

Team 11

The Electric Rover

| Revision | Description |
|----------|--|
| 1A | Chance - 1, 2, Cover Page, Formatting |
| 1B | Sloane - 4, 4.1, 4.2, 4.3, 4.4, 6, 6.1, 6.2, 6.3, 6.4, Fig. 6, Fig 7, Fig 8, Fig 9, Fig 10, Formatting |
| 1C | Zhiming - 2, 3, 3.1, 3.2, Fig. 1, Fig. 2, Fig. 3, team name, Table of Content, Document Intergration |
| 2A | Chance - 1,2,7.4,8.4,9.4 |
| 2B | Sloane - 7.3, 8.3, 9.3.a, 9.3.b, Fig. 15, Fig. 16, Fig 17, 6.4, 6.5 |
| 2C | Zhiming - 7.1,8.1,9.1,3.5 |
| 2D | William - Formatting/Revisions, 7.2, 8.2, 9.2, 3.4, 4.5 |
| 3A | Sloane - Formatting/Revisions |
| 3B | Zhiming - Formatting/Revisions 7.1, 8.1 |
| 3C | William - Formatting/Revisions 7.2, 8.2 |
| 4A | William – Formatting/Revision TOC, Software listing 9.4-9.12, Conclusion |

| | | | | |
|---|--|---|-------------------|--------------------------|
| Team Members Chance Chavis William Brazil Sloane Kathryn Cox Zhiming Dai | Originator: Chavis, Brazil, Cox, Dai | | | |
| | Checked: 11/23/2020 | Released: 11/23/2020 | | |
| | Filename: Project_10_Writeup | | | |
| | Title: Small Electronic Line-Detecting Vehicle | | | |
| This document contains information that is PRIVILEGED and CONFIDENTIAL ; you are hereby notified that any dissemination of this information is strictly prohibited. | Date: 11/23/2020 | Document Number: 0-2020-000-1123-10 | Rev: 4A | Sheet: 1 of 94 |

Table of Contents

| | | |
|--------|----------------------------------|----|
| 1. | Scope..... | 4 |
| 2. | Abbreviations..... | 4 |
| 3. | Overview..... | 4 |
| 3.1. | Power Control Board and LCD..... | 5 |
| 3.2. | MSP430..... | 5 |
| 3.3. | User Interface Blocks..... | 5 |
| 3.4. | Motor Block..... | 6 |
| 3.5. | Emitter/detector..... | 6 |
| 4. | Hardware..... | 8 |
| 4.1. | User Interface..... | 8 |
| 4.2. | Power Supply..... | 9 |
| 4.3. | Memory..... | 9 |
| 4.4. | Control Board..... | 11 |
| 4.5. | Motors..... | 12 |
| 4.6. | Emitter/Detector Board..... | 14 |
| 6. | Test Process..... | 14 |
| 6.1. | Power System Test..... | 14 |
| 6.2. | Power Circuit Test..... | 14 |
| 6.3. | Voltage Verification Test..... | 14 |
| 6.4. | P-FET and N-FET Test..... | 15 |
| 6.5. | Emitter/Detector Test..... | 15 |
| 6.6. | Baud Rate Test..... | 15 |
| 7. | Software..... | 17 |
| 7.1. | Main..... | 17 |
| 7.2. | Initialization..... | 17 |
| 7.3. | Timers..... | 17 |
| 7.4. | ADC and Interrupt Files..... | 18 |
| 8. | Flow Chart..... | 18 |
| 8.1. | Main Blocks..... | 18 |
| 8.2. | Initialization Blocks..... | 19 |
| 8.3. | Timer Blocks..... | 20 |
| 9. | Software Listing..... | 22 |
| 9.1. | Main.c..... | 22 |
| 9.2. | System_Init.c..... | 26 |
| 9.3. | Interrupt.c..... | 31 |
| 9.3.1 | Interrupt_timers.c..... | 31 |
| 9.3.2 | Timers.c..... | 34 |
| 9.3.4 | interrupt_eUSCI.c..... | 38 |
| 9.4. | ADC.c..... | 40 |
| 9.5. | Clocks.c..... | 44 |
| 9.6. | Display.c..... | 48 |
| 9.7. | Ports..... | 55 |
| 9.7.1. | Port1.c..... | 55 |
| 9.7.2. | Port2.c..... | 57 |
| 9.7.3. | Port3.c..... | 59 |

| | | |
|--------|------------------|----|
| 9.7.4. | Port4.c | 61 |
| 9.7.5. | Port5.c | 63 |
| 9.7.6. | Port6.c | 64 |
| 9.7.7. | Ports.c | 66 |
| 9.8. | States.c | 67 |
| 9.9. | Switches.c | 77 |
| 9.10. | System.c | 80 |
| 9.11. | Timers.c | 82 |
| 9.12. | Wheels.c | 90 |
| 10. | Conclusion | 93 |

Table of Figures

| | | |
|-----------|--|----|
| Figure 1 | Current System | 4 |
| Figure 2 | Power Control Board and LCD | 5 |
| Figure 3 | MSP430 | 5 |
| Figure 4 | Input Blocks..... | 6 |
| Figure 5 | emitter/detector | 6 |
| Figure 6 | LCD Schematic | 8 |
| Figure 7 | Power Supply Board..... | 9 |
| Figure 8 | MSP430 Schematics Part 1 | 10 |
| Figure 9 | MSP430 Schematics Part 2 | 10 |
| Figure 10 | Control Board Interconnect Schematic | 11 |
| Figure 11 | Left Motor H-Bridge | 12 |
| Figure 12 | Right Motor H-Bridge | 13 |
| Figure 13 | Software Flowchart | 12 |
| Figure 14 | Configuration 1 | 15 |
| Figure 15 | Configuration 2..... | 16 |
| Figure 16 | Main Block Flowchart..... | 18 |
| Figure 15 | Initialization block flowchart | 16 |
| Figure 16 | Function Timer0_B0_ISR Flowchart | 20 |
| Figure 17 | Function Timer0_B1_ISR Flowchart | 21 |
| Figure 18 | Function Init_Timers and Init_Timer_B3 Flowchart | 22 |

1. Scope

In the first project we get the DAC board which creates the battery powered system. We tested the board using the voltmeter to be sure that we were getting the 3.3 Volts out of the converter. The second project we received the LCD which we then installed onto the FRAM board. We used code already made after installing our LCD and ran it to make sure that the LCD and the backlight worked. Next created the exterior of the car moving it forward with the forward control of the H-bridge. In the second set of revisions we used controlled movement with only the forward part of the H-bridge to move forward and reverse, allowing us to move the car in predetermined shapes. In the second set of revisions we used the full control of the H-bridge, using this we did timed travel forward and then timed travel in reverse and ended with a spinning motion. The third set of revisions we added the emitter/detector that can detect a black line and created code that would show the thumbwheel as a digital value. The overall goal was to create an electrical vehicle that can follow a black line navigating using WI-FI connection.

2. Abbreviations

ADC: analog to digital converter

DAC: digital to analog converter

FRAM: ferroelectric random-access memory.

GPIO: general purpose input and output

Hz: Hertz

LCD: Liquid Crystal Display

LED: light emitting diode

N-FET: negative field effect transistor

P-FET: positive field effect transistor

PCB: power control board

SW: switch

USB: universal serial bus

3. Overview

The current system is composed of programmable boards, including the power system board and the FRAM. An LCD screen is soldered on the control board and will display messages written in the program. Once the program is installed successfully in the system, the system will work on battery power without USB connection to a computer.

