interface_Options_Menu();
void Set_Time_Menu();
void Set_Date_Menu();
void Alarms_Menu();
void MENU();
#endif /* MENUS_H_ */
```

Photocell.c

```
/*
 * Photocell.c
 *
 *  Created on: Dec 1, 2020
 *     Author: 7jorchay
 */


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

volatile int TA0_1_dutyCycle = 30000;

void ADC_pin_init(void)
{
   P6->SEL0 |= BIT1;
   P6->SEL1 |= BIT1;
}
void ADC_StartConversion_WithInterrupts(void)
{
                ADC14->CTL0 &= ~ADC14_CTL0_ENC;     //disable, no conversions runnning
                ADC14->CTL0 |= 0x04200210; //Predivide 1, ACD14SC bit, Source from timer, no inversion, divide 2,
                                                                                    //single channel single conv, not active, 16 cycle sample and hold, rising edge,
core on

                ADC14->CTL1 = 0x00000030;          //14 bit resolution
                ADC14->CTL1 |= 0x0000000;                                        //in case any other ADC is used
                ADC14->MCTL[0] = 0x0E;            //attaching 14th channel to mem[0]
                ADC14->CTL0 |= ADC14_CTL0_ENC;      //enable, start conversion
}

void ADC_Reading(void)
{

                static volatile uint16_t result;      //local variable for result in MEM[0]
                float V;                             //converted Voltage reading
                ADC14->CTL0 |= ADC14_CTL0_SC;
   if (ADC14->IFGR0 & BIT0)
   {

                                result = ADC14->MEM[0];               //result equals converted value

                                V = ((result * 3.3) / 16384);                              //V conversion from raw ADC readings
                                if (V <= .33)      //conditions are inverted. If V is low, duty is high
      {
```

```
      TA0_1_dutyCycle = 60000;
    }
    if (V > .33 && V <= 0.66)
    {
      TA0_1_dutyCycle = (9.75 * 6000);
    }
    if (V > 0.66 && V <= 0.99)
    {
      TA0_1_dutyCycle = (9.5 * 6000);
    }
    if (V > 0.99 && V <= 1.32)
    {
      TA0_1_dutyCycle = (8.5 * 6000);
    }
    if (V > 1.32 && V <= 1.65)
    {
      TA0_1_dutyCycle = (7.5 * 6000);
    }
    if (V > 1.65 && V <= 1.98)
    {
      TA0_1_dutyCycle = (5.5 * 6000);
    }
    if (V > 1.98 && V <= 2.31)
    {
      TA0_1_dutyCycle = (3.5 * 6000);
    }
    if (V > 2.31 && V <= 2.64)
    {
      TA0_1_dutyCycle = (1.5 * 6000);
    }
    if (V > 2.64 && V <= 2.97)
    {
      TA0_1_dutyCycle = (1 * 6000);
    }

    if (V > 2.97)
    {
      TA0_1_dutyCycle = (0.5 * 6000);
    }

                      ADC14->CTL0 |= ADC14_CTL0_SC; //restart conversion after changing global duty cycle

  }
}


void TimerA0_1_PWM_init(void)
{          //Timer A initialization function
  P2->SEL0 |= BIT4;                //P2.4 set to TA0.1
  P2->SEL1 &= ~BIT4;
  P2->DIR |= BIT4;                 //P2.4 Set to Output

  TIMER_A0->CCR[0] = 60000;              //20ms period
  TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;    //CCR1 reset
  TIMER_A0->CCR[1] = TA0_1_dutyCycle;              //PWM duty cycle
  TIMER_A0->CTL = 0x0294;            //SMCLK, Up Mode, Clear TAR to start
}
void change_TIMERA0_1_dutyCycle()
{
           TIMER_A0->CCR[1] = TA0_1_dutyCycle; //changes TIMERA0.1 PWM, the backlight LED for LCD
}
```

Photocell.h

```
/*
 * Photocell.h
 *
 *  Created on: Dec 1, 2020
 *      Author: 7jorchay
 */

#ifndef PHOTOCELL_H_
#define PHOTOCELL_H_
extern volatile int TA0_1_dutyCycle;

void Clock_Init48MHz(void);
void ADC_pin_init(void);              //initialize ADC pin 5.0
void ADC_StartConversion_WithInterrupts(void);       //starts ADC conversion
void ADC_Reading(void); //reads value in ADC register and converts to readable value
void TimerA0_1_PWM_init(void);         //LED
void change_TIMERA0_1_dutyCycle(void);
#endif /* PHOTOCELL_H_ */
```

ProjectButtons.c

```
/*
 * ProjectButtons.c
 *
 *  Created on: Nov 26, 2020
 *      Author: 7jorchay
 */

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

volatile int Open_Door_Btn_pressed = 0;                                        //global debounce variables
volatile int NewRTC_Info_Btn_pressed = 0;
volatile int WatchdogReset_Btn_pressed = 0;

//initializes two pins connected to toggle switch for blinkers, acts like a button
//no debounce required, 3.6 and 3,7
void Blinkers_Toggle_Switch_init()
{
   P3->SEL0 &= ~BIT6;         //P3.6 set to GPIO
   P3->SEL1 &= ~BIT6;
   P3->DIR &= ~BIT6;          //P3.6 direction set to input
   P3->REN |= BIT6;           //activating the internal resistor
   P3->OUT |= BIT6;           //declaring the internal resistor as pull-up
   P3->SEL0 &= ~BIT7;         //P3.7 set to GPIO
   P3->SEL1 &= ~BIT7;
   P3->DIR &= ~BIT7;          //P3.7 direction set to input
   P3->REN |= BIT7;           //activating the internal resistor
   P3->OUT |= BIT7;           //declaring the internal resistor as pull-up
}

//initialize door ajar button on 3.5
void openDoor_Btn_init()
{
   P3->SEL0 &= ~BIT5;         //P3.5 set to GPIO
   P3->SEL1 &= ~BIT5;
   P3->DIR &= ~BIT5;          //P3.5 direction set to input
   P3->REN |= BIT5;           //activating the internal resistor
   P3->OUT |= BIT5;           //declaring the internal resistor as pull-up
   P3->IES |= BIT5;          //interrupt edge select on falling edge
   P3->IE |= BIT5;           //Interrupt enable
   P3->IFG &= ~BIT5;         //Clears the flag bit
}

//initialize simple LED output for right blinker on 8.3
void Left_Blinker()
```

```
{
   P8->SEL0 &= ~BIT3;
   P8->SEL1 &= ~BIT3;        //P8.3 direction set to output
   P8->DIR |= BIT3;          //P8.3 direction set to output
   P8->OUT &= ~BIT3;         //P8.3 set to off or as 0
}

//initialize simple LED output for right blinker on 8.4
void Right_Blinker()
{
   P8->SEL0 &= ~BIT4;
   P8->SEL1 &= ~BIT4;        //P8.4 direction set to output
   P8->DIR |= BIT4;          //P8.4 direction set to output
   P8->OUT &= ~BIT4;         //P8.4 set to off or as 0
}

//initialize simple LED output for proximity warning, 8.2
void LED_closeProximity()
{
   P8->SEL0 &= ~BIT2;
   P8->SEL1 &= ~BIT2;        //P8.2 direction set to output
   P8->DIR |= BIT2;          //P8.2 direction set to output
   P8->OUT &= ~BIT2;         //P8.2 set to off or as 0

   /*             For TIMER A intensity if you get there
    P8->SEL0 |= BIT2;                        //P5.6 set to TA2.1
    P8->SEL1 &= ~BIT2;
    P8->DIR |= BIT2;                         //P5.6 Set to Output
    TIMER_A3->EX0 |= 0x0003; //Expand the flag timer to run slower to encapsulate a 1ms interrupt
    TIMER_A3->CTL = 0x02D6; //Timer to interrupt every 1 ms to set checking flags.
    TIMER_A3->CCR[0] = LED_Period;
    TIMER_A3->CCTL[2] = 0x00E0;
    TIMER_A3->CCR[2] = duty_cycle;           //PWM duty cycle
    */
}
//initialize watchdog reset button on 3.2
void NewRTC_Info_Btn() {
        P3->SEL0 &= ~BIT2;
        P3->SEL1 &= ~BIT2;
        P3->DIR &= ~BIT2;              //Set as input
        P3->REN |= BIT2;               //enable resistor
        P3->OUT |= BIT2;               //set as pull up
        P3->IES |= BIT2;             //Set to trigger interrupt from high to low
        P3->IFG &= ~BIT2;              //Clears interrupt flag
        P3->IE |= BIT2;                //Enables interrupt
}
//initialize watchdog reset button on 3.3
void WatchdogReset_Btn() {
        P3->SEL0 &= ~BIT3;
        P3->SEL1 &= ~BIT3;
        P3->DIR &= ~BIT3;              //Set as input
        P3->REN |= BIT3;               //enable resistor
        P3->OUT |= BIT3;               //set as pull up

        P3->IES |= BIT3;             //Set to trigger interrupt from high to low

        P3->IFG &= ~BIT3;              //Clears interrupt flag

        P3->IE |= BIT3;                //Enables interrupt
}

//debounce for New_RTC_info button
int Debounce_NewRTC_Info_Btn() {
        static uint16_t State_1 = 0;                //Current debounce status
```

```
          State_1 = (State_1 << 1) | (P3IN & 0x04) >> 2 | 0xf800; //checks pin 3.2, don

          if (State_1 == 0xfc00) {
                     NewRTC_Info_Btn_pressed = 1;
                     return 1;                     //after 0 level is

                      //stable for 10 consecutive calls
          } else
                     return 0;
}

//Door ajar button debounce
int Debounce_DoorOpen_Btn()
{
  static uint16_t State_1 = 0;           //Current debounce status
  State_1 = (State_1 << 1) | (P3IN & 0x20) >> 5 | 0xf800; //checks pin 3.5, don
  if (State_1 == 0xfc00)
  {
    Open_Door_Btn_pressed = 1;
    return 1;                         //after 0 level is
                              //stable for 10 consecutive calls
  }
  else
    return 0;
}
//debounces watchdog button, returns 1 if confirmed, along with the gloabl debounce flag
int Debounce_WatchodgReset_Btn() {
          static uint16_t State_1 = 0;           //Current debounce status

          State_1 = (State_1 << 1) | (P3IN & BIT3 ) >> 3 | 0xf800; //checks pin 3.0,

          if (State_1 == 0xfc00) {
                     WatchdogReset_Btn_pressed = 1;
                     return 1;                     //after 0 level is

                      //stable for 10 consecutive calls
          } else
                     return 0;
}
```

ProjectButtons.h

```
/*
 * ProjectButtons.h
 *
 *  Created on: Nov 26, 2020
 *      Author: 7jorchay
 */
#ifndef PROJECTBUTTONS_H_
#define PROJECTBUTTONS_H_

extern volatile int Open_Door_Btn_pressed;
extern volatile int NewRTC_Info_Btn_pressed;
extern volatile int WatchdogReset_Btn_pressed;
void Blinkers_Toggle_Switch_init(void);
void Left_Blinker(void);               //left blinker
void Right_Blinker(void);              //right blinker
void LED_closeProximity();
void openDoor_Btn_init(void);
int Debounce_DoorOpen_Btn();
void NewRTC_Info_Btn();
int Debounce_NewRTC_Info_Btn();
void WatchdogReset_Btn();
int Debounce_WatchodgReset_Btn();
```

```
#endif /* PROJECTBUTTONS_H_ */
```

RTC_ProjectFunctions.c

```
/*
 * RTC_ProjectFunctions.c
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */
#include "RTC_ProjectFunctions.h"
#include "I2C_ProjectFunctions.h"
#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "ST7735.h"
#include "Project.h"
#include "SysTick_Library.h"
//char *Message1 = "WARNING\0";
void Read_RTC(unsigned char* Time_Date_Array) //Can only use this if global works.
{                       //functions to print will no longer accept a parameter
   I2C1_init();
   I2C1_burstRead(slaveAddrRTC, 0, 7, Time_Date_Array);
}

//writes 3 bytes of a new time to RTC using I2C to correspinding bytes
void Write_New_Time_RTC(unsigned char* New_Time_Array) {
                I2C1_init();
                I2C1_burstWrite(slaveAddrRTC, 0, 3, New_Time_Array);
}
//writes a new date to RTC with 4 bytes to corresponding bytes
void Write_New_Date_RTC(unsigned char* New_Date_Array) {
                I2C1_init();
                I2C1_burstWrite(slaveAddrRTC, 3, 4, New_Date_Array);
}
//reads the RTC date and calls function to print to LCD
void Read_RTC_Print_Date_toLCD()
{
   I2C1_init();
   unsigned char RTC_Date_Time[7];
   I2C1_burstRead(slaveAddrRTC, 0, 7, RTC_Date_Time);
   print_Date_LCD(RTC_Date_Time);
}
//reads RTC time bytes andcalls a function to print to LCD
void Read_RTC_Print_Time_toLCD()
{
   I2C1_init();
   unsigned char RTC_Date_Time[7];
   I2C1_burstRead(slaveAddrRTC, 0, 7, RTC_Date_Time);
   print_Time_LCD(RTC_Date_Time);
}

//Reads RTC temperature bytes and calls function to print to LCD in a readable way
void Read_RTC_Temperature_Print_toLCD()
{
   I2C1_init();

   unsigned char Temp_toPrint[2];      //reading 2 bytes, 1 place for ack bit

   I2C1_burstRead(slaveAddrRTC, 17, 2, Temp_toPrint); //address of temp MSB is 0x11 = 17, 2 bytes

   print_Temperature_toLCD(Temp_toPrint);

}
```

```
//prints date to LCD using data read from RTC
//converts read bytes to character values to print
void print_Date_LCD(unsigned char* Time_Date_Array)                      ///
{

   unsigned char Date_Array[4] = { Time_Date_Array[5], Time_Date_Array[4],
                 Time_Date_Array[6], NULL };
        unsigned char Day_ofWeek[11];

        //char *Day_ofWeek;

        uint16_t start_PositionX = 45;
        uint16_t start_PositionY = 40;

   uint16_t text_Color = 0xF81F;     //Purple
   uint16_t bg_Color = 0;            //black

   uint8_t num_toPrint_ones;
   uint8_t num_toPrint_tens;

   int i = 0;

        if (Time_Date_Array[3] == 0x01) {
                Day_ofWeek[0] = 'S';
                Day_ofWeek[1] = 'u';
                Day_ofWeek[2] = 'n';
                Day_ofWeek[3] = 'd';
                Day_ofWeek[4] = 'a';
                Day_ofWeek[5] = 'y';
                Day_ofWeek[7] = ',';
                Day_ofWeek[6] = NULL;
                text_Color = ST7735_YELLOW; //yellow
        }

        else if (Time_Date_Array[3] == 0x02) {
                Day_ofWeek[0] = 'M';
                Day_ofWeek[1] = 'o';
                Day_ofWeek[2] = 'n';
                Day_ofWeek[3] = 'd';
                Day_ofWeek[4] = 'a';
                Day_ofWeek[5] = 'y';
                Day_ofWeek[6] = ',';
                Day_ofWeek[7] = NULL;
                text_Color = 0x0015;     //dull red
        }

        else if (Time_Date_Array[3] == 0x03) {
                Day_ofWeek[0] = 'T';
                Day_ofWeek[1] = 'u';
                Day_ofWeek[2] = 'e';
                Day_ofWeek[3] = 's';
                Day_ofWeek[4] = 'd';
                Day_ofWeek[5] = 'a';
                Day_ofWeek[6] = 'y';
                Day_ofWeek[7] = ',';
                Day_ofWeek[8] = NULL;
                text_Color = 0x07EA;     //lime green
        }

        else if (Time_Date_Array[3] == 0x04) {
                Day_ofWeek[0] = 'W';
                Day_ofWeek[1] = 'e';
                Day_ofWeek[2] = 'd';
                Day_ofWeek[3] = 'n';
                Day_ofWeek[4] = 'e';
                Day_ofWeek[5] = 's';
                Day_ofWeek[6] = 'd';
                Day_ofWeek[7] = 'a';
                Day_ofWeek[8] = 'y';
                Day_ofWeek[9] = ',';
                Day_ofWeek[10] = NULL;
                text_Color = ST7735_RED;
        }

        else if (Time_Date_Array[3] == 0x05) {
                Day_ofWeek[0] = 'T';
                Day_ofWeek[1] = 'h';
                Day_ofWeek[2] = 'u';
                Day_ofWeek[3] = 'r';
                Day_ofWeek[4] = 's';
                Day_ofWeek[5] = 'd';
                Day_ofWeek[6] = 'a';
                Day_ofWeek[7] = 'y';
                Day_ofWeek[8] = ',';
                Day_ofWeek[9] = NULL;
                text_Color = 0xF800;
        }

        else if (Time_Date_Array[3] == 0x06) {
                Day_ofWeek[0] = 'F';
                Day_ofWeek[1] = 'r';
                Day_ofWeek[2] = 'i';
                Day_ofWeek[3] = 'd';
                Day_ofWeek[4] = 'a';
                Day_ofWeek[5] = 'y';
                Day_ofWeek[6] = ',';
                Day_ofWeek[7] = NULL;
                text_Color = ST7735_BLUE;
        }

        else if (Time_Date_Array[3] == 0x07) {
                Day_ofWeek[0] = 'S';
                Day_ofWeek[1] = 'a';
                Day_ofWeek[2] = 't';
                Day_ofWeek[3] = 'u';
                Day_ofWeek[4] = 'r';
                Day_ofWeek[5] = 'd';
                Day_ofWeek[6] = 'a';
                Day_ofWeek[7] = 'y';
                Day_ofWeek[8] = ',';
                Day_ofWeek[9] = NULL;
                text_Color = ST7735_CYAN;
        }
        for (i = 0; Day_ofWeek[i] != NULL; i++) {
                ST7735_DrawChar(start_PositionX, 30, Day_ofWeek[i], text_Color,
                                bg_Color, 1);

                start_PositionX += 6;
        }
   text_Color = 0xF81F;     //Purple

        //start_PositionY += 10;
   start_PositionX = 40;

   for (i = 0; Date_Array[i] != NULL; i++)
   {
     num_toPrint_tens = ((Date_Array[i] & 0xF0) >> 4) + '0';
```