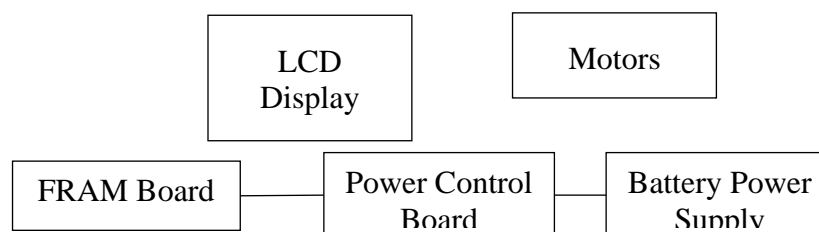


Figure 1 Current System

3.1. Power Control Board and LCD

The Power Control Board and LCD Block is the block where the battery pack is connected. The power is fed into the whole system and the message is transferred to the LCD for displaying through the through-hole pins

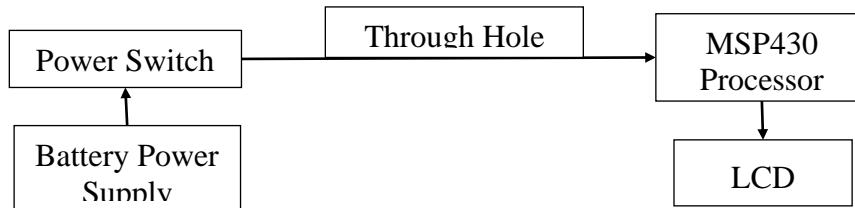


Figure 2 Power Control Board and LCD

3.2. MSP430

The MSP430 Board Block is the main processing board, where our software is written into, and where the controls and signals are being transferred and processed. The board is powered by the battery power transferred through the through-hole pins.

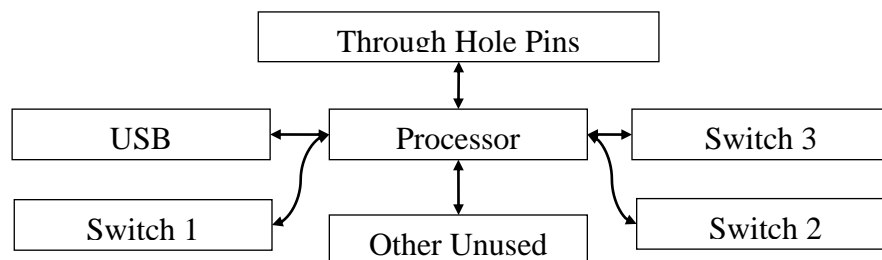


Figure 3 MSP430

3.3. User Interface Blocks

The user interface is controlled through the control board and the MSP430. Both of these boards control the LCD display, the thumbwheel, all LEDs, and all switches.

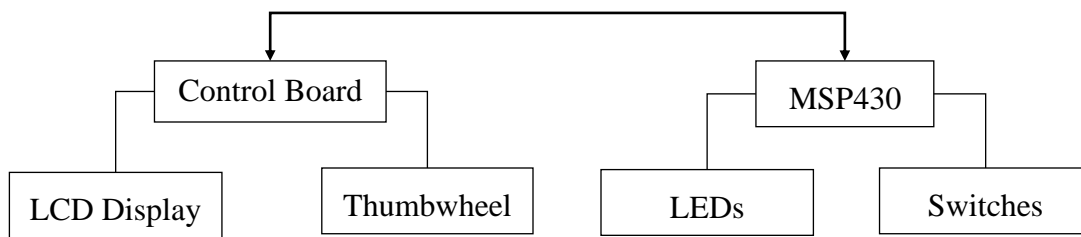


Figure 4 Input Blocks

3.4. Motor Block

The motor block is controlled through the control board. The board sends instructions to the left and right motors.

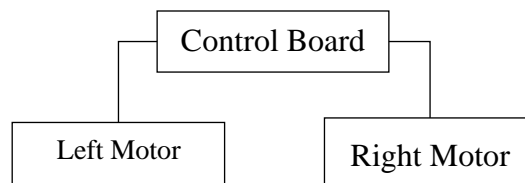


Figure 5 Motor Block

3.5. Emitter/detector

The emitter and detector blocks are soldered into a PCB board that is separated from the FET board. Then the PCB board is connected to the FET board by a 5-pin connector with wires. The FET board is then connected to the power control board by through-hole pins.

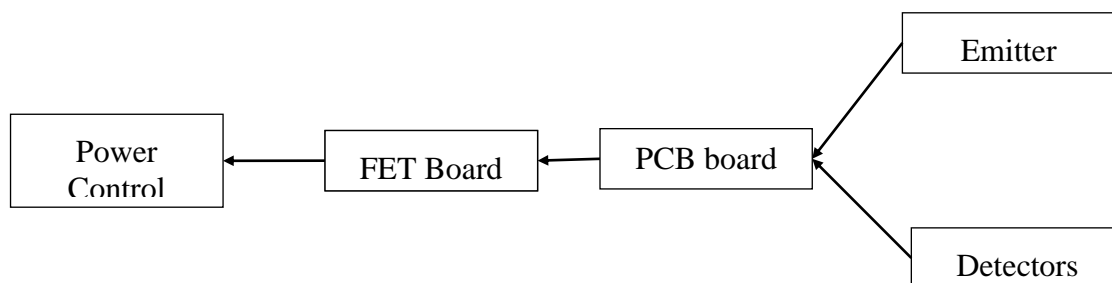


Figure 6 Emitter/Detector Block

4. Hardware

The sections below are descriptions of hardware for the user interface, power supply, control board, motors, and the emitter/detector board.

4.1. User Interface

The user interface consists of the LCD display which operates using 3.3 volts. The part number for this display is EA DOGS104-A. The LCD displays characters from the FRAM board as programmed by the user.

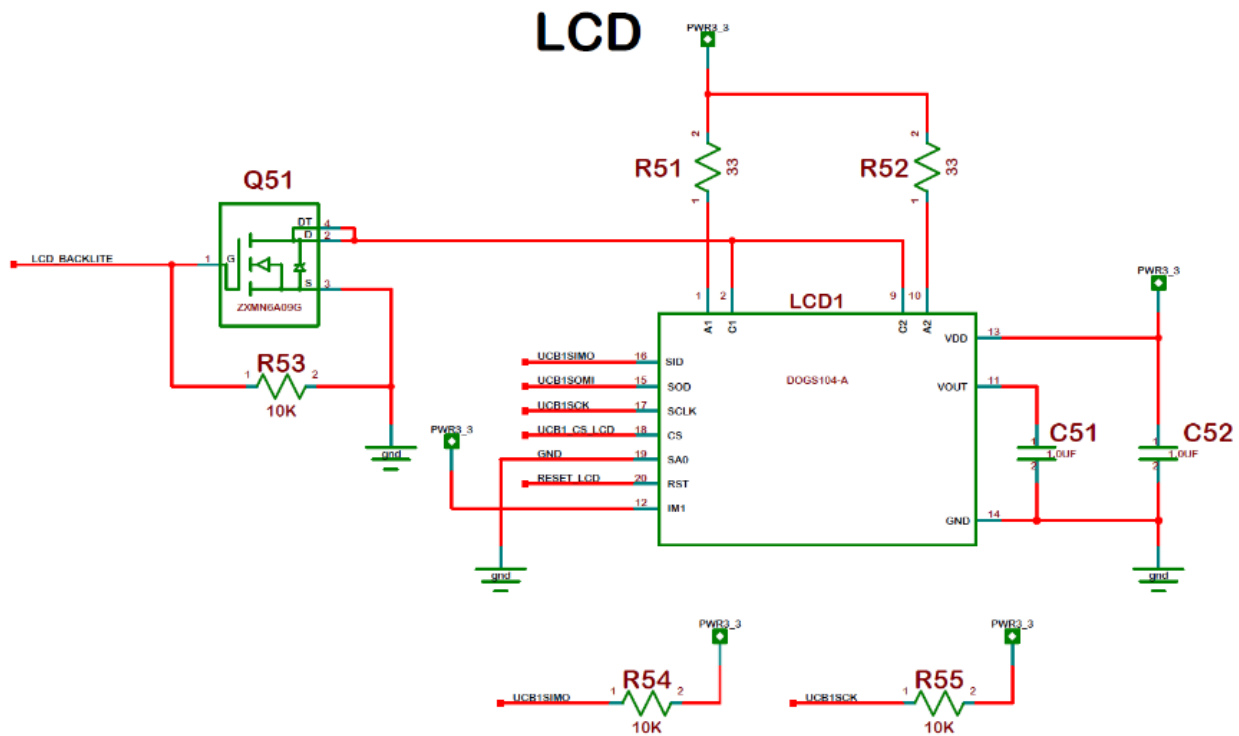


Figure 7 LCD Schematic

4.2. Power Supply

All electronic parts of the vehicle are powered by 4 AA batteries. This vehicle operates between 1.8V – 3.3V and has a shutdown current of <1uA. The power supply is connected to a boost converter used as a sepic converter.

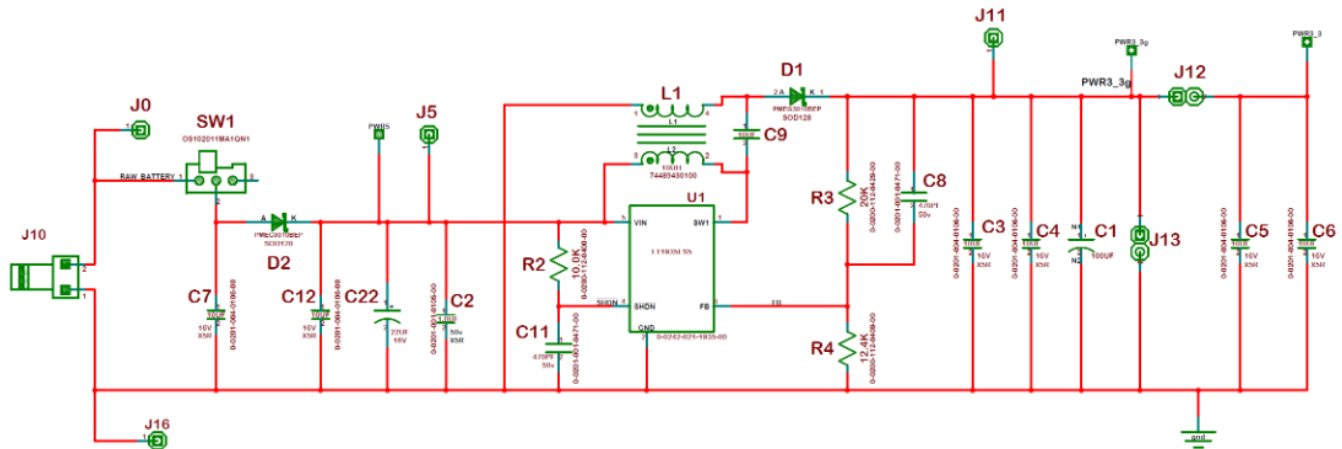


Figure 8 Power Supply Board

4.3. Memory

The FRAM board is a MSP430. This board can be connected to a computer for programming using a USB port. The board can store up to 32KB of memory, split between the program and data. Three switches are attached to the MSP430 as well.

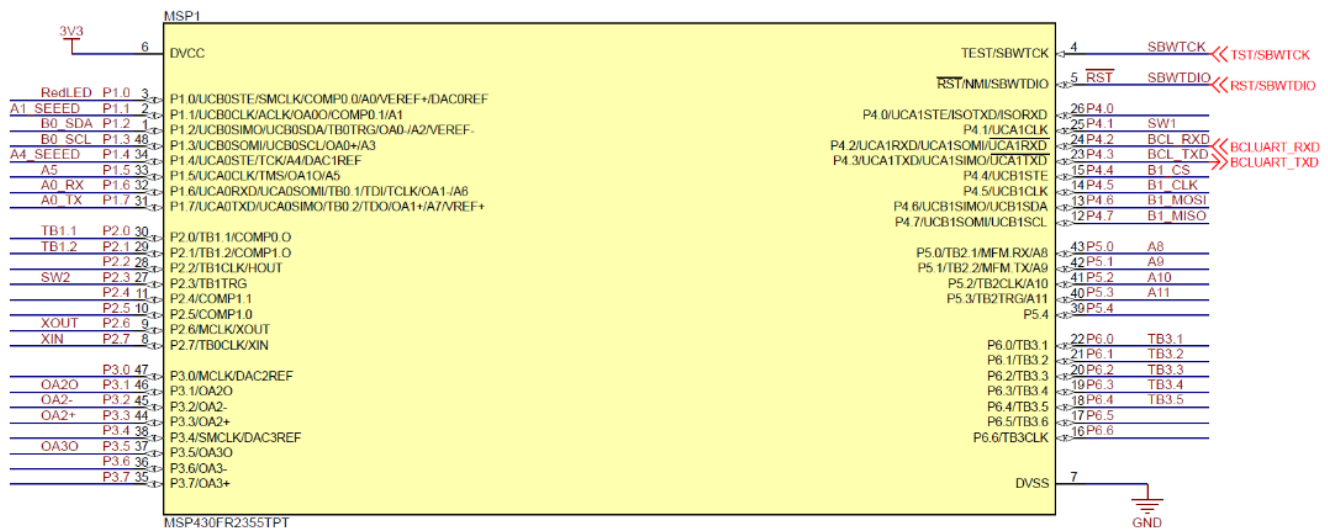


Figure 9 MSP430 Schematics Part 1

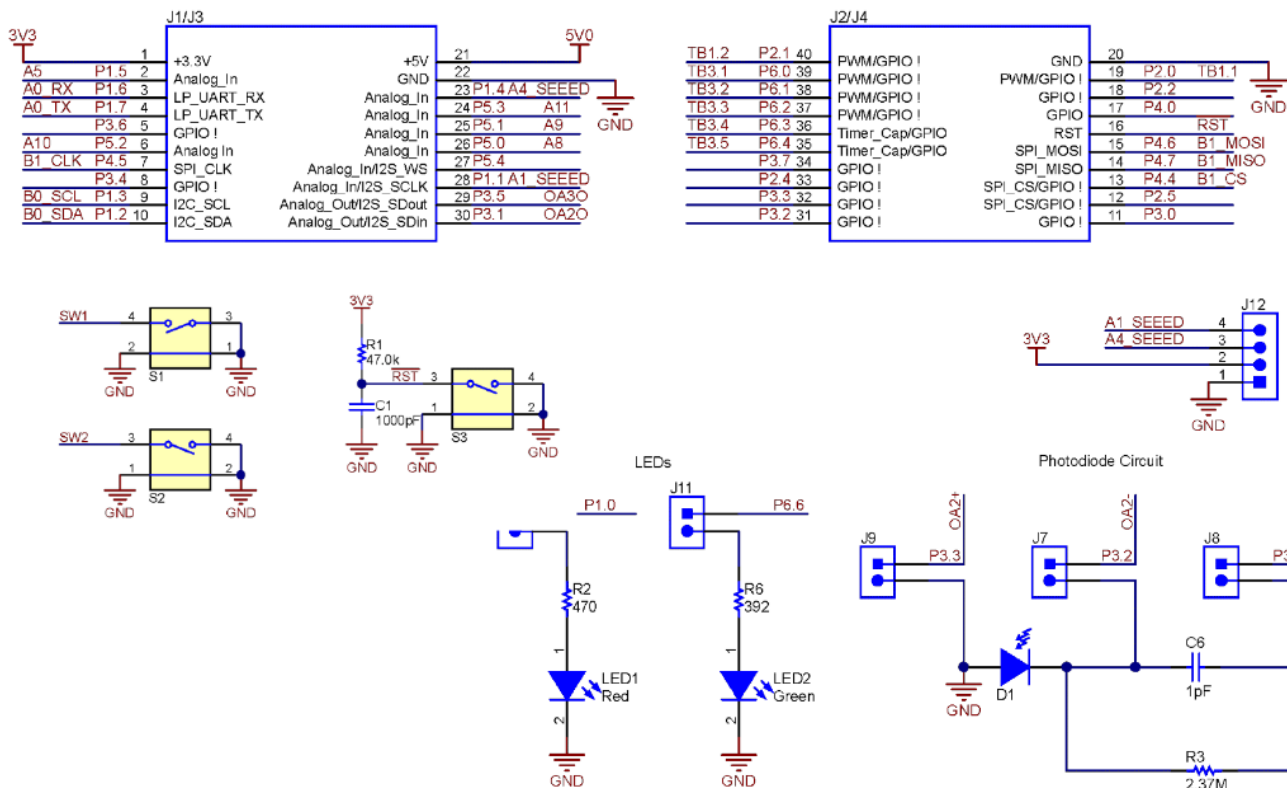


Figure 10 MSP430 Schematics Part 2

4.4. Control Board

The control board is connected to the FRAM board using pins. The board contains a thumbwheel and LCD display, as well as the on and off switch to power the vehicle.