```
    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_tens,
            text_Color, bg_Color, 1);

    start_PositionX += 6;

    num_toPrint_ones = (Date_Array[i] & 0x0F) + '0';

    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_ones,
            text_Color, bg_Color, 1);

    start_PositionX += 6;

    if (i != 2)
    {
      ST7735_DrawChar(start_PositionX, start_PositionY, '/', text_Color,
              bg_Color, 1);
      start_PositionX += 6;
    }
  }

}

//prints time to LCD using data read from RTC
//converts read bytes to character values to print
void print_Time_LCD(unsigned char* Time_Date_Array)
{
  //Time_Data_array[0] = seconds
  //Time_Data_array[1] = minues
  //Time_Data_array[2] = hours

//   uint16_t start_PositionX = 40;                     //centers time under Date
//   uint16_t start_PositionY = 42;
//   uint16_t text_Color = 0xF800;     //blue
//   uint16_t bg_Color = 0;        //white

          uint16_t start_PositionX = 20;                              //makes time Lower and Bigger
          uint16_t start_PositionY = 60;
  uint16_t text_Color = 0xF800;     //blue
          uint16_t bg_Color = 0;         //black


  uint8_t num_toPrint_ones;
  uint8_t num_toPrint_tens;

  int i = 0;

  for (i = 2; i >= 0; i--)
  {
    num_toPrint_tens = ((Time_Date_Array[i] & 0xF0) >> 4) + '0';

    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_tens,
                              text_Color, bg_Color, 2);

                 start_PositionX += 12;

    num_toPrint_ones = (Time_Date_Array[i] & 0x0F) + '0';

    ST7735_DrawChar(start_PositionX, start_PositionY, num_toPrint_ones,
                              text_Color, bg_Color, 2);

                 start_PositionX += 12;

    if (i != 0)
    {
        ST7735_DrawChar(start_PositionX, start_PositionY, ':', text_Color,
                                                          bg_Color, 2);
                                          start_PositionX += 12;
      }
    }
}

//takes data from RTC and converts it to fahrenheit, then converts to charater value to be printed
//can print 2 or 3 digit temperature
//high cabin temp warning will be displayed
void print_Temperature_toLCD(unsigned char* Temperature_Array)
{
    int temp_tens;              //local variable for MSB of temp byte
    int temp_ones;      //local variable for LSB of temp byte, .25 precision

    uint16_t start_PositionX = 2;
    uint16_t start_PositionY = 12;
    uint16_t text_Color = ST7735_YELLOW;        //yellow

    uint16_t bg_Color = 0;              //white

    uint8_t num_toPrint_ones;
    uint8_t num_toPrint_tens;
    uint8_t num_toPrint_hundreds;


    float temp_Celsius;     //variable to combine the bytes into a 10 bit number
    float temp_Fahrenheit;

    temp_tens = Temperature_Array[0] << 2; //ints are 32 bit, we can set it to the MSB of temp and shift it twice left
    temp_ones = Temperature_Array[1] >> 6; //LSB of temp only has data on bits 6 and 7 so we shift 6

    temp_Celsius = (temp_tens | temp_ones) * .25; //now we can or them for a 10 bit number
                                    //times .25 because 2 least sig bits are decimal places
                                    //the shift had multiplied our actual temp by 2^2, so we divide
    temp_Fahrenheit = temp_Celsius * 1.8 + 32;


    num_toPrint_ones = (int) temp_Fahrenheit % 10;

    num_toPrint_tens = (((int) temp_Fahrenheit - num_toPrint_ones) / 10) % 10;

    num_toPrint_hundreds = (((int) temp_Fahrenheit
        - (num_toPrint_ones + 10 * num_toPrint_tens)) / 100) % 10;

    //num_toPrint_tens = (((int) temperature_Fahrenheit & 0xF0) >> 4) + '0';

    if (num_toPrint_hundreds == 0)
    {
        ST7735_DrawChar(start_PositionX, start_PositionY,
                num_toPrint_tens + '0',
            text_Color, bg_Color, 2);

                        start_PositionX += 12;

        //num_toPrint_ones = ((int) temperature & 0x0F) + '0';

        ST7735_DrawChar(start_PositionX, start_PositionY,
                num_toPrint_ones + '0',
            text_Color, bg_Color, 2);

                        start_PositionX += 12;

    ST7735_DrawChar(start_PositionX, start_PositionY, (char) 247, text_Color,
            bg_Color, 2);                         //degree symbol

    start_PositionX += 12;

       ST7735_DrawChar(start_PositionX, start_PositionY, 'F', text_Color,
                bg_Color, 1);
    }

    else
    {
       displayHiCabinTemp();
    }
}
```

RTC_ProjectFunctions.h

```
/*
 * RTC.h
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */
#ifndef RTC_H_
#define RTC_H_
extern char *Message1;
void Read_RTC();

void Write_New_Time_RTC(unsigned char* New_Time_Array);
void Write_New_Date_RTC(unsigned char* New_Time_Array);
void Read_RTC_Temperature_Print_toLCD();
void print_Temperature_toLCD(unsigned char* Temperature_Array);
void Read_RTC_Print_Date_toLCD();
void print_Date_LCD(unsigned char* Time_Date_Array);
void Read_RTC_Print_Time_toLCD();
void print_Time_LCD(unsigned char* Time_Date_Array);

#endif /* RTC_H_ */
```

Songs.c

```
/*
 * Songs.c
 *
 *  Created on: Dec 3, 2020
 *      Author: 7jorchay
 */

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

//Note frequencies in .h

uint8_t song = 0;

void Speaker(int Note, int on) { //on = 2, if you want it off, call Speaker(#, 0)

                int freq = 3000000 / Note;

                P7->SEL0 |= BIT7;                        //Set TimerA

                P7->SEL1 = 0;
                P7->DIR |= BIT7;                         //Set P2.4 for OutPut
                TIMER_A1->CCR[0] = freq;                     //Period based off note input
```

```
        TIMER_A1->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;     //CCR1 reset
        TIMER_A1->CCR[1] = freq / on;                   //PWM duty cycle
        TIMER_A1->CTL = 0x0214;                 //SMCLK, Up Mode, Clear TAR to start

}

void Speaker_3MHz(int Note, int on) {//on = 2, if you want it off, call Speaker(#, 0)

        int freq = 3000000 / Note;

        P7->SEL0 |= BIT7;                       //Set TimerA

        P7->SEL1 = 0;

        P7->DIR |= BIT7;                        //Set P2.4 for OutPut

        TIMER_A1->CCR[0] = freq;                //Period based off note input

        TIMER_A1->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;     //CCR1 reset

        TIMER_A1->CCR[1] = freq / on;                   //PWM duty cycle

        TIMER_A1->CTL = 0x0214;                 //SMCLK, Up Mode, Clear TAR to start

}

//initializes speaker for a 12MHz clock
void Speaker_12MHz(int Note, int on) {

        int freq = 12000000 / Note;

        P5->SEL0 |= BIT6;                       //Set TimerA

        P5->SEL1 &= ~BIT6;

        P5->DIR |= BIT6;                        //Set P2.4 for OutPut

        TIMER_A2->CCR[0] = freq;                //Period based off note input

        TIMER_A2->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;     //CCR1 reset

        TIMER_A2->CCR[1] = freq / on;                   //PWM duty cycle

        TIMER_A2->CTL = 0x0214;                 //SMCLK, Up Mode, Clear TAR to start
}

void Bells(int repeats) {

        int i;

        for (i = repeats; i > 0; i--) {         //For loop allows for repeating

                Speaker_12MHz(D_5, 2);                  //Note frequency D_5 plays

                SysTick_delay_ms(50);           //Delays for 50ms

                Speaker_12MHz(C_5, 2);                  //Plays note frequency C_5

                SysTick_delay_ms(50);           //Delays 50ms
        }
        Speaker_12MHz(0, 0);                    //Speaker off
}

void Pirates_C(void) {
```

```
//FIRST PART
Speaker(E_4, 2);
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(250);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(250);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(250);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
```

```
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(375);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);
//END OF PART 1

//PART 2
Speaker(E_4, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(250);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(250);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(B_4, 2);
```

```
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(250);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(375);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);
//END OF PART 2

//PART 3
Speaker(E_4, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(250);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(250);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
```

```
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(E_5, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(F_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT3 );
SysTick_delay_ms(250);
Speaker(F_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT3 );
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(E_5, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(E_5, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(250);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);
//END OF PART 3

//PART 4
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7 );
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(250);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 );
SysTick_delay_ms(125);
Speaker(0, 0);
```

```
	P3->OUT = 0;
	SysTick_delay_ms(125);

	Speaker(D_5, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
	SysTick_delay_ms(250);
	Speaker(E_5, 2);
	P3->OUT = 0;
	P3->OUT |= BIT0;
	SysTick_delay_ms(125);
	Speaker(A_4, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT5 | BIT6 | BIT7 );
	SysTick_delay_ms(250);
	Speaker(0, 0);
	P3->OUT = 0;
	SysTick_delay_ms(125);

	Speaker(A_4, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT5 | BIT6 | BIT7 );
	SysTick_delay_ms(125);
	Speaker(C_5, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT0 | BIT2 | BIT3 );
	SysTick_delay_ms(125);
	Speaker(B_4, 2);
	P3->OUT = 0;
	P3->OUT |= BIT3;
	SysTick_delay_ms(250);
	Speaker(B_4, 2);
	P3->OUT = 0;
	P3->OUT |= BIT3;
	SysTick_delay_ms(125);
	Speaker(0, 0);
	P3->OUT = 0;
	SysTick_delay_ms(125);

	Speaker(C_5, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT0 | BIT2 | BIT3 );
	SysTick_delay_ms(125);
	Speaker(A_4, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT5 | BIT6 | BIT7 );
	SysTick_delay_ms(125);
	Speaker(B_4, 2);
	P3->OUT = 0;
	P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 );
	SysTick_delay_ms(375);
	Speaker(0, 0);
	P3->OUT = 0;
	SysTick_delay_ms(1000);
	//END OF PART 4

}

void Despacito(void) {

	Speaker(D_5, 2);
	P3->OUT |= BIT0;		//Blue
	SysTick_delay_ms(575);
	Speaker(0, 0);
```

```
P3->OUT = 0;                    //Off
SysTick_delay_ms(200);
Speaker(C_S5, 2);
P3->OUT |= BIT2;                //Orange
SysTick_delay_ms(575);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(200);
Speaker(B_4, 2);
P3->OUT |= BIT3;                //Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7 );    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);

Speaker(B_4, 2);
P3->OUT |= BIT3;                //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
```

```
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);

Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7 );       //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);

Speaker(B_4, 2);
```

```
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;            //Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);

Speaker(C_S5, 2);
P3->OUT |= BIT2;            //Orange
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(D_5, 2);
P3->OUT |= BIT0;             //Blue
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;              //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7 ); //All
```

```
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);

Speaker(D_5, 2);
P3->OUT |= BIT0;              //Blue
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);
Speaker(C_S5, 2);
P3->OUT |= BIT2;              //Orange
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);
Speaker(D_5, 2);
P3->OUT |= BIT0;              //Blue
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);
Speaker(C_S5, 2);
P3->OUT |= BIT2;              //Orange
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);

Speaker(D_5, 2);
P3->OUT |= BIT0;              //Blue
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(100);

Speaker(E_5, 2);
P3->OUT |= (BIT3 | BIT5 );        //Red and Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(50);
Speaker(E_5, 2);
P3->OUT |= (BIT3 | BIT5 );        //Red and Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(100);
Speaker(C_S5, 2);
P3->OUT |= BIT2;              //Orange
SysTick_delay_ms(575);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(400);

Speaker(D_5, 2);
P3->OUT |= BIT0;              //Blue
SysTick_delay_ms(575);
Speaker(0, 0);
P3->OUT = 0;                //Off
SysTick_delay_ms(200);
Speaker(C_S5, 2);
P3->OUT |= BIT2;              //Orange
```

```
		SysTick_delay_ms(575);
		Speaker(0, 0);
		P3->OUT = 0;			//Off
		SysTick_delay_ms(200);
		Speaker(B_4, 2);
		P3->OUT |= BIT3;			//Yellow
		SysTick_delay_ms(290);
		Speaker(0, 0);
		P3->OUT = 0;			//Off
		SysTick_delay_ms(100);
		Speaker(F_S4, 2);
		P3->OUT |= (BIT5 | BIT6 | BIT7 );	//Reds
		SysTick_delay_ms(290);
		Speaker(0, 0);
		P3->OUT = 0;			//Off
		SysTick_delay_ms(100);

}
```

Songs.h

```
/*
 * Songs.h
 *
 *  Created on: Dec 3, 2020
 *      Author: 7jorchay
 */

#ifndef SONGS_H_
#define SONGS_H_

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

//Note frequencies
#define A_2    110
#define A_S2   117
#define B_2    123
#define C_3    131
#define C_S3   139
#define D_3    147
#define D_S3   156
#define E_3    165
#define F_3    175
#define F_S3   185
#define G_3    196
#define G_S3   208
#define A_3    220
#define A_S3   233
#define B_3    247
#define C_4    262
#define C_S4   277
#define D_4    294
#define D_S4   311
#define E_4    330
#define F_4    349
#define F_S4   370
#define G_4    392
#define G_S4   415
#define A_4    440
#define A_S4   466
#define B_4    494
#define C_5    523
#define C_S5   554
```

```
#define D_5     587
#define D_S5    622
#define E_5     659
#define F_5     698
#define F_S5    740
#define G_5     784
#define G_S5    831
#define A_5     880
#define A_S5    932
#define B_5     988

#define WHOLE 4
#define HALF 2
#define QUARTER 1

extern uint8_t song;

/********************* Function Prototypes *****************************
 *************************************************************************/
void Speaker(int Note, int on);
void Speaker_3MHz(int Note, int on);
void Speaker_12MHz(int Note, int on);

void Pirates_C(void);
void Despacito(void);
void Bells(int repeats);
/********************* Function Prototypes *****************************
 *************************************************************************/



#endif /* SONGS_H_ */
```

ST7735.c

```
/***************************************************
  This is a library for the Adafruit 1.8" SPI display.
  This library works with the Adafruit 1.8" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/358
  as well as Adafruit raw 1.8" TFT displayun
  ----> http://www.adafruit.com/products/618

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

// ST7735.c
// Runs on MSP432
// Low level drivers for the ST7735 160x128 LCD based off of
// the file described above.
//    16-bit color, 128 wide by 160 high LCD
// Daniel Valvano
// October 12, 2015
// Augmented 7/17/2014 to have a simple graphics facility
// Tested with LaunchPadDLL.dll simulator 9/2/2014

/* This example accompanies the book
   "Embedded Systems: Introduction to the MSP432 Microcontroller",
  ISBN: 978-1512185676, Jonathan Valvano, copyright (c) 2015

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// hardware connections
// **********ST7735 TFT and SDC*******************
// ST7735
// Backlight (pin 10) connected to +3.3 V
// MISO (pin 9) unconnected
// SCK (pin 8) connected to P9.5 (UCA3CLK)
// MOSI (pin 7) connected to P9.7 (UCA3SIMO)
// TFT_CS (pin 6) connected to P9.4 (UCA3STE)
// CARD_CS (pin 5) unconnected
// Data/Command (pin 4) connected to P9.2 (GPIO), high for data, low for command
// RESET (pin 3) connected to P9.3 (GPIO)
// VCC (pin 2) connected to +3.3 V
// Gnd (pin 1) connected to ground

// **********wide.hk ST7735R with ADXL345 accelerometer *******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - P9.5 UCA3CLK SPI clock from microcontroller to TFT or SDC
// !SDA - P9.7 UCA3SIMO SPI data from microcontroller to TFT or SDC
// DC   - P9.2 TFT data/command
// RES  - P9.3 TFT reset
// CS   - P9.4 UCA3STE TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// SDA  – (NC) I2C data for ADXL345 accelerometer
// SCL  – (NC) I2C clock for ADXL345 accelerometer
// SDO  – (NC) I2C alternate address for ADXL345 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

// **********wide.hk ST7735R with ADXL335 accelerometer *******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - P9.5 UCA3CLK SPI clock from microcontroller to TFT or SDC
// !SDA - P9.7 UCA3SIMO SPI data from microcontroller to TFT or SDC
// DC   - P9.2 TFT data/command
// RES  - P9.3 TFT reset
// CS   - P9.4 UCA3STE TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// X– (NC) analog input X-axis from ADXL335 accelerometer
// Y– (NC) analog input Y-axis from ADXL335 accelerometer
// Z– (NC) analog input Z-axis from ADXL335 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

#include <stdio.h>
#include <stdint.h>
#include "ST7735.h"
#include "msp.h"
```

```
// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
uint32_t StX=0; // position along the horizonal axis 0 to 20
uint32_t StY=0; // position along the vertical axis 0 to 15
uint16_t StTextColor = ST7735_YELLOW;

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

#define TFT_CS                  (*((volatile uint8_t *)0x40004C82))  /* Port 9 Output, bit 4 is TFT CS */
#define TFT_CS_BIT              0x10        // CS normally controlled by hardware
#define DC                      (*((volatile uint8_t *)0x40004C82))  /* Port 9 Output, bit 2 is DC */
#define DC_BIT                  0x04
#define RESET                   (*((volatile uint8_t *)0x40004C82))  /* Port 9 Output, bit 3 is RESET*/
#define RESET_BIT               0x08

#define ST7735_TFTWIDTH  128
#define ST7735_TFTHEIGHT 160

#define ST7735_NOP     0x00
```

```
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

//////////////////////////////////////////////////
//////////////////////////////////////////////////
/*
 *		Project Custom Functions
 */
//////////////////////////////////////////////////
//////////////////////////////////////////////////
//fiunction to display custom warning when an object is within 15 inches of vehicle
void displayParkingWarning()
{
   //Obstacle
   ST7735_FillScreen(ST7735_BLACK);
   ST7735_DrawChar(22, 0, 30, ST7735_RED, ST7735_BLACK, 16);
   ST7735_DrawChar(50, 55, '!', ST7735_BLACK, ST7735_RED, 5);
   ST7735_DrawChar(19, 100, 'O', ST7735_WHITE, ST7735_BLACK, 2);
   ST7735_DrawChar(30, 100, 'b', ST7735_WHITE, ST7735_BLACK, 2);
   ST7735_DrawChar(41, 100, 's', ST7735_WHITE, ST7735_BLACK, 2);
   ST7735_DrawChar(52, 100, 't', ST7735_WHITE, ST7735_BLACK, 2);
   ST7735_DrawChar(63, 100, 'a', ST7735_WHITE, ST7735_BLACK, 2);
```

```
  ST7735_DrawChar(74, 100, 'c', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(85, 100, 'l', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(96, 100, 'e', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(41, 120, 'N', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(52, 120, 'e', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(63, 120, 'a', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(74, 120, 'r', ST7735_WHITE, ST7735_BLACK, 2);
  //ST7735_FillScreen(ST7735_BLACK);
}

//function to display custom warning when cabin temp is above 100 degrees fahrenheit
void displayHiCabinTemp()
{
  ST7735_FillScreen(ST7735_BLACK);
  ST7735_DrawChar(22, 0, 30, ST7735_RED, ST7735_BLACK, 16);
  ST7735_DrawChar(50, 55, '!', ST7735_BLACK, ST7735_RED, 5);
  ST7735_DrawChar(23, 100, 'H', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(34, 100, 'i', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(48, 100, 'C', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(59, 100, 'a', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(70, 100, 'b', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(81, 100, 'i', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(92, 100, 'n', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(41, 120, 'T', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(52, 120, 'e', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(63, 120, 'm', ST7735_WHITE, ST7735_BLACK, 2);
  ST7735_DrawChar(74, 120, 'p', ST7735_WHITE, ST7735_BLACK, 2);
}


// standard ascii 5x7 font
// originally from glcdfont.c from Adafruit project
static const uint8_t Font[] = {
 0x00, 0x00, 0x00, 0x00, 0x00,
 0x3E, 0x5B, 0x4F, 0x5B, 0x3E,
 0x3E, 0x6B, 0x4F, 0x6B, 0x3E,
 0x1C, 0x3E, 0x7C, 0x3E, 0x1C,
 0x18, 0x3C, 0x7E, 0x3C, 0x18,
 0x1C, 0x57, 0x7D, 0x57, 0x1C,
 0x1C, 0x5E, 0x7F, 0x5E, 0x1C,
 0x00, 0x18, 0x3C, 0x18, 0x00,
 0xFF, 0xE7, 0xC3, 0xE7, 0xFF,
 0x00, 0x18, 0x24, 0x18, 0x00,
 0xFF, 0xE7, 0xDB, 0xE7, 0xFF,
 0x30, 0x48, 0x3A, 0x06, 0x0E,
 0x26, 0x29, 0x79, 0x29, 0x26,
 0x40, 0x7F, 0x05, 0x05, 0x07,
 0x40, 0x7F, 0x05, 0x25, 0x3F,
 0x5A, 0x3C, 0xE7, 0x3C, 0x5A,
 0x7F, 0x3E, 0x1C, 0x1C, 0x08,
 0x08, 0x1C, 0x1C, 0x3E, 0x7F,
 0x14, 0x22, 0x7F, 0x22, 0x14,
 0x5F, 0x5F, 0x00, 0x5F, 0x5F,
 0x06, 0x09, 0x7F, 0x01, 0x7F,
 0x00, 0x66, 0x89, 0x95, 0x6A,
 0x60, 0x60, 0x60, 0x60, 0x60,
 0x94, 0xA2, 0xFF, 0xA2, 0x94,
 0x08, 0x04, 0x7E, 0x04, 0x08,
 0x10, 0x20, 0x7E, 0x20, 0x10,
 0x08, 0x08, 0x2A, 0x1C, 0x08,
 0x08, 0x1C, 0x2A, 0x08, 0x08,
 0x1E, 0x10, 0x10, 0x10, 0x10,
 0x0C, 0x1E, 0x0C, 0x1E, 0x0C,
 0x30, 0x38, 0x3E, 0x38, 0x30,
 0x06, 0x0E, 0x3E, 0x0E, 0x06,
 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x5F, 0x00, 0x00,
 0x00, 0x07, 0x00, 0x07, 0x00,
 0x14, 0x7F, 0x14, 0x7F, 0x14,
 0x24, 0x2A, 0x7F, 0x2A, 0x12,
 0x23, 0x13, 0x08, 0x64, 0x62,
 0x36, 0x49, 0x56, 0x20, 0x50,
 0x00, 0x08, 0x07, 0x03, 0x00,
 0x00, 0x1C, 0x22, 0x41, 0x00,
 0x00, 0x41, 0x22, 0x1C, 0x00,
 0x2A, 0x1C, 0x7F, 0x1C, 0x2A,
 0x08, 0x08, 0x3E, 0x08, 0x08,
 0x00, 0x80, 0x70, 0x30, 0x00,
 0x08, 0x08, 0x08, 0x08, 0x08,
 0x00, 0x00, 0x60, 0x60, 0x00,
 0x20, 0x10, 0x08, 0x04, 0x02,
 0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
 0x00, 0x42, 0x7F, 0x40, 0x00, // 1
 0x72, 0x49, 0x49, 0x49, 0x46, // 2
 0x21, 0x41, 0x49, 0x4D, 0x33, // 3
 0x18, 0x14, 0x12, 0x7F, 0x10, // 4
 0x27, 0x45, 0x45, 0x45, 0x39, // 5
 0x3C, 0x4A, 0x49, 0x49, 0x31, // 6
 0x41, 0x21, 0x11, 0x09, 0x07, // 7
 0x36, 0x49, 0x49, 0x49, 0x36, // 8
 0x46, 0x49, 0x49, 0x29, 0x1E, // 9
 0x00, 0x00, 0x14, 0x00, 0x00,
 0x00, 0x40, 0x34, 0x00, 0x00,
 0x00, 0x08, 0x14, 0x22, 0x41,
 0x14, 0x14, 0x14, 0x14, 0x14,
 0x00, 0x41, 0x22, 0x14, 0x08,
 0x02, 0x01, 0x59, 0x09, 0x06,
 0x3E, 0x41, 0x5D, 0x59, 0x4E,
 0x7C, 0x12, 0x11, 0x12, 0x7C, // A
 0x7F, 0x49, 0x49, 0x49, 0x36, // B
 0x3E, 0x41, 0x41, 0x41, 0x22, // C
 0x7F, 0x41, 0x41, 0x41, 0x3E, // D
 0x7F, 0x49, 0x49, 0x49, 0x41, // E
 0x7F, 0x09, 0x09, 0x09, 0x01, // F
 0x3E, 0x41, 0x41, 0x51, 0x73, // G
 0x7F, 0x08, 0x08, 0x08, 0x7F, // H
 0x00, 0x41, 0x7F, 0x41, 0x00, // I
 0x20, 0x40, 0x41, 0x3F, 0x01, // J
 0x7F, 0x08, 0x14, 0x22, 0x41, // K
 0x7F, 0x40, 0x40, 0x40, 0x40, // L
 0x7F, 0x02, 0x1C, 0x02, 0x7F, // M
 0x7F, 0x04, 0x08, 0x10, 0x7F, // N
 0x3E, 0x41, 0x41, 0x41, 0x3E, // O
 0x7F, 0x09, 0x09, 0x09, 0x06, // P
 0x3E, 0x41, 0x51, 0x21, 0x5E, // Q
 0x7F, 0x09, 0x19, 0x29, 0x46, // R
 0x26, 0x49, 0x49, 0x49, 0x32, // S
 0x03, 0x01, 0x7F, 0x01, 0x03, // T
 0x3F, 0x40, 0x40, 0x40, 0x3F, // U
 0x1F, 0x20, 0x40, 0x20, 0x1F, // V
 0x3F, 0x40, 0x38, 0x40, 0x3F, // W
 0x63, 0x14, 0x08, 0x14, 0x63, // X
 0x03, 0x04, 0x78, 0x04, 0x03, // Y
 0x61, 0x59, 0x49, 0x4D, 0x43, // Z
 0x00, 0x7F, 0x41, 0x41, 0x41,
 0x02, 0x04, 0x08, 0x10, 0x20,
 0x00, 0x41, 0x41, 0x41, 0x7F,
 0x04, 0x02, 0x01, 0x02, 0x04,
 0x40, 0x40, 0x40, 0x40, 0x40,
 0x00, 0x03, 0x07, 0x08, 0x00,
 0x20, 0x54, 0x54, 0x78, 0x40, // a
 0x7F, 0x28, 0x44, 0x44, 0x38, // b
 0x38, 0x44, 0x44, 0x44, 0x28, // c
 0x38, 0x44, 0x44, 0x28, 0x7F, // d
 0x38, 0x54, 0x54, 0x54, 0x18, // e
 0x00, 0x08, 0x7E, 0x09, 0x02, // f
 0x18, 0xA4, 0xA4, 0x9C, 0x78, // g
 0x7F, 0x08, 0x04, 0x04, 0x78, // h
 0x00, 0x44, 0x7D, 0x40, 0x00, // i
 0x20, 0x40, 0x40, 0x3D, 0x00, // j
 0x7F, 0x10, 0x28, 0x44, 0x00, // k
 0x00, 0x41, 0x7F, 0x40, 0x00, // l
 0x7C, 0x04, 0x78, 0x04, 0x78, // m
 0x7C, 0x08, 0x04, 0x04, 0x78, // n
 0x38, 0x44, 0x44, 0x44, 0x38, // o
 0xFC, 0x18, 0x24, 0x24, 0x18, // p
 0x18, 0x24, 0x24, 0x18, 0xFC, // q
 0x7C, 0x08, 0x04, 0x04, 0x08, // r
 0x48, 0x54, 0x54, 0x54, 0x24, // s
 0x04, 0x04, 0x3F, 0x44, 0x24, // t
 0x3C, 0x40, 0x40, 0x20, 0x7C, // u
 0x1C, 0x20, 0x40, 0x20, 0x1C, // v
 0x3C, 0x40, 0x30, 0x40, 0x3C, // w
 0x44, 0x28, 0x10, 0x28, 0x44, // x
 0x4C, 0x90, 0x90, 0x90, 0x7C, // y
 0x44, 0x64, 0x54, 0x4C, 0x44, // z
 0x00, 0x08, 0x36, 0x41, 0x00,
 0x00, 0x00, 0x77, 0x00, 0x00,
 0x00, 0x41, 0x36, 0x08, 0x00,
 0x02, 0x01, 0x02, 0x04, 0x02,
 0x3C, 0x26, 0x23, 0x26, 0x3C,
 0x1E, 0xA1, 0xA1, 0x61, 0x12,
 0x3A, 0x40, 0x40, 0x20, 0x7A,
 0x38, 0x54, 0x54, 0x55, 0x59,
 0x21, 0x55, 0x55, 0x79, 0x41,
 0x21, 0x54, 0x54, 0x78, 0x41,
 0x21, 0x55, 0x54, 0x78, 0x40,
 0x20, 0x54, 0x55, 0x79, 0x40,
 0x0C, 0x1E, 0x52, 0x72, 0x12,
 0x39, 0x55, 0x55, 0x55, 0x59,
 0x39, 0x54, 0x54, 0x54, 0x59,
 0x39, 0x55, 0x54, 0x54, 0x58,
 0x00, 0x00, 0x45, 0x7C, 0x41,
 0x00, 0x02, 0x45, 0x7D, 0x42,
 0x00, 0x01, 0x45, 0x7C, 0x40,
 0xF0, 0x29, 0x24, 0x29, 0xF0,
 0xF0, 0x28, 0x25, 0x28, 0xF0,
 0x7C, 0x54, 0x55, 0x45, 0x00,
 0x20, 0x54, 0x54, 0x7C, 0x54,
 0x7C, 0x0A, 0x09, 0x7F, 0x49,
 0x32, 0x49, 0x49, 0x49, 0x32,
 0x32, 0x48, 0x48, 0x48, 0x32,
 0x32, 0x4A, 0x48, 0x48, 0x30,
 0x3A, 0x41, 0x41, 0x21, 0x7A,
 0x3A, 0x42, 0x40, 0x20, 0x78,
 0x00, 0x9D, 0xA0, 0xA0, 0x7D,
 0x39, 0x44, 0x44, 0x44, 0x39,
 0x3D, 0x40, 0x40, 0x40, 0x3D,
 0x3C, 0x24, 0xFF, 0x24, 0x24,
 0x48, 0x7E, 0x49, 0x43, 0x66,
 0x2B, 0x2F, 0xFC, 0x2F, 0x2B,
 0xFF, 0x09, 0x29, 0xF6, 0x20,
 0xC0, 0x88, 0x7E, 0x09, 0x03,
 0x20, 0x54, 0x54, 0x79, 0x41,
 0x00, 0x00, 0x44, 0x7D, 0x41,
 0x30, 0x48, 0x48, 0x4A, 0x32,
 0x38, 0x40, 0x40, 0x22, 0x7A,
 0x00, 0x7A, 0x0A, 0x0A, 0x72,
 0x7D, 0x0D, 0x19, 0x31, 0x7D,
 0x26, 0x29, 0x29, 0x2F, 0x28,
 0x26, 0x29, 0x29, 0x29, 0x26,
 0x30, 0x48, 0x4D, 0x40, 0x20,
 0x38, 0x08, 0x08, 0x08, 0x08,
 0x08, 0x08, 0x08, 0x08, 0x38,
 0x2F, 0x10, 0xC8, 0xAC, 0xBA,
 0x2F, 0x10, 0x28, 0x34, 0xFA,
 0x00, 0x00, 0x7B, 0x00, 0x00,
 0x08, 0x14, 0x2A, 0x14, 0x22,
 0x22, 0x14, 0x2A, 0x14, 0x08,
 0xAA, 0x00, 0x55, 0x00, 0xAA,
 0xAA, 0x55, 0xAA, 0x55, 0xAA,
 0x00, 0x00, 0x00, 0xFF, 0x00,
 0x10, 0x10, 0x10, 0xFF, 0x00,
 0x14, 0x14, 0x14, 0xFF, 0x00,
 0x10, 0x10, 0xFF, 0x00, 0xFF,
 0x10, 0x10, 0xF0, 0x10, 0xF0,
 0x14, 0x14, 0x14, 0xFC, 0x00,
 0x14, 0x14, 0xF7, 0x00, 0xFF,
 0x00, 0x00, 0xFF, 0x00, 0xFF,
 0x14, 0x14, 0xF4, 0x04, 0xFC,
 0x14, 0x14, 0x17, 0x10, 0x1F,
 0x10, 0x10, 0x1F, 0x10, 0x1F,
 0x14, 0x14, 0x14, 0x1F, 0x00,
 0x10, 0x10, 0x10, 0xF0, 0x00,
 0x00, 0x00, 0x00, 0x1F, 0x10,
 0x10, 0x10, 0x10, 0x1F, 0x10,
 0x10, 0x10, 0x10, 0xF0, 0x10,
 0x00, 0x00, 0x00, 0xFF, 0x10,
 0x10, 0x10, 0x10, 0x10, 0x10,
 0x10, 0x10, 0x10, 0xFF, 0x10,
 0x00, 0x00, 0x00, 0xFF, 0x14,
 0x00, 0x00, 0xFF, 0x00, 0xFF,
 0x00, 0x00, 0x1F, 0x10, 0x17,
 0x00, 0x00, 0xFC, 0x04, 0xF4,
 0x14, 0x14, 0x17, 0x10, 0x17,
 0x14, 0x14, 0xF4, 0x04, 0xF4,
 0x00, 0x00, 0xFF, 0x00, 0xF7,
 0x14, 0x14, 0x14, 0x14, 0x14,
 0x14, 0x14, 0xF7, 0x00, 0xF7,
 0x14, 0x14, 0x14, 0x17, 0x14,
 0x10, 0x10, 0x1F, 0x10, 0x1F,
 0x14, 0x14, 0x14, 0xF4, 0x14,
 0x10, 0x10, 0xF0, 0x10, 0xF0,
 0x00, 0x00, 0x1F, 0x10, 0x1F,
 0x00, 0x00, 0x00, 0x1F, 0x14,
 0x00, 0x00, 0x00, 0xFC, 0x14,
 0x00, 0x00, 0xF0, 0x10, 0xF0,
 0x10, 0x10, 0xFF, 0x10, 0xFF,
 0x14, 0x14, 0x14, 0xFF, 0x14,
 0x10, 0x10, 0x10, 0x1F, 0x00,
 0x00, 0x00, 0x00, 0xF0, 0x10,
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
 0xFF, 0xFF, 0xFF, 0x00, 0x00,
 0x00, 0x00, 0x00, 0xFF, 0xFF,
 0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
  0x38, 0x44, 0x44, 0x38, 0x44,
  0x7C, 0x2A, 0x2A, 0x3E, 0x14,
  0x7E, 0x02, 0x02, 0x06, 0x06,
  0x02, 0x7E, 0x02, 0x7E, 0x02,
  0x63, 0x55, 0x49, 0x41, 0x63,
  0x38, 0x44, 0x44, 0x3C, 0x04,
  0x40, 0x7E, 0x20, 0x1E, 0x20,
  0x06, 0x02, 0x7E, 0x02, 0x02,
  0x99, 0xA5, 0xE7, 0xA5, 0x99,
  0x1C, 0x2A, 0x49, 0x2A, 0x1C,
  0x4C, 0x72, 0x01, 0x72, 0x4C,
  0x30, 0x4A, 0x4D, 0x4D, 0x30,
  0x30, 0x48, 0x78, 0x48, 0x30,
  0xBC, 0x62, 0x5A, 0x46, 0x3D,
  0x3E, 0x49, 0x49, 0x49, 0x00,
  0x7E, 0x01, 0x01, 0x01, 0x7E,
  0x2A, 0x2A, 0x2A, 0x2A, 0x2A,
  0x44, 0x44, 0x5F, 0x44, 0x44,
  0x40, 0x51, 0x4A, 0x44, 0x40,
  0x40, 0x44, 0x4A, 0x51, 0x40,
  0x00, 0x00, 0xFF, 0x01, 0x03,
  0xE0, 0x80, 0xFF, 0x00, 0x00,
  0x08, 0x08, 0x6B, 0x6B, 0x08,
  0x36, 0x12, 0x36, 0x24, 0x36,
  0x06, 0x0F, 0x09, 0x0F, 0x06,
  0x00, 0x00, 0x18, 0x18, 0x00,
  0x00, 0x00, 0x10, 0x10, 0x00,
  0x30, 0x40, 0xFF, 0x01, 0x01,
  0x00, 0x1F, 0x01, 0x01, 0x1E,
  0x00, 0x19, 0x1D, 0x17, 0x12,
  0x00, 0x3C, 0x3C, 0x3C, 0x3C,
  0x00, 0x00, 0x00, 0x00, 0x00,
};