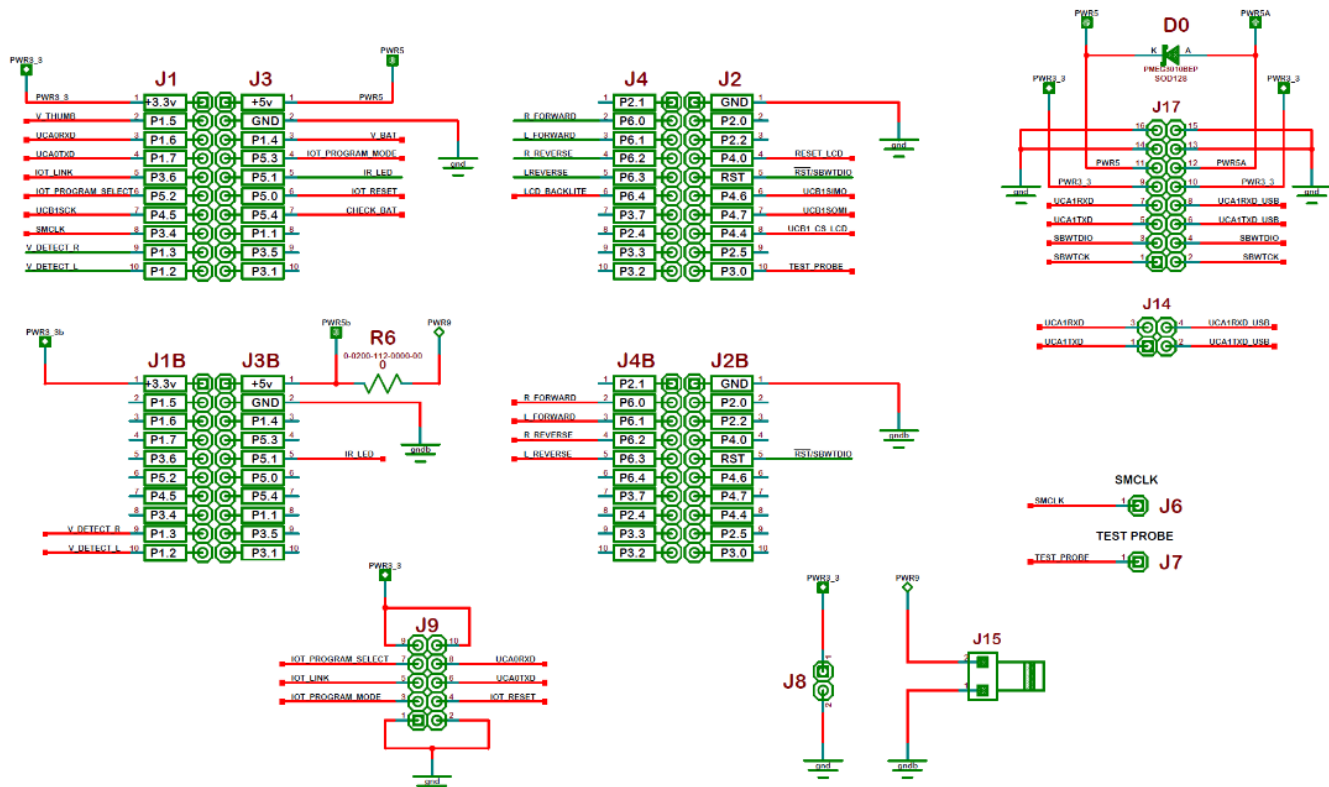


Figure 11 Control Board Interconnect Schematic

4.5. Motors

There are two motors attached to the main frame that have the capability to move the car in two directions: forward and reverse. Both motors are able to operate thanks to the J43 and J21 which control the left and right motors respectively.

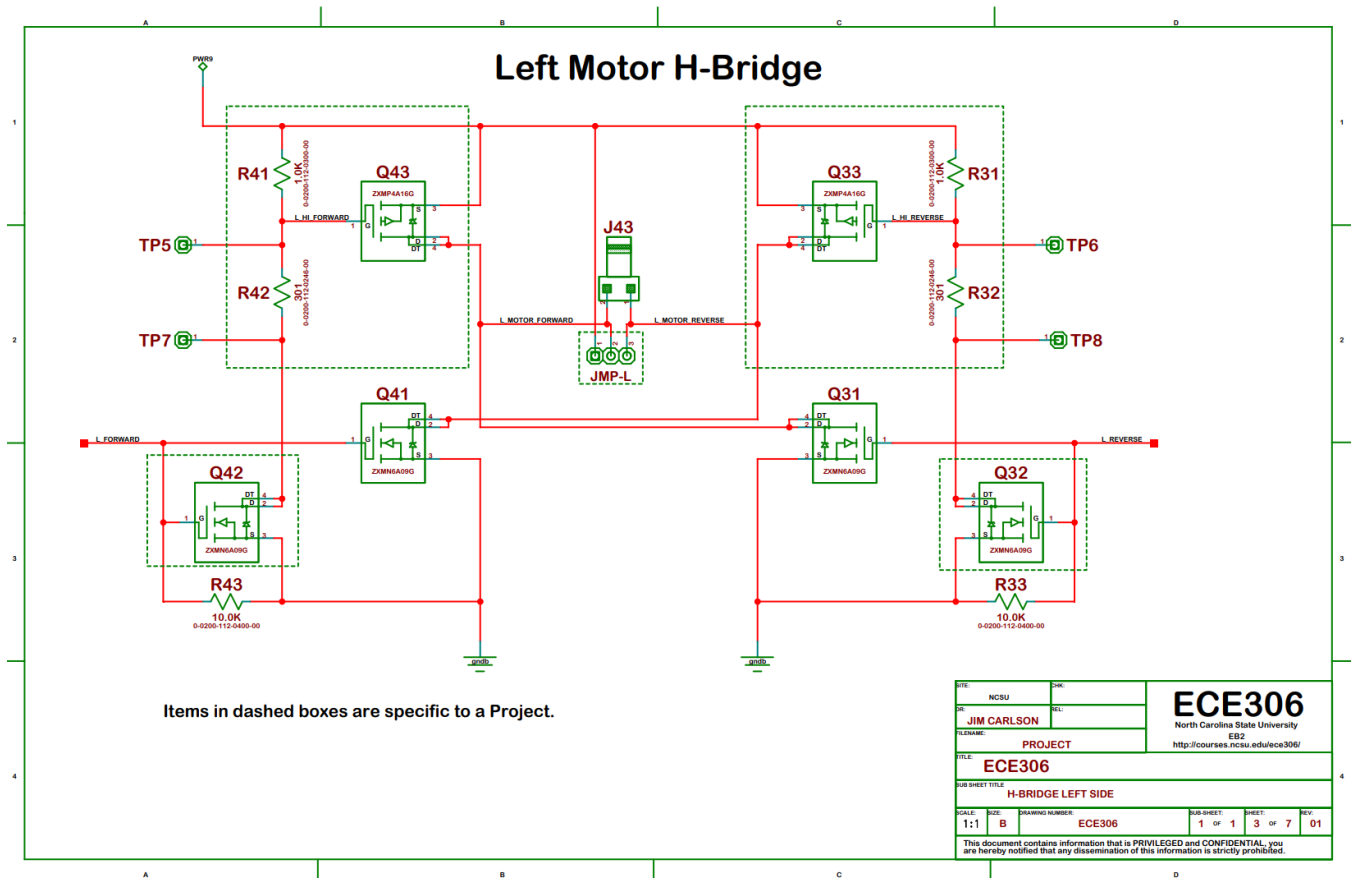


Figure 12 Left Motor H-Bridge

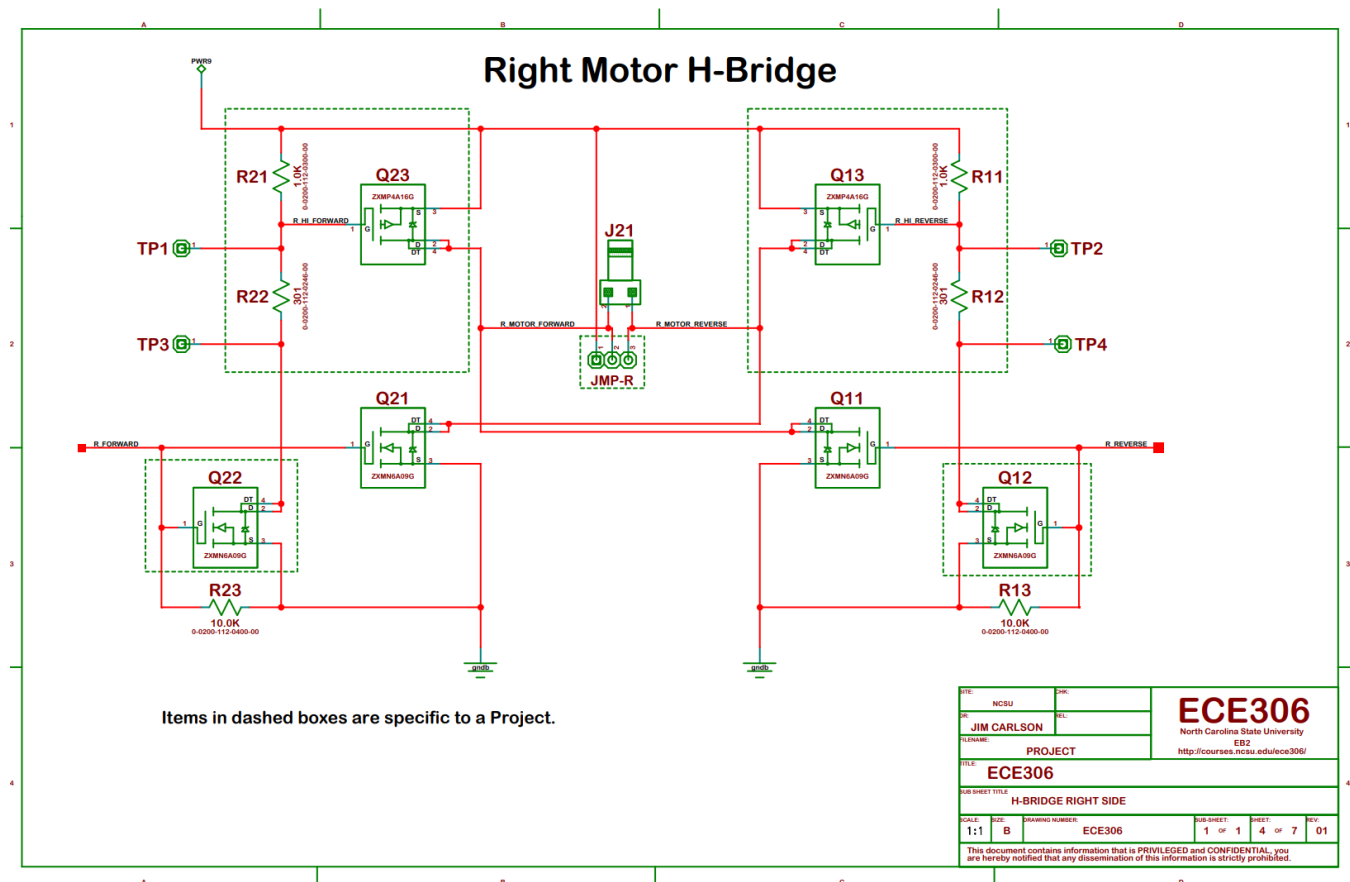
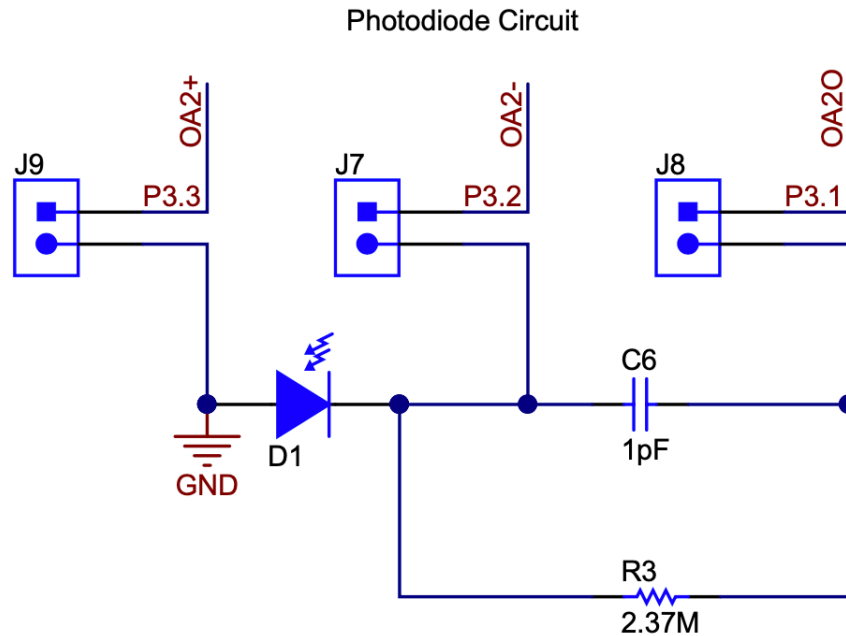


Figure 13 Right Motor H-Bridge

4.6. Emitter/Detector Board

The emitter and the detectors are soldered on a photodiode circuit board. The emitter, detectors and the board are connected back to the control board through pins.



6. Test Process

Throughout the project in order to ensure the functionality of every new part, it must undergo several tests in order to be considered operational.

6.1. Power System Test

This test was performed to ensure the power board was operating at the correct voltage. Using the Analog Discovery 2, we supplied 4.5-5 volts to the board and measured 3.3 – 3.6 volts. The measured voltages were expected due to the maximum current of 20 mA.

6.2. Power Circuit Test

This test ensures that the power board is connected by checking the resistance between two pins, J0 and J5. With the battery pack connected at J10 the switch was turned on and off. Using the Analog Discovery 2, we verified the resistance between J0 (ground) and J5 (5 volts) was approximately 165 ohms.

6.3. Voltage Verification Test

This test was performed to check that the connected batteries were supplying the appropriate voltage to the power supply board. Using 4 AA batteries in the battery pack and the pack connected to the

power connector at J10 the switch was turned on and off manually. The voltage measured between pin J0 and ground was approximate 6.3 volts for new batteries

6.4. P-FET and N-FET Test

The battery is connected, and the power switch is set to on. Using code, the tester writes a function that sets R_FORWARD to off and uses a voltmeter to check that test points one and three have a battery voltage. R_FORWARD is then set to on and the voltmeter shows that test point one is half the battery voltage while test point three is zero volts. Then R_FORWARD is changed to R_REVERSE which is set to off. The tester checks test point two and test point four after setting R_REVERSE to on. Then R_REVERSE is changed to L_FORWARD which is set to off. The tester checks test point five and test point seven after setting L_FORWARD to on. Then L_FORWARD is changed to L_REVERSE which is set to off. The tester checks test point six and test point eight after setting L_REVERSE to on.