static uint8_t ColStart, RowStart; // some displays need this changed
static uint8_t Rotation;          // 0 to 3
static enum initRFlags TabColor;
static int16_t _width = ST7735_TFTWIDTH;   // this could probably be a constant, except it is used in Adafruit_GFX and depends on image rotation
static int16_t _height = ST7735_TFTHEIGHT;


// The Data/Command pin must be valid when the eighth bit is
// sent.  The eUSCI module has no hardware input or output
// FIFOs, so this implementation is much simpler than it was
// for the Tiva LaunchPads.
// All operations wait until all data has been sent,
// configure the Data/Command pin, queue the message, and
// return the reply once it comes in.

// This is a helper function that sends an 8-bit command to the LCD.
// Inputs: c  8-bit code to transmit
// Outputs: 8-bit reply
// Assumes: UCA3 and Port 9 have already been initialized and enabled
uint8_t static writecommand(uint8_t c) {
  while((EUSCI_A3->IFG&0x0002)==0x0000){};    // wait until EUSCI_A3->TXBUF empty
  DC &= ~DC_BIT;
  EUSCI_A3->TXBUF = c;                      // command out
  while((EUSCI_A3->IFG&0x0001)==0x0000){};    // wait until EUSCI_A3->RXBUF full
  return EUSCI_A3->RXBUF;                   // return the response
}


// This is a helper function that sends a piece of 8-bit data to the LCD.
// Inputs: c  8-bit data to transmit
// Outputs: 8-bit reply
// Assumes: UCA3 and Port 9 have already been initialized and enabled
uint8_t static writedata(uint8_t c) {
  while((EUSCI_A3->IFG&0x0002)==0x0000){};    // wait until EUSCI_A3->TXBUF empty
  DC |= DC_BIT;
  EUSCI_A3->TXBUF = c;                     // data out
  while((EUSCI_A3->IFG&0x0001)==0x0000){};    // wait until EUSCI_A3->RXBUF full
  return EUSCI_A3->RXBUF;                   // return the response
}


// delay function for testing
// which delays about 8.1*ulCount cycles
#ifdef __TI_COMPILER_VERSION__
  //Code Composer Studio Code
  void parrotdelay(unsigned long ulCount){
  __asm (  "pdloop:  subs    r0, #1\n"
      "    bne    pdloop\n");
}

#else
  //Keil uVision Code
  __asm void
  parrotdelay(unsigned long ulCount)
  {
    subs    r0, #1
    bne     parrotdelay
    bx      lr
  }

#endif
// Subroutine to wait 1 msec
// Inputs: n  number of 1 msec to wait
// Outputs: None
// Notes: ...
void Delay1ms(uint32_t n){
  while(n){
    parrotdelay(5901);                // 1 msec, tuned at 48 MHz
    n--;
  }
}


// Rather than a bazillion writecommand() and writedata() calls, screen
// initialization commands and arguments are organized in these tables
// stored in ROM.  The table may look bulky, but that's mostly the
// formatting -- storage-wise this is hundreds of bytes more compact
// than the equivalent code.  Companion function follows.
#define DELAY 0x80
static const uint8_t
  Bcmd[] = {                 // Initialization commands for 7735B screens
    18,                      // 18 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, no args, w/delay
      50,                    //     50 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, no args, w/delay
      255,                   //     255 = 500 ms delay
    ST7735_COLMOD , 1+DELAY,  //  3: Set color mode, 1 arg + delay:
      0x05,                  //     16-bit color
      10,                    //     10 ms delay
    ST7735_FRMCTR1, 3+DELAY,  //  4: Frame rate control, 3 args + delay:
      0x00,                  //     fastest refresh
      0x06,                  //     6 lines front porch
      0x03,                  //     3 lines back porch
      10,                    //     10 ms delay
    ST7735_MADCTL , 1      ,  //  5: Memory access ctrl (directions), 1 arg:
      0x08,                  //     Row addr/col addr, bottom to top refresh
    ST7735_DISSET5, 2      ,  //  6: Display settings #5, 2 args, no delay:
      0x15,                  //     1 clk cycle nonoverlap, 2 cycle gate
                           //     rise, 3 cycle osc equalize
      0x02,                  //     Fix on VTL
    ST7735_INVCTR , 1      ,  //  7: Display inversion control, 1 arg:
      0x0,                   //     Line inversion
    ST7735_PWCTR1 , 2+DELAY,  //  8: Power control, 2 args + delay:
      0x02,                  //     GVDD = 4.7V
      0x70,                  //     1.0uA
      10,                    //     10 ms delay
    ST7735_PWCTR2 , 1      ,  //  9: Power control, 1 arg, no delay:
      0x05,                  //     VGH = 14.7V, VGL = -7.35V
    ST7735_PWCTR3 , 2      ,  // 10: Power control, 2 args, no delay:
      0x01,                  //     Opamp current small
      0x02,                  //     Boost frequency
    ST7735_VMCTR1 , 2+DELAY,  // 11: Power control, 2 args + delay:
      0x3C,                  //     VCOMH = 4V
      0x38,                  //     VCOML = -1.1V
      10,                    //     10 ms delay
    ST7735_PWCTR6 , 2      ,  // 12: Power control, 2 args, no delay:
      0x11, 0x15,
    ST7735_GMCTRP1,16      ,  // 13: Magical unicorn dust, 16 args, no delay:
      0x09, 0x16, 0x09, 0x20, //     (seriously though, not sure what
      0x21, 0x1B, 0x13, 0x19, //      these config values represent)
      0x17, 0x15, 0x1E, 0x2B,
      0x04, 0x05, 0x02, 0x0E,
    ST7735_GMCTRN1,16+DELAY,  // 14: Sparkles and rainbows, 16 args + delay:
      0x0B, 0x14, 0x08, 0x1E, //     (ditto)
      0x22, 0x1D, 0x18, 0x1E,
      0x1B, 0x1A, 0x24, 0x2B,
      0x06, 0x06, 0x02, 0x0F,
      10,                    //     10 ms delay
    ST7735_CASET  , 4      ,  // 15: Column addr set, 4 args, no delay:
      0x00, 0x02,            //     XSTART = 2
      0x00, 0x81,            //     XEND = 129
    ST7735_RASET  , 4      ,  // 16: Row addr set, 4 args, no delay:
      0x00, 0x02,            //     XSTART = 1
      0x00, 0x81,            //     XEND = 160
    ST7735_NORON  ,   DELAY,  // 17: Normal display on, no args, w/delay
      10,                    //     10 ms delay
    ST7735_DISPON ,   DELAY,  // 18: Main screen turn on, no args, w/delay
      255 };                 //     255 = 500 ms delay
static const uint8_t
  Rcmd1[] = {                // Init for 7735R, part 1 (red or green tab)
    15,                       // 15 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, 0 args, w/delay
      150,                   //     150 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, 0 args, w/delay
      255,                   //     500 ms delay
    ST7735_FRMCTR1, 3      ,  //  3: Frame rate ctrl - normal mode, 3 args:
      0x01, 0x2C, 0x2D,      //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR2, 3      ,  //  4: Frame rate control - idle mode, 3 args:
      0x01, 0x2C, 0x2D,      //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR3, 6      ,  //  5: Frame rate ctrl - partial mode, 6 args:
      0x01, 0x2C, 0x2D,      //     Dot inversion mode
      0x01, 0x2C, 0x2D,      //     Line inversion mode
    ST7735_INVCTR , 1      ,  //  6: Display inversion ctrl, 1 arg, no delay:
      0x07,                  //     No inversion
    ST7735_PWCTR1 , 3      ,  //  7: Power control, 3 args, no delay:
      0xA2,
      0x02,                  //     -4.6V
      0x84,                  //     AUTO mode
    ST7735_PWCTR2 , 1      ,  //  8: Power control, 1 arg, no delay:
      0xC5,                  //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
    ST7735_PWCTR3 , 2      ,  //  9: Power control, 2 args, no delay:
      0x0A,                  //     Opamp current small
      0x00,                  //     Boost frequency
    ST7735_PWCTR4 , 2      ,  // 10: Power control, 2 args, no delay:
      0x8A,                  //     BCLK/2, Opamp current small & Medium low
      0x2A,
    ST7735_PWCTR5 , 2      ,  // 11: Power control, 2 args, no delay:
      0x8A, 0xEE,
    ST7735_VMCTR1 , 1      ,  // 12: Power control, 1 arg, no delay:
      0x0E,
    ST7735_INVOFF , 0      ,  // 13: Don't invert display, no args, no delay
    ST7735_MADCTL , 1      ,  // 14: Memory access control (directions), 1 arg:
      0xC8,                  //     row addr/col addr, bottom to top refresh
    ST7735_COLMOD , 1      ,  // 15: set color mode, 1 arg, no delay:
      0x05 };                //     16-bit color
static const uint8_t
  Rcmd2green[] = {          // Init for 7735R, part 2 (green tab only)
    2,                      //  2 commands in list:
    ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 0
      0x00, 0x7F+0x02,        //     XEND = 127
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
      0x00, 0x01,             //     XSTART = 0
      0x00, 0x9F+0x01 };      //     XEND = 159
static const uint8_t
  Rcmd2red[] = {            // Init for 7735R, part 2 (red tab only)
    2,                      //  2 commands in list:
    ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x7F,             //     XEND = 127
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x9F };           //     XEND = 159
static const uint8_t
  Rcmd3[] = {               // Init for 7735R, part 3 (red or green tab)
    4,                      //  4 commands in list:
    ST7735_GMCTRP1, 16      , //  1: Magical unicorn dust, 16 args, no delay:
      0x02, 0x1c, 0x07, 0x12,
      0x37, 0x32, 0x29, 0x2d,
      0x29, 0x25, 0x2B, 0x39,
      0x00, 0x01, 0x03, 0x10,
    ST7735_GMCTRN1, 16      , //  2: Sparkles and rainbows, 16 args, no delay:
      0x03, 0x1d, 0x07, 0x06,
      0x2E, 0x2C, 0x29, 0x2D,
      0x2E, 0x2E, 0x37, 0x3F,
      0x00, 0x00, 0x02, 0x10,
    ST7735_NORON  ,    DELAY, //  3: Normal display on, no args, w/delay
      10,                     //     10 ms delay
    ST7735_DISPON ,    DELAY, //  4: Main screen turn on, no args w/delay
      100 };                  //     100 ms delay