6.5. Emitter/Detector Test

The tester runs code that will convert HEX values to BCD and displays these values on the LCD screen. After turning on the detectors and running this code. The tester should note that when the detectors are exposed to a black surface, they value displayed is greater than 400 and under 400 over a white surface. These values are verified to be correct by using the watch window in IAR while coding.

6.6. Baud Rate Test

The tester uses code to change to baud rate between 115,200 Hz and 460,800 Hz. After these changes are applied, the tester applies two pin jumpers in the two configurations shown below. Both UCA0 and UCA1 are set to 115,200 Hz first and the tester uses the first configuration. Using an oscilloscope, the tester can confirm that the symbols have been transmitted at this frequency. The tester then changes the configuration two and checks that 115,200 Hz works as well. The baud rate is then changed to 460,800 Hz and the test is repeated for both configurations to ensure this transmission is successful as well.

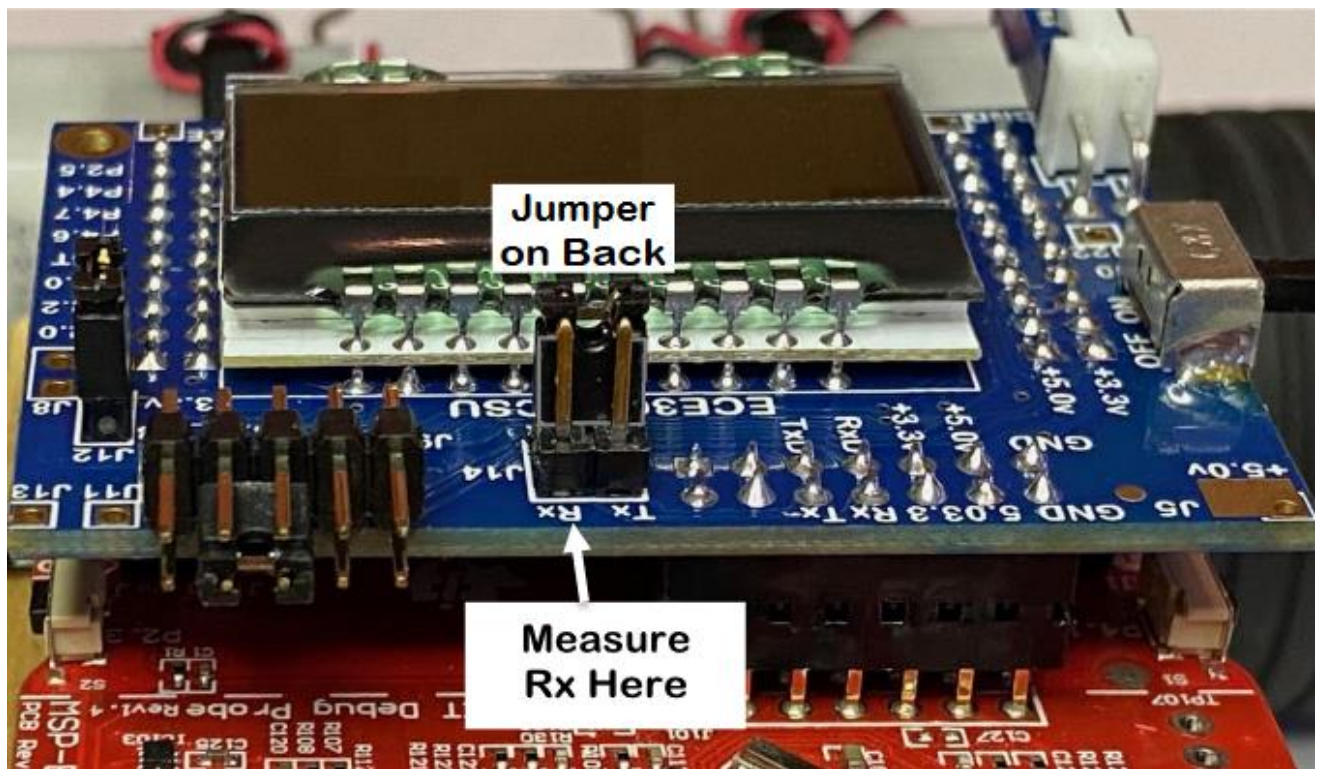


Figure 14 Configuration 1

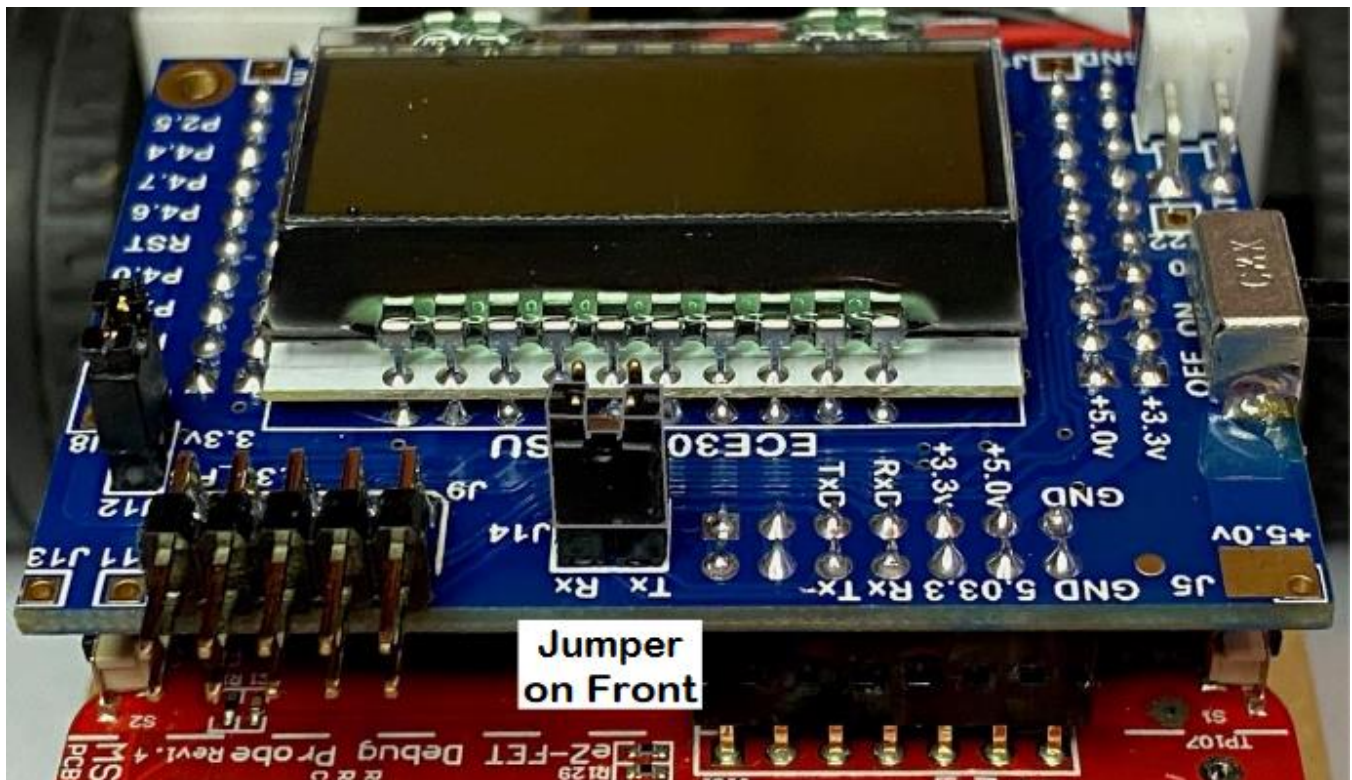


Figure 15 Configuration 1

7. Software

The software is configured using a modular approach. When powered on, the software will first initialize the ports that will be used to initialize the clocks, the timers, the LCD and the switches. Full power up is indicated when the LCD Backlight turns on and the green and red LEDs begin to flash.