// Companion code to the above tables.  Reads and issues
// a series of LCD commands stored in ROM byte array.
void static commandList(const uint8_t *addr) {

  uint8_t numCommands, numArgs;
  uint16_t ms;

  numCommands = *(addr++);               // Number of commands to follow
  while(numCommands--) {                 // For each command...
```

```
    writecommand(*(addr++));          //   Read, issue command
    numArgs  = *(addr++);             //   Number of args to follow
    ms       = numArgs & DELAY;       //   If hibit set, delay follows args
    numArgs &= ~DELAY;                //   Mask out delay bit
    while(numArgs--) {                //   For each argument...
      writedata(*(addr++));           //     Read, issue argument
    }

    if(ms) {
      ms = *(addr++);           // Read post-command delay time (ms)
      if(ms == 255) ms = 500;    // If 255, delay for 500 ms
      Delay1ms(ms);
    }
  }
}


// Initialization code common to both 'B' and 'R' type displays
void static commonInit(const uint8_t *cmdList) {
  ColStart  = RowStart = 0; // May be overridden in init func

  // toggle RST low to reset; CS low so it'll listen to us
  // UCA3STE is temporarily used as GPIO
  P9->SEL0 &= ~0x1C;
  P9->SEL1 &= ~0x1C;                    // configure P9.2 (D/C), P9.3 (Reset), and P9.4 (TFT_CS) as GPIO
  P9->DIR |= 0x1C;                     // make P9.2 (D/C), P9.3 (Reset), and P9.4 (TFT_CS) out
  TFT_CS &= ~TFT_CS_BIT;
  RESET |= RESET_BIT;
  Delay1ms(500);
  RESET &= ~RESET_BIT;
  Delay1ms(500);
  RESET |= RESET_BIT;
  Delay1ms(500);

  // initialize eUSCI
  EUSCI_A3->CTLW0 = 0x0001;                 // hold the eUSCI module in reset mode
  // configure EUSCI_A3->CTLW0 for:
  // bit15      EUSCI_A_CTLW0_CKPH = 1; data shifts in on first edge, out on following edge
  // bit14      EUSCI_A_CTLW0_CKPL = 0; clock is low when inactive
  // bit13      EUSCI_B_CTLW0_MSB = 1; MSB first
  // bit12      EUSCI_B_CTLW0_SEVENBIT = 0; 8-bit data
  // bit11      EUSCI_B_CTLW0_MST = 1; master mode
  // bits10-9   UCMODEx = 2; UCSTE active low
  // bit8       EUSCI_B_CTLW0_SYNC = 1; synchronous mode
  // bits7-6    UCSSELx = 2; eUSCI clock SMCLK
  // bits5-2    reserved
  // bit1       EUSCI_B_CTLW0_STEM = 1; UCSTE pin enables slave
  // bit0       EUSCI_A_CTLW0_SWRST = 1; reset enabled
  EUSCI_A3->CTLW0 = 0xAD83;
  // set the baud rate for the eUSCI which gets its clock from SMCLK
  // Clock_Init48MHz() from ClockSystem.c sets SMCLK = HFXTCLK/4 = 12 MHz
  // if the SMCLK is set to 12 MHz, divide by 3 for 4 MHz baud clock
  EUSCI_A3->BRW = 3;
  // modulation is not used in SPI mode, so clear EUSCI_A3->MCTLW
  EUSCI_A3->MCTLW = 0;
  P9->SEL0 |= 0xB0;
  P9->SEL1 &= ~0xB0;                    // configure P9.7, P9.5, and P9.4 as primary module function
  P9->SEL0 &= ~0x0C;
  P9->SEL1 &= ~0x0C;                    // configure P9.3 and P9.2 as GPIO (Reset and D/C pins)
  P9->DIR |= 0x0C;                     // make P9.3 and P9.2 out (Reset and D/C pins)
  EUSCI_A3->CTLW0 &= ~0x0001;                // enable eUSCI module
  EUSCI_A3->IE &= ~0x0003;                  // disable interrupts

  if(cmdList) commandList(cmdList);
```

```
}

//------------ST7735_InitB------------
// Initialization for ST7735B screens.
// Input: none
// Output: none
void ST7735_InitB(void) {
  commonInit(Bcmd);
  ST7735_SetCursor(0,0);
  StTextColor = ST7735_YELLOW;
  ST7735_FillScreen(0);                 // set screen to black
}

//------------ST7735_InitR------------
// Initialization for ST7735R screens (green or red tabs).
// Input: option one of the enumerated options depending on tabs
// Output: none
void ST7735_InitR(enum initRFlags option) {
  commonInit(Rcmd1);
  if(option == INITR_GREENTAB) {
    commandList(Rcmd2green);
    ColStart = 2;
    RowStart = 1;
  } else {
    // colstart, rowstart left at default '0' values
    commandList(Rcmd2red);
  }
  commandList(Rcmd3);

  // if black, change MADCTL color filter
  if (option == INITR_BLACKTAB) {
    writecommand(ST7735_MADCTL);
    writedata(0xC0);
  }
  TabColor = option;
  ST7735_SetCursor(0,0);
  StTextColor = ST7735_YELLOW;
  ST7735_FillScreen(0);                 // set screen to black
}


// Set the region of the screen RAM to be modified
// Pixel colors are sent left to right, top to bottom
// (same as Font table is encoded; different from regular bitmap)
// Requires 11 bytes of transmission
void static setAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {

  writecommand(ST7735_CASET); // Column addr set
  writedata(0x00);
  writedata(x0+ColStart);     // XSTART
  writedata(0x00);
  writedata(x1+ColStart);     // XEND

  writecommand(ST7735_RASET); // Row addr set
  writedata(0x00);
  writedata(y0+RowStart);     // YSTART
  writedata(0x00);
  writedata(y1+RowStart);     // YEND

  writecommand(ST7735_RAMWR); // write to RAM
}


// Send two bytes of data, most significant byte first
// Requires 2 bytes of transmission
void static pushColor(uint16_t color) {
  writedata((uint8_t)(color >> 8));
  writedata((uint8_t)color);
}


//------------ST7735_DrawPixel------------
// Color the pixel at the given coordinates with the given color.
// Requires 13 bytes of transmission
// Input: x     horizontal position of the pixel, columns from the left edge
//              must be less than 128
//              0 is on the left, 126 is near the right
//        y     vertical position of the pixel, rows from the top edge
//              must be less than 160
//              159 is near the wires, 0 is the side opposite the wires
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawPixel(int16_t x, int16_t y, uint16_t color) {

  if((x < 0) || (x >= _width) || (y < 0) || (y >= _height)) return;

//  setAddrWindow(x,y,x+1,y+1); // original code, bug???
  setAddrWindow(x,y,x,y);

  pushColor(color);
}


//------------ST7735_DrawFastVLine------------
// Draw a vertical line at the given coordinates with the given height and color.
// A vertical line is parallel to the longer side of the rectangular display
// Requires (11 + 2*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
//        y     vertical position of the start of the line, rows from the top edge
//        h     vertical height of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color) {
  uint8_t hi = color >> 8, lo = color;

  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((y+h-1) >= _height) h = _height-y;
  setAddrWindow(x, y, x, y+h-1);

  while (h--) {
    writedata(hi);
    writedata(lo);
  }
}


//------------ST7735_DrawFastHLine------------
// Draw a horizontal line at the given coordinates with the given width and color.
// A horizontal line is parallel to the shorter side of the rectangular display
// Requires (11 + 2*w) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
//        y     vertical position of the start of the line, rows from the top edge
//        w     horizontal width of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color) {
  uint8_t hi = color >> 8, lo = color;
  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((x+w-1) >= _width)  w = _width-x;
  setAddrWindow(x, y, x+w-1, y);

  while (w--) {
    writedata(hi);
    writedata(lo);
  }
}


//------------ST7735_FillScreen------------
// Fill the screen with the given color.
// Requires 40,971 bytes of transmission
// Input: color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillScreen(uint16_t color) {
  ST7735_FillRect(0, 0, _width, _height, color);  // original
//  screen is actually 129 by 161 pixels, x 0 to 128, y goes from 0 to 160
}


//------------ST7735_FillRect------------
// Draw a filled rectangle at the given coordinates with the given width, height, and color.
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the top left corner of the rectangle, columns from the left edge
//        y     vertical position of the top left corner of the rectangle, rows from the top edge
//        w     horizontal width of the rectangle
//        h     vertical height of the rectangle
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color) {
  uint8_t hi = color >> 8, lo = color;

  // rudimentary clipping (drawChar w/big text requires this)
  if((x >= _width) || (y >= _height)) return;
  if((x + w - 1) >= _width)  w = _width  - x;
  if((y + h - 1) >= _height) h = _height - y;

  setAddrWindow(x, y, x+w-1, y+h-1);

  for(y=h; y>0; y--) {
    for(x=w; x>0; x--) {
      writedata(hi);
      writedata(lo);
    }
  }
}


//------------ST7735_Color565------------
// Pass 8-bit (each) R,G,B and get back 16-bit packed color.
// Input: r red value
//        g green value
//        b blue value
// Output: 16-bit color
uint16_t ST7735_Color565(uint8_t r, uint8_t g, uint8_t b) {
  return ((b & 0xF8) << 8) | ((g & 0xFC) << 3) | (r >> 3);
}


//------------ST7735_SwapColor------------
```

```
// Swaps the red and blue values of the given 16-bit packed color;
// green is unchanged.
// Input: x 16-bit color in format B, G, R
// Output: 16-bit color in format R, G, B
uint16_t ST7735_SwapColor(uint16_t x) {
  return (x << 11) | (x & 0x07E0) | (x >> 11);
}


//------------ST7735_DrawBitmap------------
// Displays a 16-bit color BMP image.  A bitmap file that is created
// by a PC image processing program has a header and may be padded
// with dummy columns so the data have four byte alignment.  This
// function assumes that all of that has been stripped out, and the
// array image[] has one 16-bit halfword for each pixel to be
// displayed on the screen (encoded in reverse order, which is
// standard for bitmap files).  An array can be created in this
// format from a 24-bit-per-pixel .bmp file using the associated
// converter program.
// (x,y) is the screen location of the lower left corner of BMP image
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the bottom left corner of the image, columns from the left edge
//        y     vertical position of the bottom left corner of the image, rows from the top edge
//        image pointer to a 16-bit color BMP image
//        w     number of pixels wide
//        h     number of pixels tall
// Output: none
// Must be less than or equal to 128 pixels wide by 160 pixels high
void ST7735_DrawBitmap(int16_t x, int16_t y, const uint16_t *image, int16_t w, int16_t h){
  int16_t skipC = 0;                      // non-zero if columns need to be skipped due to clipping
  int16_t originalWidth = w;              // save this value; even if not all columns fit on the screen, the image is still this width in ROM
  int i = w*(h - 1);

  if((x >= _width) || ((y - h + 1) >= _height) || ((x + w) <= 0) || (y < 0)){
    return;                               // image is totally off the screen, do nothing
  }
  if((w > _width) || (h > _height)){    // image is too wide for the screen, do nothing
    //***This isn't necessarily a fatal error, but it makes the
    //following logic much more complicated, since you can have
    //an image that exceeds multiple boundaries and needs to be
    //clipped on more than one side.
    return;
  }
  if((x + w - 1) >= _width){            // image exceeds right of screen
    skipC = (x + w) - _width;           // skip cut off columns
    w = _width - x;
  }
  if((y - h + 1) < 0){                  // image exceeds top of screen
    i = i - (h - y - 1)*originalWidth;  // skip the last cut off rows
    h = y + 1;
  }
  if(x < 0){                            // image exceeds left of screen
    w = w + x;
    skipC = -1*x;                       // skip cut off columns
    i = i - x;                          // skip the first cut off columns
    x = 0;
  }
  if(y >= _height){                     // image exceeds bottom of screen
    h = h - (y - _height + 1);
    y = _height - 1;
  }

  setAddrWindow(x, y-h+1, x+w-1, y);
  for(y=0; y<h; y=y+1){
    for(x=0; x<w; x=x+1){
                                    // send the top 8 bits
      writedata((uint8_t)(image[i] >> 8));
                                    // send the bottom 8 bits
      writedata((uint8_t)image[i]);
      i = i + 1;                        // go to the next pixel
    }
    i = i + skipC;
    i = i - 2*originalWidth;
  }
}


//------------ST7735_DrawCharS------------
// Simple character draw function.  This is the same function from
// Adafruit_GFX.c but adapted for this processor.  However, each call
// to ST7735_DrawPixel() calls setAddrWindow(), which needs to send
// many extra data and commands.  If the background color is the same
// as the text color, no background will be printed, and text can be
// drawn right over existing images without covering them with a box.
// Requires (11 + 2*size*size)*6*8 bytes of transmission (image fully on screen; textcolor != bgColor)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawCharS(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t size){
  uint8_t line; // vertical column of pixels of character in font
  int32_t i, j;
  if((x >= _width)            || // Clip right
     (y >= _height)           || // Clip bottom
     ((x + 5 * size - 1) < 0) || // Clip left
     ((y + 8 * size - 1) < 0))   // Clip top
    return;

  for (i=0; i<6; i++ ) {
    if (i == 5)
      line = 0x0;
    else
      line = Font[(c*5)+i];
    for (j = 0; j<8; j++) {
      if (line & 0x1) {
        if (size == 1) // default size
          ST7735_DrawPixel(x+i, y+j, textColor);
        else {  // big size
          ST7735_FillRect(x+(i*size), y+(j*size), size, size, textColor);
        }
      } else if (bgColor != textColor) {
        if (size == 1) // default size
          ST7735_DrawPixel(x+i, y+j, bgColor);
        else {  // big size
          ST7735_FillRect(x+i*size, y+j*size, size, size, bgColor);
        }
      }
      line >>= 1;
    }
  }
}

void Print_RPMs_toLCD(int RPMs)
{
    int hundredDigit = RPMs / 100;
    int tenDigit = (RPMs % 100) / 10;
    int oneDigit = (RPMs % 100) % 10;
    char hundredChar = hundredDigit + '0';
    char tenChar = tenDigit + '0';
    char oneChar = oneDigit + '0';

    ST7735_DrawChar(22, 40, '0', ST7735_WHITE, ST7735_BLACK, 5);
    ST7735_DrawChar(51, 40, '0', ST7735_WHITE, ST7735_BLACK, 5);
    ST7735_DrawChar(80, 40, '0', ST7735_WHITE, ST7735_BLACK, 5);
    ST7735_DrawChar(33, 90, 'R', ST7735_WHITE, ST7735_BLACK, 3);
    ST7735_DrawChar(49, 90, 'P', ST7735_WHITE, ST7735_BLACK, 3);
    ST7735_DrawChar(65, 90, 'M', ST7735_WHITE, ST7735_BLACK, 3);
    ST7735_DrawChar(81, 90, 's', ST7735_WHITE, ST7735_BLACK, 3);