7.1. Main

The main block runs all the required initialization upon start, which includes ports initialization, clocks initialization, conditions initialization, ADC and DAC initialization, LCD and display initialization, and lastly, serial ports initialization. Then in the project 8, a timer variable is derived from the Time_Sequence to execute the required delay time. We'll use this to show the welcome/waiting screen for 1 seconds upon start. The car will wait for switches to be pressed by the user to change its transmission baud rate, which will be set to 460800 if switch 1 is pressed and will be set to 115200 if switch 2 is pressed. After changing the baud rate, the car will wait for user inputs. It'll show any message it received from its receiver on the LCD display and send them back out through its transmitter. Change of baud rate can be done at any time.

7.2. Initialization

The initialization block contains an assortment of functions. This includes, ports, clocks, conditions, timers, LCD, ADC, DAC, and Serial communications. First it will run the configuration for all ports 1-6, defining their functions and input/output directions. Then it will initialize the clock that will be used by the processor. Then it will set all relevant conditions to 0 or an equivalent. Then we initialize timers B0 and B3. This function determines the clock sources and time intervals of each timer that we are going to be using. Then we will initialize the LCD display by clearing it and writing the splash screen to the display. When the ADC is initialized it defines the clock speed, resolution, mode, sample and hold, trigger mode, and connect reference voltage. When the DAC is initialized it will set the DAC data value and with select Vref to be the DAC reference and then will initialize MUX control and then finally will enable the conversions. Finally, for serial communications it will establish the desired baud rate that we have chosen, from there it will enable the receiver and transmitter interrupts to prepare for any commands that are sent to the board.

7.3. Timers

This section contains two files, interrupt_timers.c and timers.c. Interrupt timers contains two functions, Timer0_B0_ISR and Timer0_B1_ISR which control timer B0 for the first and zeroth capture control registers at specified times. Timer0_B0_ISR toggles the backlight based on the time. In timers.c the timers B0_0, B0_1, B0_2 and B2 are initialized by calling the two functions Init_Timers and Init_Timer_B3. Init_Timers uses the sub-main clock as the source and divides the clock by 2 then divides the clock again by 8. The overflow interrupts are enabled. For function Init_Timer_B3 the sub-main clock is used in the up mode and a pulse width modulation period is defined for the capture control register zero. All wheel speeds are set to off using the pulse-width modulation values.

7.4. ADC and Interrupt Files

This section contains several different functions. Everything from Hex to BCD conversion, to receiving reading from the IR emitters and detectors. The interrupts would handle various functions such as updating the display. Every 200msec, an interrupt would be run to check to see if there were updates that needed to be pushed to the display. Another interrupt also handled switch debounce. In order to prevent the switch from going off several times after being pressed an ISR was triggered that would disable and re-enable the switch after a set delay. There were also ISR that would handle switch interactions. When a switch was pressed the ISR would trigger, running the command/commands that were placed inside of the ISR subsequently triggering the debounce ISR.

8. Flow Chart

The following flow chart shows the structure of the code

8.1. Main Blocks

The main block is initiated after the initial power up of the system. Within the main function, it first calls other function to initialize the ports, clocks, condition, timers, and the LCD. Then the code enters a continuous while loop that allows for interrupts to take over and run other functions.