    ST7735_DrawChar(22, 40, hundredChar, ST7735_WHITE, ST7735_BLACK, 5);
    ST7735_DrawChar(51, 40, tenChar, ST7735_WHITE, ST7735_BLACK, 5);
    ST7735_DrawChar(80, 40, oneChar, ST7735_WHITE, ST7735_BLACK, 5);
}
//------------ST7735_DrawChar------------
// Advanced character draw function.  This is similar to the function
// from Adafruit_GFX.c but adapted for this processor.  However, this
// function only uses one call to setAddrWindow(), which allows it to
// run at least twice as fast.
// Requires (11 + size*size*6*8) bytes of transmission (assuming image fully on screen)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawChar(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t size){
  uint8_t line; // horizontal row of pixels of character
  int32_t col, row, i, j;// loop indices
  if(((x + 5*size - 1) >= _width)  || // Clip right
     ((y + 8*size - 1) >= _height) || // Clip bottom
     ((x + 5*size - 1) < 0)        || // Clip left
     ((y + 8*size - 1) < 0)){         // Clip top
    return;
  }

  setAddrWindow(x, y, x+6*size-1, y+8*size-1);

  line = 0x01;        // print the top row first
  // print the rows, starting at the top
  for(row=0; row<8; row=row+1){
    for(i=0; i<size; i=i+1){
      // print the columns, starting on the left
      for(col=0; col<5; col=col+1){
        if(Font[(c*5)+col]&line){
          // bit is set in Font, print pixel(s) in text color
          for(j=0; j<size; j=j+1){
           pushColor(textColor);
          }
        } else{
          // bit is cleared in Font, print pixel(s) in background color
          for(j=0; j<size; j=j+1){
           pushColor(bgColor);
          }
        }
      }
      // print blank column(s) to the right of character
      for(j=0; j<size; j=j+1){
    pushColor(bgColor);
   }
  }
  line = line<<1;   // move up to the next row
 }
}

//------------ST7735_DrawString------------
// String draw function.
// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
// Input: x         columns from the left edge (0 to 20)
//        y         rows from the top edge (0 to 15)
//        pt        pointer to a null terminated string to be printed
//        textColor 16-bit color of the characters
// bgColor is Black and size is 1
// Output: number of characters printed
uint32_t ST7735_DrawString(uint16_t x, uint16_t y, char *pt, int16_t textColor){
 uint32_t count = 0;
 if(y>15) return 0;
 while(*pt){
                      //ST7735_DrawCharS(x*6, y*10, *pt, textColor, ST7735_BLACK, 1);
                      ST7735_DrawCharS(x * 12, y * 10, *pt, textColor, ST7735_BLACK, 2);
  pt++;
  x = x+1;
//    if(x>20) return count;  // number of characters printed
//    count++;
 }
 return count;  // number of characters printed
}

//-----------------------fillmessage-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
char Message[12];
uint32_t Messageindex;

void fillmessage(uint32_t n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
 if(n >= 10){
  fillmessage(n/10);
  n = n%10;
 }
 Message[Messageindex] = (n+'0'); /* n is between 0 and 9 */
 if(Messageindex<11)Messageindex++;
}

//********ST7735_SetCursor*****************
// Move the cursor to the desired X- and Y-position.  The
// next character will be printed here.  X=0 is the leftmost
// column.  Y=0 is the top row.
// inputs: newX  new X-position of the cursor (0<=newX<=20)
//         newY  new Y-position of the cursor (0<=newY<=15)
// outputs: none
void ST7735_SetCursor(uint32_t newX, uint32_t newY){
 if((newX > 20) || (newY > 15)){       // bad input
  return;                        // do nothing
 }
 StX = newX;
 StY = newY;
}
```

```
//----------------------ST7735_OutUDec----------------------
// Output a 32-bit number in unsigned decimal format
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void ST7735_OutUDec(uint32_t n){
  Messageindex = 0;
  fillmessage(n);
  Message[Messageindex] = 0; // terminate
  ST7735_DrawString(StX,StY,Message,StTextColor);
  StX = StX+Messageindex;
  if(StX>20){
    StX = 20;
    ST7735_DrawCharS(StX*6,StY*10,'*',ST7735_RED,ST7735_BLACK, 1);
  }
}


#define MADCTL_MY  0x80
#define MADCTL_MX  0x40
#define MADCTL_MV  0x20
#define MADCTL_ML  0x10
#define MADCTL_RGB 0x00
#define MADCTL_BGR 0x08
#define MADCTL_MH  0x04

//------------ST7735_SetRotation------------
// Change the image rotation.
// Requires 2 bytes of transmission
// Input: m new rotation value (0 to 3)
// Output: none
void ST7735_SetRotation(uint8_t m) {

  writecommand(ST7735_MADCTL);
  Rotation = m % 4; // can't be higher than 3
  switch (Rotation) {
   case 0:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
     _height = ST7735_TFTHEIGHT;
     break;
   case 1:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_BGR);
     }
     _width  = ST7735_TFTHEIGHT;
     _height = ST7735_TFTWIDTH;
     break;
  case 2:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_RGB);
     } else {
       writedata(MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
     _height = ST7735_TFTHEIGHT;
     break;
   case 3:
     if (TabColor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_BGR);
     }
     _width  = ST7735_TFTHEIGHT;
     _height = ST7735_TFTWIDTH;
     break;
  }
}

//------------ST7735_InvertDisplay------------
// Send the command to invert all of the colors.
// Requires 1 byte of transmission
// Input: i 0 to disable inversion; non-zero to enable inversion
// Output: none
void ST7735_InvertDisplay(int i) {
  if(i){
     writecommand(ST7735_INVON);
  } else{
     writecommand(ST7735_INVOFF);
  }
}
// graphics routines
// y coordinates 0 to 31 used for labels and messages
// y coordinates 32 to 159  128 pixels high
// x coordinates 0 to 127   128 pixels wide

int32_t Ymax,Ymin,X;       // X goes from 0 to 127
int32_t Yrange; //YrangeDiv2;

// *************** ST7735_PlotClear ********************
// Clear the graphics buffer, set X coordinate to 0
// This routine clears the display
// Inputs: ymin and ymax are range of the plot
// Outputs: none
void ST7735_PlotClear(int32_t ymin, int32_t ymax){
  ST7735_FillRect(0, 32, 128, 128, ST7735_Color565(228,228,228)); // light grey
  if(ymax>ymin){
     Ymax = ymax;
     Ymin = ymin;
     Yrange = ymax-ymin;
  } else{
     Ymax = ymin;
     Ymin = ymax;
     Yrange = ymax-ymin;
  }
  //YrangeDiv2 = Yrange/2;
  X = 0;
}

// *************** ST7735_PlotPoint ********************
// Used in the voltage versus time plot, plot one point at y
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotPoint(int32_t y){int32_t j;
  if(y<Ymin) y=Ymin;
  if(y>Ymax) y=Ymax;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=Ymax maps to j=32
  // y=Ymin maps to j=159
  j = 32+(127*(Ymax-y))/Yrange;
  if(j<32) j = 32;
  if(j>159) j = 159;
  ST7735_DrawPixel(X,   j,   ST7735_BLUE);
  ST7735_DrawPixel(X+1, j,   ST7735_BLUE);
  ST7735_DrawPixel(X,   j+1, ST7735_BLUE);
  ST7735_DrawPixel(X+1, j+1, ST7735_BLUE);
}

// *************** ST7735_PlotLine ********************
// Used in the voltage versus time plot, plot line to new point
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
int32_t lastj=0;
void ST7735_PlotLine(int32_t y){int32_t i,j;
  if(y<Ymin) y=Ymin;
  if(y>Ymax) y=Ymax;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=Ymax maps to j=32
  // y=Ymin maps to j=159
  j = 32+(127*(Ymax-y))/Yrange;
  if(j < 32) j = 32;
  if(j > 159) j = 159;
  if(lastj < 32) lastj = j;
  if(lastj > 159) lastj = j;
  if(lastj < j){
    for(i = lastj+1; i<=j ; i++){
      ST7735_DrawPixel(X,   i,   ST7735_BLUE) ;
      ST7735_DrawPixel(X+1, i,   ST7735_BLUE) ;
    }
  }else if(lastj > j){
    for(i = j; i<lastj ; i++){
      ST7735_DrawPixel(X,   i,   ST7735_BLUE) ;
      ST7735_DrawPixel(X+1, i,   ST7735_BLUE) ;
    }
  }else{
    ST7735_DrawPixel(X,   j,   ST7735_BLUE) ;
    ST7735_DrawPixel(X+1, j,   ST7735_BLUE) ;
  }
  lastj = j;
}

// *************** ST7735_PlotPoints ********************
// Used in the voltage versus time plot, plot two points at y1, y2
// It does output to display
// Inputs: y1 is the y coordinate of the first point plotted
//         y2 is the y coordinate of the second point plotted
// Outputs: none
void ST7735_PlotPoints(int32_t y1,int32_t y2){int32_t j;
  if(y1<Ymin) y1=Ymin;
  if(y1>Ymax) y1=Ymax;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=Ymax maps to j=32
  // y=Ymin maps to j=159
  j = 32+(127*(Ymax-y1))/Yrange;
  if(j<32) j = 32;
  if(j>159) j = 159;
  ST7735_DrawPixel(X, j, ST7735_BLUE);
  if(y2<Ymin) y2=Ymin;
  if(y2>Ymax) y2=Ymax;
  j = 32+(127*(Ymax-y2))/Yrange;
  if(j<32) j = 32;
  if(j>159) j = 159;
  ST7735_DrawPixel(X, j, ST7735_BLACK);
}

// *************** ST7735_PlotBar ********************
// Used in the voltage versus time bar, plot one bar at y
// It does not output to display until RIT128x96x4ShowPlot called
// Inputs: y is the y coordinate of the bar plotted
// Outputs: none
void ST7735_PlotBar(int32_t y){
int32_t j;
  if(y<Ymin) y=Ymin;
  if(y>Ymax) y=Ymax;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=Ymax maps to j=32
  // y=Ymin maps to j=159
  j = 32+(127*(Ymax-y))/Yrange;
  ST7735_DrawFastVLine(X, j, 159-j, ST7735_BLACK);

}

// full scaled defined as 3V
// Input is 0 to 511, 0 => 159 and 511 => 32
uint8_t const dBfs[512]={
159, 159, 145, 137, 131, 126, 123, 119, 117, 114, 112, 110, 108, 107, 105, 104, 103, 101,
  100, 99, 98, 97, 96, 95, 94, 93, 93, 92, 91, 90, 90, 89, 88, 88, 87, 87, 86, 85, 85, 84,
  84, 83, 83, 82, 82, 81, 81, 81, 80, 80, 79, 79, 79, 78, 78, 77, 77, 77, 76, 76, 76, 75,
  75, 75, 74, 74, 74, 73, 73, 73, 72, 72, 72, 72, 71, 71, 71, 71, 70, 70, 70, 70, 69, 69,
  69, 69, 68, 68, 68, 68, 67, 67, 67, 67, 66, 66, 66, 66, 66, 65, 65, 65, 65, 65, 64, 64,
  64, 64, 64, 63, 63, 63, 63, 63, 63, 62, 62, 62, 62, 62, 62, 61, 61, 61, 61, 61, 61, 60,
  60, 60, 60, 60, 60, 59, 59, 59, 59, 59, 59, 59, 58, 58, 58, 58, 58, 58, 58, 57, 57, 57,
  57, 57, 57, 57, 56, 56, 56, 56, 56, 56, 56, 56, 55, 55, 55, 55, 55, 55, 55, 55, 54, 54,
  54, 54, 54, 54, 54, 54, 53, 53, 53, 53, 53, 53, 53, 53, 53, 52, 52, 52, 52, 52, 52, 52,
  52, 52, 52, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 50, 50, 50, 50, 50, 50, 50, 50, 50,
  50, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48,
  48, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 46, 46, 46, 46, 46, 46, 46, 46, 46,
  46, 46, 46, 46, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 44, 44, 44, 44, 44,
  44, 44, 44, 44, 44, 44, 44, 44, 44, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43,
  43, 43, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 41, 41, 41, 41, 41,
  41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
  40, 40, 40, 40, 40, 40, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
  39, 39, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 37,
  37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 36, 36, 36, 36,
  36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 35, 35, 35, 35, 35,
  35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 34, 34, 34, 34, 34, 34,
  34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 33, 33, 33, 33, 33,
  33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 32, 32
};

// *************** ST7735_PlotdBfs ********************
// Used in the amplitude versus frequency plot, plot bar point at y
// 0 to 0.625V scaled on a log plot from min to max
// It does output to display
// Inputs: y is the y ADC value of the bar plotted
// Outputs: none
void ST7735_PlotdBfs(int32_t y){
int32_t j;
  y = y/2; // 0 to 2047
  if(y<0) y=0;
  if(y>511) y=511;
  // X goes from 0 to 127
  // j goes from 159 to 32
  // y=511 maps to j=32
  // y=0 maps to j=159
  j = dBfs[y];
  ST7735_DrawFastVLine(X, j, 159-j, ST7735_BLACK);

}

// *************** ST7735_PlotNext ********************
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It does not output to display
// Inputs: none
// Outputs: none
void ST7735_PlotNext(void){
  if(X==127){
    X = 0;
  } else{
    X++;
  }
}

// *************** ST7735_PlotNextErase ********************
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It clears the vertical space into which the next pixel will be drawn
// Inputs: none
// Outputs: none
void ST7735_PlotNextErase(void){
  if(X==127){
    X = 0;
  } else{
    X++;
  }
  ST7735_DrawFastVLine(X,32,128,ST7735_Color565(228,228,228));
}

// Used in all the plots to write buffer to LCD
// Example 1 Voltage versus time
//    ST7735_PlotClear(0,4095);  // range from 0 to 4095
//    ST7735_PlotPoint(data); ST7735_PlotNext(); // called 128 times

// Example 2a Voltage versus time (N data points/pixel, time scale)
//    ST7735_PlotClear(0,4095);  // range from 0 to 4095
//    {   for(j=0;j<N;j++){
//          ST7735_PlotPoint(data[i++]); // called N times
//        }
//        ST7735_PlotNext();
//    }   // called 128 times

// Example 2b Voltage versus time (N data points/pixel, time scale)
//    ST7735_PlotClear(0,4095);  // range from 0 to 4095
//    {   for(j=0;j<N;j++){
//          ST7735_PlotLine(data[i++]); // called N times
//        }
//        ST7735_PlotNext();
//    }   // called 128 times

// Example 3 Voltage versus frequency (512 points)
//    perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//    ST7735_PlotClear(0,1023);  // clip large magnitudes
//    {
```

```
//       ST7735_PlotBar(mag[i++]); // called 4 times
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotNext();
//    }   // called 128 times

// Example 4 Voltage versus frequency (512 points), dB scale
//    perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//    ST7735_PlotClear(0,511);  // parameters ignored
//    {
//       ST7735_PlotdBfs(mag[i++]); // called 4 times
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotNext();
//    }   // called 128 times

// *************** ST7735_OutChar ********************
// Output one character to the LCD
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Inputs: 8-bit ASCII character
// Outputs: none
void ST7735_OutChar(char ch){
  if((ch == 10) || (ch == 13) || (ch == 27)){
    StY++; StX=0;
    if(StY>15){
      StY = 0;
    }
    ST7735_DrawString(0,StY,"                     ",StTextColor);
    return;
  }
  ST7735_DrawCharS(StX*6,StY*10,ch,ST7735_YELLOW,ST7735_BLACK, 1);
  StX++;
  if(StX>20){
    StX = 20;
    ST7735_DrawCharS(StX*6,StY*10,'*',ST7735_RED,ST7735_BLACK, 1);
  }
  return;
}

//********ST7735_OutString*****************
// Print a string of characters to the ST7735 LCD.
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// The string will not automatically wrap.
// inputs: ptr  pointer to NULL-terminated ASCII string
// outputs: none
void ST7735_OutString(char *ptr){
  while(*ptr){
    ST7735_OutChar(*ptr);
    ptr = ptr + 1;
  }
}

// ************** ST7735_SetTextColor ************************
// Sets the color in which the characters will be printed
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
// ********************************************************
void ST7735_SetTextColor(uint16_t color){
  StTextColor = color;
```

```
}

// Print a character to ST7735 LCD.
int fputc(int ch, FILE *f){
  ST7735_OutChar(ch);
  return 1;
}

// No input from Nokia, always return data.
int fgetc (FILE *f){
  return 0;
}

// Function called when file error occurs.
int ferror(FILE *f){
  /* Your implementation of ferror */
  return EOF;
}

// Abstraction of general output device
// Volume 2 section 3.4.5

// *************** Output_Init ********************
// Standard device driver initialization function for printf
// Initialize ST7735 LCD
// Inputs: none
// Outputs: none
void Output_Init(void){
  ST7735_InitR(INITR_REDTAB);
  ST7735_FillScreen(0);                 // set screen to black
}

// Clear display
void Output_Clear(void){ // Clears the display
  ST7735_FillScreen(0);    // set screen to black
}

// Turn off display (low power)
void Output_Off(void){   // Turns off the display
  Output_Clear();  // not implemented
}

// Turn on display
void Output_On(void){ // Turns on the display
  Output_Init();      // reinitialize
}

// set the color for future output
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
void Output_Color(uint32_t newColor){ // Set color of future output
  ST7735_SetTextColor(newColor);
}
```

ST7735.h

```
/***************************************************
  This is a library for the Adafruit 1.8" SPI display.
  This library works with the Adafruit 1.8" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/358
  as well as Adafruit raw 1.8" TFT display
  ----> http://www.adafruit.com/products/618

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

// ST7735.h
// Runs on MSP432
// Low level drivers for the ST7735 160x128 LCD based off of
// the file described above.
//    16-bit color, 128 wide by 160 high LCD
// Daniel Valvano
// July 9, 2015
// Augmented 7/17/2014 to have a simple graphics facility
// Tested with LaunchPadDLL.dll simulator 9/2/2014

/* This example accompanies the book
   "Embedded Systems: Introduction to the MSP432 Microcontroller",
   ISBN: 978-1512185676, Jonathan Valvano, copyright (c) 2015

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// hardware connections
// **********ST7735 TFT and SDC*******************
// ST7735
// Backlight (pin 10) connected to +3.3 V
// MISO (pin 9) unconnected
// SCK (pin 8) connected to P9.5 (UCA3CLK)
// MOSI (pin 7) connected to P9.7 (UCA3SIMO)
// TFT_CS (pin 6) connected to P9.4 (UCA3STE)
// CARD_CS (pin 5) unconnected
// Data/Command (pin 4) connected to P9.2 (GPIO), high for data, low for command
// RESET (pin 3) connected to P9.3 (GPIO)
// VCC (pin 2) connected to +3.3 V
// Gnd (pin 1) connected to ground

// **********wide.hk ST7735R with ADXL345 accelerometer *******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - P9.5 UCA3CLK SPI clock from microcontroller to TFT or SDC
// !SDA - P9.7 UCA3SIMO SPI data from microcontroller to TFT or SDC
// DC   - P9.2 TFT data/command
// RES  - P9.3 TFT reset
// CS   - P9.4 UCA3STE TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// SDA  – (NC) I2C data for ADXL345 accelerometer
// SCL  – (NC) I2C clock for ADXL345 accelerometer
// SDO  – (NC) I2C alternate address for ADXL345 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

// **********wide.hk ST7735R with ADXL335 accelerometer *******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - P9.5 UCA3CLK SPI clock from microcontroller to TFT or SDC
// !SDA - P9.7 UCA3SIMO SPI data from microcontroller to TFT or SDC
// DC   - P9.2 TFT data/command
// RES  - P9.3 TFT reset
// CS   - P9.4 UCA3STE TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// X– (NC) analog input X-axis from ADXL335 accelerometer
// Y– (NC) analog input Y-axis from ADXL335 accelerometer
// Z– (NC) analog input Z-axis from ADXL335 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

#ifndef _ST7735H_
#define _ST7735H_

// some flags for ST7735_InitR()
enum initRFlags{
  none,
  INITR_GREENTAB,
  INITR_REDTAB,
  INITR_BLACKTAB
};

#define ST7735_TFTWIDTH  128
#define ST7735_TFTHEIGHT 160


// Color definitions
#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0xF800
#define ST7735_RED     0x001F
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0xFFE0
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0x07FF
#define ST7735_WHITE   0xFFFF

void print_Alarm_LCD(unsigned char* Time_Date_Array, int count);
void displayHiCabinTemp();
void displayParkingWarning();

//------------ST7735_InitB------------
// Initialization for ST7735B screens.
// Input: none
// Output: none
void ST7735_InitB(void);


//------------ST7735_InitR------------
// Initialization for ST7735R screens (green or red tabs).
// Input: option one of the enumerated options depending on tabs
// Output: none
void ST7735_InitR(enum initRFlags option);


//------------ST7735_DrawPixel------------
// Color the pixel at the given coordinates with the given color.
// Requires 13 bytes of transmission
// Input: x     horizontal position of the pixel, columns from the left edge
//              must be less than 128
//              0 is on the left, 126 is near the right
//        y     vertical position of the pixel, rows from the top edge
//              must be less than 160
//              159 is near the wires, 0 is the side opposite the wires
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawPixel(int16_t x, int16_t y, uint16_t color);

//------------ST7735_DrawFastVLine------------
// Draw a vertical line at the given coordinates with the given height and color.
// A vertical line is parallel to the longer side of the rectangular display
// Requires (11 + 2*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
//        y     vertical position of the start of the line, rows from the top edge
//        h     vertical height of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);


//------------ST7735_DrawFastHLine------------
// Draw a horizontal line at the given coordinates with the given width and color.
// A horizontal line is parallel to the shorter side of the rectangular display
// Requires (11 + 2*w) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
//        y     vertical position of the start of the line, rows from the top edge
//        w     horizontal width of the line
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);


//------------ST7735_FillScreen------------
// Fill the screen with the given color.
// Requires 40,971 bytes of transmission
// Input: color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillScreen(uint16_t color);

//------------ST7735_FillRect------------
// Draw a filled rectangle at the given coordinates with the given width, height, and color.
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the top left corner of the rectangle, columns from the left edge
//        y     vertical position of the top left corner of the rectangle, rows from the top edge
//        w     horizontal width of the rectangle
//        h     vertical height of the rectangle
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);

//------------ST7735_Color565------------
// Pass 8-bit (each) R,G,B and get back 16-bit packed color.
// Input: r red value
//        g green value
//        b blue value
// Output: 16-bit color
uint16_t ST7735_Color565(uint8_t r, uint8_t g, uint8_t b);

//------------ST7735_SwapColor------------
// Swaps the red and blue values of the given 16-bit packed color;
// green is unchanged.
// Input: x 16-bit color in format B, G, R
// Output: 16-bit color in format R, G, B
uint16_t ST7735_SwapColor(uint16_t x) ;

//------------ST7735_DrawBitmap------------
// Displays a 16-bit color BMP image.  A bitmap file that is created
// by a PC image processing program has a header and may be padded
// with dummy columns so the data have four byte alignment.  This
// function assumes that all of that has been stripped out, and the
// array image[] has one 16-bit halfword for each pixel to be
// displayed on the screen (encoded in reverse order, which is
// standard for bitmap files).  An array can be created in this
// format from a 24-bit-per-pixel .bmp file using the associated
// converter program.
// (x,y) is the screen location of the lower left corner of BMP image
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the bottom left corner of the image, columns from the left edge
//        y     vertical position of the bottom left corner of the image, rows from the top edge
//        image pointer to a 16-bit color BMP image
//        w     number of pixels wide
//        h     number of pixels tall
// Output: none
// Must be less than or equal to 128 pixels wide by 160 pixels high
void ST7735_DrawBitmap(int16_t x, int16_t y, const uint16_t *image, int16_t w, int16_t h);

void Print_RPMs_toLCD(int RPMs);


//------------ST7735_DrawCharS------------
// Simple character draw function.  This is the same function from
// Adafruit_GFX.c but adapted for this processor.  However, each call
// to ST7735_DrawPixel() calls setAddrWindow(), which needs to send
// many extra data and commands.  If the background color is the same
// as the text color, no background will be printed, and text can be
// drawn right over existing images without covering them with a box.
// Requires (11 + 2*size*size)*6*8 bytes of transmission (image fully on screen; textcolor != bgColor)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawCharS(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t size);


//------------ST7735_DrawChar------------
// Advanced character draw function.  This is similar to the function
// from Adafruit_GFX.c but adapted for this processor.  However, this
// function only uses one call to setAddrWindow(), which allows it to
// run at least twice as fast.
// Requires (11 + size*size*6*8) bytes of transmission (assuming image fully on screen)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2 square)
// Output: none
void ST7735_DrawChar(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t size);


//------------ST7735_DrawString------------
// String draw function.
// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
// Input: x       columns from the left edge (0 to 20)
//        y       rows from the top edge (0 to 15)
//        pt      pointer to a null terminated string to be printed
//        textColor 16-bit color of the characters
// bgColor is Black and size is 1
// Output: number of characters printed
uint32_t ST7735_DrawString(uint16_t x, uint16_t y, char *pt, int16_t textColor);;


//********ST7735_SetCursor*****************
// Move the cursor to the desired X- and Y-position.  The
// next character will be printed here.  X=0 is the leftmost
// column.  Y=0 is the top row.
// inputs: newX  new X-position of the cursor (0<=newX<=20)
//         newY  new Y-position of the cursor (0<=newY<=15)
// outputs: none
void ST7735_SetCursor(uint32_t newX, uint32_t newY);


//-----------------------ST7735_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void ST7735_OutUDec(uint32_t n);


//------------ST7735_SetRotation------------
// Change the image rotation.
// Requires 2 bytes of transmission
// Input: m new rotation value (0 to 3)
// Output: none
void ST7735_SetRotation(uint8_t m) ;
//------------ST7735_InvertDisplay------------
// Send the command to invert all of the colors.
// Requires 1 byte of transmission
// Input: i 0 to disable inversion; non-zero to enable inversion
// Output: none
void ST7735_InvertDisplay(int i) ;

// graphics routines
// y coordinates 0 to 31 used for labels and messages
// y coordinates 32 to 159  128 pixels high
// x coordinates 0 to 127   128 pixels wide

// *************** ST7735_PlotClear ********************
// Clear the graphics buffer, set X coordinate to 0
// This routine clears the display
// Inputs: ymin and ymax are range of the plot
// Outputs: none
void ST7735_PlotClear(int32_t ymin, int32_t ymax);

// *************** ST7735_PlotPoint ********************
// Used in the voltage versus time plot, plot one point at y
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotPoint(int32_t y);

// *************** ST7735_PlotLine ********************
// Used in the voltage versus time plot, plot line to new point
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotLine(int32_t y);

// *************** ST7735_PlotPoints ********************
// Used in the voltage versus time plot, plot two points at y1, y2
// It does output to display
// Inputs: y1 is the y coordinate of the first point plotted
//        y2 is the y coordinate of the second point plotted
// Outputs: none
void ST7735_PlotPoints(int32_t y1,int32_t y2);

// *************** ST7735_PlotBar ********************
// Used in the voltage versus time bar, plot one bar at y
// It does not output to display until RIT128x96x4ShowPlot called
// Inputs: y is the y coordinate of the bar plotted
// Outputs: none
void ST7735_PlotBar(int32_t y);

// *************** ST7735_PlotdBfs ********************
// Used in the amplitude versus frequency plot, plot bar point at y
// 0 to 0.625V scaled on a log plot from min to max
// It does output to display
// Inputs: y is the y ADC value of the bar plotted
// Outputs: none
void ST7735_PlotdBfs(int32_t y);

// *************** ST7735_PlotNext ********************
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It does not output to display
// Inputs: none
// Outputs: none
void ST7735_PlotNext(void);

// *************** ST7735_PlotNextErase ********************
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It clears the vertical space into which the next pixel will be drawn
// Inputs: none
// Outputs: none
void ST7735_PlotNextErase(void);

// Used in all the plots to write buffer to LCD
// Example 1 Voltage versus time
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   ST7735_PlotPoint(data); ST7735_PlotNext(); // called 128 times

// Example 2a Voltage versus time (N data points/pixel, time scale)
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   {   for(j=0;j<N;j++){
//         ST7735_PlotPoint(data[i++]); // called N times
//       }
//       ST7735_PlotNext();
//   }   // called 128 times

// Example 2b Voltage versus time (N data points/pixel, time scale)
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   {   for(j=0;j<N;j++){
//         ST7735_PlotLine(data[i++]); // called N times
//       }
//       ST7735_PlotNext();
//   }   // called 128 times
// Example 3 Voltage versus frequency (512 points)
//    perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//    ST7735_PlotClear(0,1023);  // clip large magnitudes
//    {
//        ST7735_PlotBar(mag[i++]); // called 4 times
//        ST7735_PlotBar(mag[i++]);
//        ST7735_PlotBar(mag[i++]);
//        ST7735_PlotBar(mag[i++]);
//        ST7735_PlotNext();
//    }   // called 128 times

// Example 4 Voltage versus frequency (512 points), dB scale
//    perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//    ST7735_PlotClear(0,511);  // parameters ignored
//    {
//        ST7735_PlotdBfs(mag[i++]); // called 4 times
//        ST7735_PlotdBfs(mag[i++]);
//        ST7735_PlotdBfs(mag[i++]);
//        ST7735_PlotdBfs(mag[i++]);
//        ST7735_PlotNext();
//    }   // called 128 times

// *************** ST7735_OutChar ********************
// Output one character to the LCD
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Inputs: 8-bit ASCII character
// Outputs: none
void ST7735_OutChar(char ch);

//********ST7735_OutString*****************
// Print a string of characters to the ST7735 LCD.
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// The string will not automatically wrap.
// inputs: ptr  pointer to NULL-terminated ASCII string
// outputs: none
void ST7735_OutString(char *ptr);

// ************** ST7735_SetTextColor ************************
// Sets the color in which the characters will be printed
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
// ********************************************************
void ST7735_SetTextColor(uint16_t color);

// *************** Output_Init ********************
// Standard device driver initialization function for printf
// Initialize ST7735 LCD
// Inputs: none
// Outputs: none
void Output_Init(void);

// Clear display
void Output_Clear(void);

// Turn off display (low power)
void Output_Off(void);

// Turn on display
void Output_On(void);
```

```
// set the color for future output
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
void Output_Color(uint32_t newColor);

#endif
```

Stepper.c

```
/*
 * Stepper.c
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#include <stdio.h>
#include <stdint.h>
#include "msp.h"
#include "Stepper.h"
#include "SysTick_Library.h"

void Stepper1_Motor1_Pin1(void)
{
   P4->SEL0 &= ~BIT0;          //Set to GPIO Pin 2.3, IN1of motor
   P4->SEL1 &= ~BIT0;
   P4->DIR |= BIT0;          //Set to Output
   P4->OUT &= ~BIT0;          //turn off on initialize
}
void Stepper1_Motor1_Pin2(void)
{
   P4->SEL0 &= ~BIT1;          //Set to GPIO Pin 2.4, IN2
   P4->SEL1 &= ~BIT1;
   P4->DIR |= BIT1;          //Set to Output
   P4->OUT &= ~BIT1;          //turn off on initialize
}
void Stepper1_Motor2_Pin1(void)
{
   P4->SEL0 &= ~BIT2;          //Set to GPIO Pin 2.5, IN3
   P4->SEL1 &= ~BIT2;
   P4->DIR |= BIT2;          //Set to Output
   P4->OUT &= ~BIT2;          //turn off on initialize
}
void Stepper1_Motor2_Pin2(void)
{
   P4->SEL0 &= ~BIT3;          //Set to GPIO Pin 2.6, IN4
   P4->SEL1 &= ~BIT3;
   P4->DIR |= BIT3;          //Set to Output
   P4->OUT &= ~BIT3;          //turn off on initialize
}

//initialize speedometer stepper motor by calling all initializations for 4 pins
void Stepper1_Motor()
{
   Stepper1_Motor1_Pin1();
   Stepper1_Motor1_Pin2();
   Stepper1_Motor2_Pin1();
   Stepper1_Motor2_Pin2();
   Rotate_Stepper1_Forward_s_Steps(120);

   Rotate_Stepper1_Backward_s_Steps(120);
}
//turns the GPIO pins of motor on/off in  predetermined sequence using macros defined in .h
void Rotate_Stepper1_Forward_s_Steps(int s)
```

```
{
                STEPPER_PINS_OFF
                //macro to turn stepper pins off before
                ;
   int i;
   for (i = s; i > 0; i--)             //each loop = 1 step, loops 's' times
   {
      STEP1; //each macro is predefined as a fraction of a step, there are four, shown in Report
                                SysTick_delay_ms(2);                    //delay a little between
      STEP2;
                                SysTick_delay_ms(2);
      STEP3;
                                SysTick_delay_ms(2);
      STEP4;
                                SysTick_delay_ms(2);
   }
   STEPPER_PINS_OFF
   ;
}

//reverses forward step rotation, rotation motor backward s steps
void Rotate_Stepper1_Backward_s_Steps(int s)
{
                STEPPER_PINS_OFF
                ;
   int i;
   for (i = s; i > 0; i--)             //each loop = 1 step, loops 's' times
   {
      STEP4;                              //reverse steps to go backward
                                SysTick_delay_ms(2);
      STEP3;
                                SysTick_delay_ms(2);
      STEP2;
                                SysTick_delay_ms(2);
      STEP1;
                                SysTick_delay_ms(2);
   }
                STEPPER_PINS_OFF
                //turn pins off after rotation
   ;
}

//initialization of stepper motor pins
//GPIO outputs
void Stepper2_Motor1_Pin1(void)
{
   P4->SEL0 &= ~BIT0;        //Set to GPIO Pin 2.3, IN1of motor
   P4->SEL1 &= ~BIT0;
   P4->DIR |= BIT0;          //Set to Output
   P4->OUT &= ~BIT0;         //turn off on initialize
}

void Stepper2_Motor1_Pin2(void)
{
   P4->SEL0 &= ~BIT1;        //Set to GPIO Pin 2.4, IN2
   P4->SEL1 &= ~BIT1;
   P4->DIR |= BIT1;          //Set to Output
   P4->OUT &= ~BIT1;         //turn off on initialize

}

void Stepper2_Motor2_Pin1(void)
{
   P4->SEL0 &= ~BIT2;        //Set to GPIO Pin 2.5, IN3
   P4->SEL1 &= ~BIT2;
   P4->DIR |= BIT2;          //Set to Output
   P4->OUT &= ~BIT2;          //turn off on initialize
}

void Stepper2_Motor2_Pin2(void)
{
   P4->SEL0 &= ~BIT3;         //Set to GPIO Pin 2.6, IN4
   P4->SEL1 &= ~BIT3;
   P4->DIR |= BIT3;          //Set to Output
   P4->OUT &= ~BIT3;          //turn off on initialize
}

////////////////////////////////
/*
 *                                 Initialization of the test stepper for speed checking
 */
///////////////////////////////
void TESTStepper_Motor()
{
   TESTStepper_Motor1_Pin1();
   TESTStepper_Motor1_Pin2();
   TESTStepper_Motor2_Pin1();
   TESTStepper_Motor2_Pin2();
}

void TESTStepper_Motor1_Pin1(void)
{
   P9->SEL0 &= ~BIT0;         //Set to GPIO Pin 2.3, IN1of motor
   P9->SEL1 &= ~BIT0;
   P9->DIR |= BIT0;          //Set to Output
   P9->OUT &= ~BIT0;          //turn off on initialize
}

void TESTStepper_Motor1_Pin2(void)
{
   P9->SEL0 &= ~BIT1;         //Set to GPIO Pin 2.4, IN2
   P9->SEL1 &= ~BIT1;
   P9->DIR |= BIT1;          //Set to Output
   P9->OUT &= ~BIT1;          //turn off on initialize

}

void TESTStepper_Motor2_Pin1(void)
{
   P9->SEL0 &= ~BIT2;         //Set to GPIO Pin 2.5, IN3
   P9->SEL1 &= ~BIT2;
   P9->DIR |= BIT2;          //Set to Output
   P9->OUT &= ~BIT2;          //turn off on initialize
}

void TESTStepper_Motor2_Pin2(void)
{
   P9->SEL0 &= ~BIT3;         //Set to GPIO Pin 2.6, IN4
   P9->SEL1 &= ~BIT3;
   P9->DIR |= BIT3;          //Set to Output
   P9->OUT &= ~BIT3;          //turn off on initialize
}

void TEST_Stepper_Rotate(int s)
{

   int i;
   for (i = s; i > 0; i--)              //each loop = 1 step, loops 's' times
   {
      P9->OUT &= ~0x0F;
   TESTSTEP1; //each macro is predefined as a fraction of a step, there are four, shown in Report
      SysTick_delay_ms(10);                    //delay a little between
      //TESTSTEP2;
      //    SysTick_delay_us(200);
      P9->OUT &= ~0x0F;

   TESTSTEP3;
      SysTick_delay_ms(10);
      //TESTSTEP4;
      //    SysTick_delay_us(200);
      P9->OUT &= ~0x0F;
      TESTSTEP5;
      SysTick_delay_ms(10);
      //     TESTSTEP6;
      //    SysTick_delay_us(200);
      P9->OUT &= ~0x0F;
      TESTSTEP7;
      SysTick_delay_ms(10);
      //     TESTSTEP8;
      //    SysTick_delay_us(200);
   }

}
//initialiizes tachometer
//unused because I did not have time to find an unused timerA that did not affect other components
void Tachometer_Init(void) {
            P5->SEL0 |= BIT7;
            P5->SEL1 &= ~BIT7;
            P5->DIR |= BIT7;

            TIMER_A2->CTL = 0x0294;
            TIMER_A2->CCR[0] = 60000;
            TIMER_A2->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7;
            TIMER_A2->CCR[2] = 6000;
            SysTick_delay_ms(10);
}
//sets servo position depending on the RPMs calculated
void TachometerSpeed(int rotations) {
            int duty = rotations / 20;
            TIMER_A2->CCR[2] = 6000 - ((duty * 6000) / 35);
}
```

Stepper.h

```
/*
 * Stepper.h
 *
 *  Created on: Nov 29, 2020
 *      Author: 7jorchay
 */

#ifndef STEPPER_H_
#define STEPPER_H_

extern volatile int Current_Speed_mph;

#define STEP1   P4->OUT = (BIT1 | BIT3)        //in2 | in4    0x0A
#define STEP2   P4->OUT = (BIT3 | BIT0)        //in4 | in1    0x
#define STEP3   P4->OUT = (BIT0 | BIT2)        //in1 | in3
#define STEP4   P4->OUT = (BIT1 | BIT2)        //in2 | in3
#define STEPPER_PINS_OFF    P4->OUT = 0;
```

```
#define TESTSTEP1   P9->OUT = 0x0E        //in1 | in2 | in3
#define TESTSTEP2   P9->OUT = 0x0C        //in1 | in2
#define TESTSTEP3   P9->OUT = 0x0D        //in1 | in2 | in4
#define TESTSTEP4   P9->OUT = 0x09        //in1 | in4
#define TESTSTEP5   P9->OUT = 0x0B        //in1 | in3 | in4
#define TESTSTEP6   P9->OUT = 0x03        //in3 | in4
#define TESTSTEP7   P9->OUT = 0x07        //in2 | in3 | in4
#define TESTSTEP8   P9->OUT = 0x06        //in2 | in3

void Stepper1_Motor1_Pin1();
void Stepper1_Motor1_Pin2();
void Stepper1_Motor2_Pin1();
void Stepper1_Motor2_Pin2();
void Stepper1_Motor();                  //calls all motor pin initializations
void Rotate_Stepper1_Forward_s_Steps(int s);     //runs forward stepper sequence
void Rotate_Stepper1_Backward_s_Steps(int s);   //runs backward stepper sequence


void TESTStepper_Motor(void);
void TESTStepper_Motor1_Pin1(void);
void TESTStepper_Motor1_Pin2(void);
void TESTStepper_Motor2_Pin1(void);
void TESTStepper_Motor2_Pin2(void);

void Tachometer_Init(void);
void TachometerSpeed(int rotations);

void TEST_Stepper_Rotate(int s);

#endif /* STEPPER_H_ */
```

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"

//volatile uint8_t ButtonPress = 0;
//uint8_t num;

/// ****| 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, core clock, no interrupts, this is the ENABLE and CLKSOURSE
}

/// ****| SysTickInit_WithInterrupts | **************//*
/// * Brief: Initialize the SysTick peripheral for use
/// *         with interrupts (interrupt delays)
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// *****************************************************/
void SysTickInit_WithInterrupts(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_INT_EN | STCSR_EN);         // this is the ENABLE, TICKINT, CLKSOURSE a WITH interrupts 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
                //clock now 48MHz, multiply ms for 3MHz by 16 and run in a loop for the amount of ms
   int i;
   for (i = 0; i < ms_delay; i++)
   {
      SysTick->LOAD = 3000 * 16;
   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){
   int i;
   //Delays time_ms number of milliseconds
   //Assume 3MHz clock -> 3 cycles per microsecond
   for (i = 0; i < us_delay; i++)
   {
                                SysTick->LOAD = 48; //cant overflow, so we count to 48 (3 * 16) the amount of microsends we 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) {
}
```

SysTick_Library.h

```
/*******************************************************************************************
*******************************************************************************************
*                          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).
*
*
* ******************************************************************************************/

#ifndef SYSTICK_LIBRARY_H_
#define SYSTICK_LIBRARY_H_

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

// SysTick Control and Status Register (STCSR)
#define STCSR_COUNT_FG  (0x0100)
#define STCSR_CLKSRC    (0x0004)
#define STCSR_INT_EN    (0x0002)
#define STCSR_EN        (0x0001)

extern volatile uint8_t ButtonPress;
extern uint8_t num;

/********************** Macro Prototypes *********************************
**************************************************************************
  SysTick Control and Status Register (STCSR) as discussed in lectures

//#define STCSR_COUNT_FG  BIT16
//#define STCSR_CLKSRC    BIT2         // this is the CLKSOURSE bit
//#define STCSR_INT_EN    BIT1         // This is the TICKINT  bit
//#define STCSR_EN        BIT0         // This is the ENABLE bit

********************** Macro Prototypes *********************************
**************************************************************************

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

```
#endif /* SYSTICK_LIBRARY_H_ */
```

TIMERA.c

```
/*
 * TIMERA.c
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */
// bit  mode
// 9-8  20    TASSEL, SMCLK=12MHz
// 7-6  11    ID, divide by 8
// 5-4  11    MC, up-down mode
// 2    0     TACLR, no clear
// 1    0     TAIE, no interrupt
// 0          TAIFG
//NVIC_EnableIRQ(TA1_0_IRQn);
//void TA1_0_IRQHandler(void);
#include <stdio.h>
#include <stdint.h>
#include "TIMERA.h"
#include "msp.h"

void TIMERA0_PWM_init(uint16_t period)
{
   TIMER_A0->CCR[0] = period;   // Period is 2*period*8*83.33ns is 1.333*period
   TIMER_A0->CTL = 0x02F0; // SMCLK=12MHz, divide by 8, up-down mode (makes period 2 * period * 8)
}
/*
 void TIMERA0_1_PWM_Init(uint16_t duty1)
{
   P2DIR |= 0x30;                              // P2.4, P2.5 output
   P2SEL0 |= 0x30;                             // P2.4, P2.5 Timer0A functions
   P2SEL1 &= ~0x30;                            // P2.4, P2.5 Timer0A function
   TIMER_A0->CCTL[0] = 0x0080;                 // CCI0 toggle
   //TA0EX0 = 0x0000;       //    divide by 1
   TIMER_A0->CCTL[1] = 0x0040;                 // CCR1 toggle/reset
   TIMER_A0->CCR[1] = duty1;               // CCR1 duty cycle is duty1/period

}

void TIMERA0_2_PWM_Init(uint16_t duty2)
{
   P7->DIR |= 0x08;         // 7.3 output
   P7->SEL0 |= 0x08;        // P7.3 Timer0A functions
   P7->SEL1 &= ~0x08;       // P7.3 Timer0A functions

   TIMER_A0->CCTL[2] = 0x0040;     // CCR2 toggle/reset
   TIMER_A0->CCR[2] = duty2;       // CCR2 duty cycle is duty2/period
}
 */
/*
void TimerA0_1_PWM_init(float dutyCycle_percentage)
{                          //Timer A initialization function

   P2->SEL0 |= BIT4;                           //P2.4 set to TA0.1
   P2->SEL1 &= ~BIT4;
   P2->DIR |= BIT4;                            //P2.4 Set to Output
   TIMER_A0->CCR[0] = 10000;
   TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;    //CCR1 outmode 7
   TIMER_A0->CCR[1] = 10000 * dutyCycle_percentage;           //PWM duty cycle
   TIMER_A0->CTL = 0x0214;               //SMCLK, Up Mode, Clear TAR to start
}
 */
```

```
void TIMERA1_0_Timer_withInterrupts()
{
   P2->SEL0 &= ~BIT0;
   P2->SEL1 &= ~BIT0;  //P2.0 set to GPIO
   P2->DIR |= BIT0;    //P2.0 direction set to output
   P2->OUT &= ~BIT0;   //P2.0 set to off or as 0

   TIMER_A1->CTL = 0x0111;           //0000_0001_1101_0110: ACLK (32KHz),
   TIMER_A1->CCR[0] = 16000 - 1;      //half a second of 32KHz
   TIMER_A1->CCTL[0] |= 10;
}

void TIMERA1_0_oneMSInterval_init()
{
   //note ACLK has to be already divided by 4
   TIMER_A1->CTL = 0x0136;     //ACLK, Up mode, clear TAR, /1, interrupt enable
               TIMER_A1->CCR[0] = 4; //clock is 32768 Hz divided by 4, so 32768 / 4 / 2 = 1 second
   TIMER_A1->CCTL[0] = 0x0810; //outmode not needed, just interval timer, enable interrupts and synchronize

   //4092 = 1 second
   //2046 = 1/2 second
   //4092 / 1000 = 4 = 1 ms
}
void TIMERA3_0_HalfSecondInterval_init()
{
   //note ACLK has to be already divided by 4
   TIMER_A3->CTL = 0x0136;     //ACLK, Up mode, clear TAR, /1, interrupt enable
   TIMER_A3->CCR[0] = 2046; //clock is 32768 Hz divided by 4, so 32768 / 4 / 2 = 1 second
   TIMER_A3->CCTL[0] = 0x0810; //outmode not needed, just interval timer, enable interrupts and synchronize

   //4092 = 1 second
   //2046 = 1/2 second
}

/*
 void TIMERA3_1_oneMSInterval_init()
 {
 //note ACLK has to be already divided by 4
 TIMER_A3->CTL = 0x0136;     //ACLK, Up mode, clear TAR, /1, interrupt enable
 TIMER_A3->CCR[1] = 4; //clock is 32768 Hz divided by 4, so 32768 / 4 / 2 = 1 second
 TIMER_A3->CCTL[1] = 0x0810; //outmode not needed, just interval timer, enable interrupts and synchronize

 //4092 = 1 second
 //2046 = 1/2 second
 //4092 / 1000 = 5 = 1 ms
 }
 */
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
/*
 * HANDLERS
 *
 * Each function includes a clear and a toggle onboard LED 2.0 command
 */
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
```

TIMERA.h

```
/*
 * TIMERA.h
 *
 *  Created on: Nov 19, 2020
 *      Author: 7jorchay
 */
```

```
#ifndef TIMERA_H_
#define TIMERA_H_

//////////////////////////////////////////
/*
 * PWM Modes
 */
//////////////////////////////////////////
void TIMERA0_PWM_init(uint16_t period);
void TIMERA0_1_PWM_Init(uint16_t duty1); //use for blinkers
void TIMERA0_2_PWM_Init(uint16_t duty2);
//void TimerA0_1_PWM_init(float dutyCycle);
//////////////////////////////////////////////////////////////////////////////
/*
 *                        Timer Mode
 */
/////////////////////////////////////////////////////////////////////////////
/*
 *                      Initializations
 */
////////
void TIMERA1_0_oneMSInterval_init();
void TIMERA1_0_Timer_withInterrupts(); //unused
void TIMERA3_0_HalfSecondInterval_init();
void TIMERA3_1_oneMSInterval_init();

////////////////////////////////////////////////////////////////////////////
/*
 *                      Handlers
 */
////////////////////////////////////////////////////////////////////////////
void TA0_0_IRQHandler(void);
void TA1_0_IRQHandler(void);
void TA2_0_IRQHandler(void);
void TA3_0_IRQHandler(void);
#endif /* TIMERA_H_ */
```