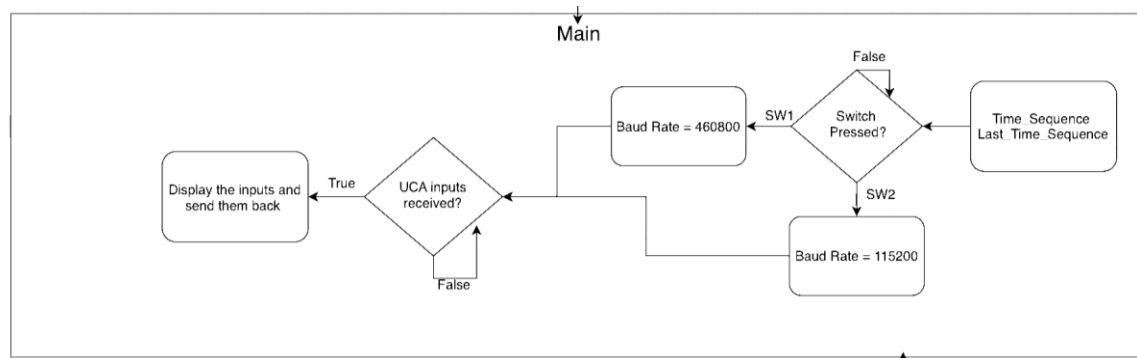


Figure 16 Main Block Flowchart

8.2. Initialization Blocks

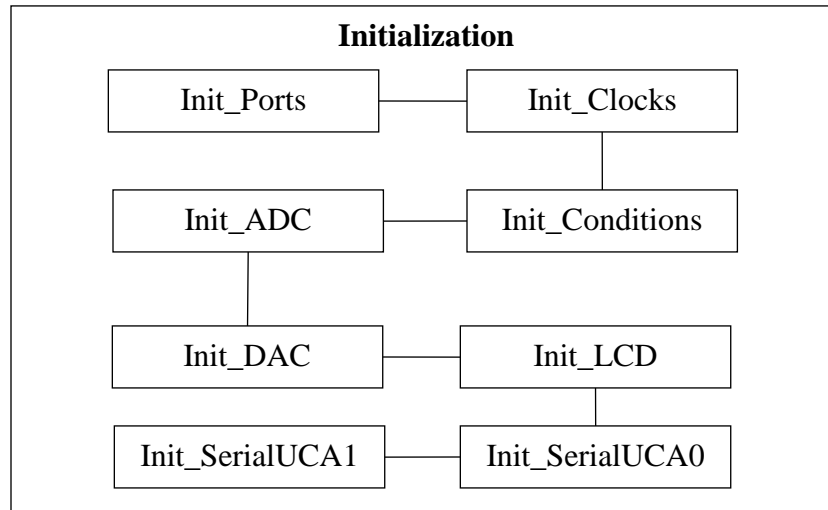


Figure 17 Initialization block flowchart

8.3. Timer Blocks

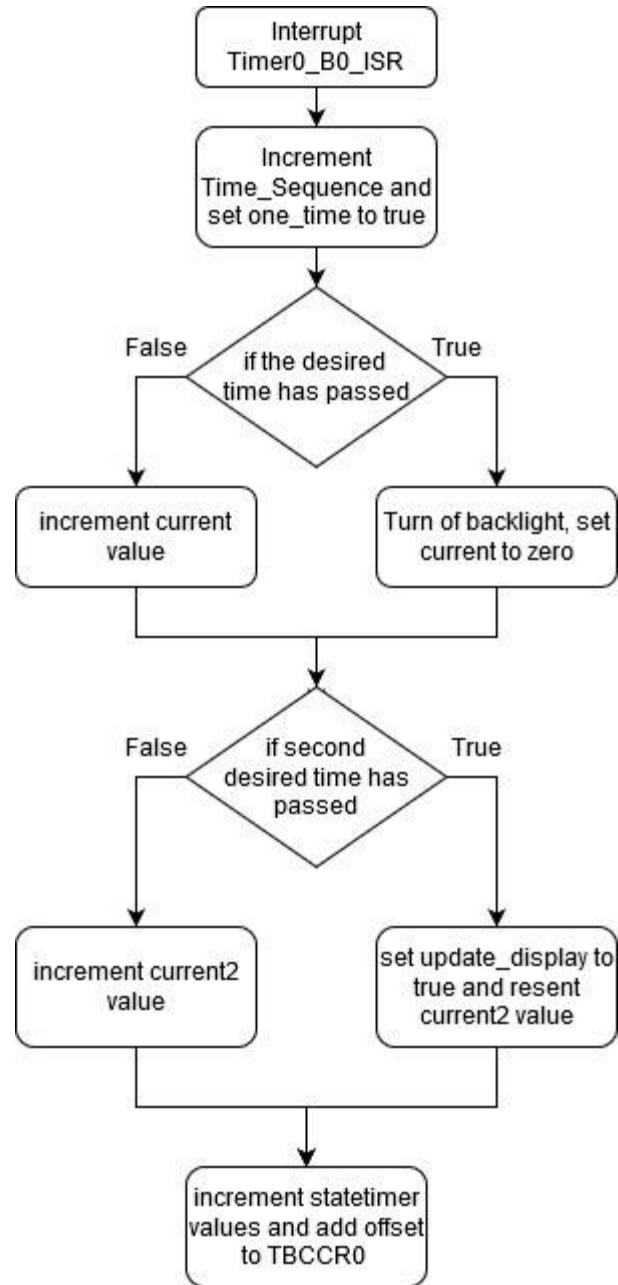


Figure 16 Function Timer0_B0_ISR Flowchart

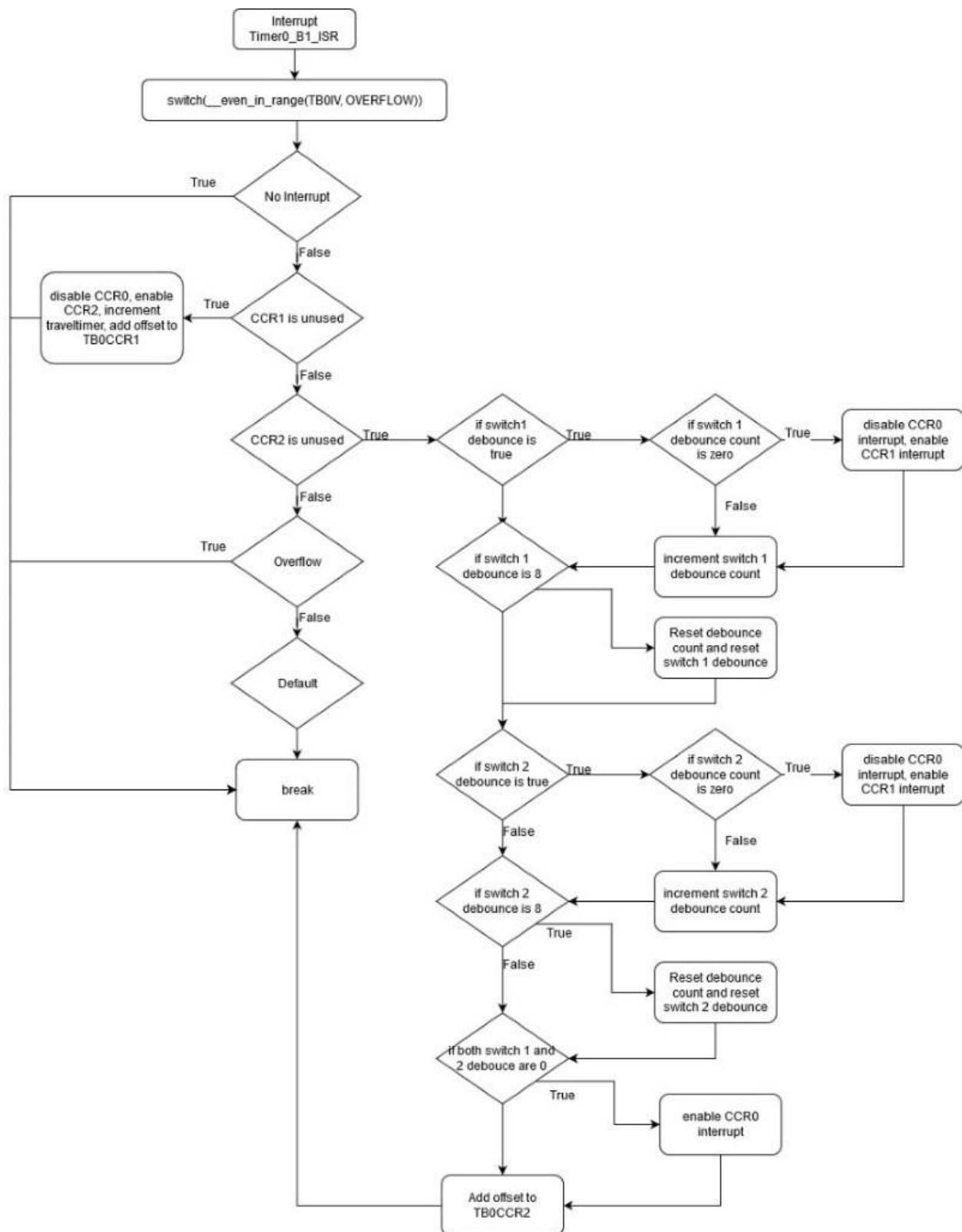


Figure 17 Function Timer0_B1_ISR Flowchart

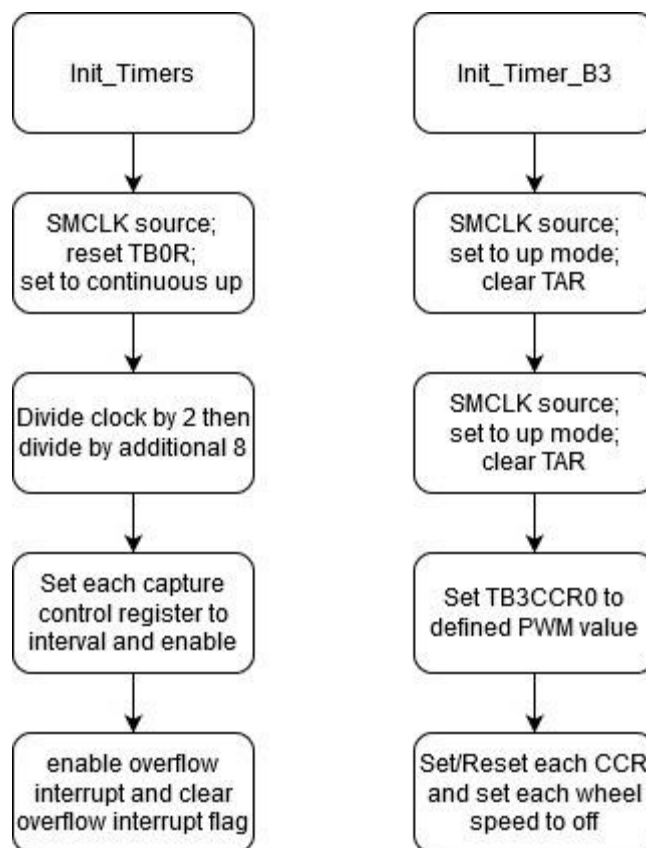


Figure 18 Function Init_Timers and Init_Timer_B3 Flowchart

9. Software Listing

This is just a printout of the actual code, with each file in it's own section.

9.1. Main.c

```

//-----
//
// Description: This file contains the Main Routine - "While" Operating System
//
//
// Zhiming Dai
// Oct 2020
// Built with IAR Embedded Workbench Version: V7.10.2 (8.0.6.4651)
//-----
#include "msp430.h"
#include "functions.h"

```

```
#include <string.h>
#include "macros.h"

// Global Variables
char display_adc;
char state;
//int count = 0;
unsigned char display_mode;
volatile unsigned int wheel_run;

extern volatile unsigned char update_display;
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
extern char pwm;
volatile char slow_input_down;
extern char display_line[Display_Roles][Display_Columns];
extern char *display[Display_Roles];
extern unsigned int wheel_on_count;
extern char wheels_on;
extern unsigned int running;
extern char wheel_direction;
extern char number_of_runs;
extern char move_state;
extern char next_state;
extern char last_state;
extern volatile unsigned char display_changed;
extern volatile unsigned int update_display_count;
extern volatile unsigned int SW2_display;
extern volatile unsigned int SW1_display;
extern unsigned int BLACK;
extern char iot_start;
```

```

/--time variable
unsigned char Last_Time_Sequence = 0;
unsigned char cycle_time; // is a new time base used to control making shapes
unsigned char time_change; // is an identifier that a change has occurred
unsigned char event = NONE;
unsigned char delay_start;
unsigned char left_motor_count;
unsigned char right_motor_count;
unsigned char segment_count;
unsigned int wheel_timer = 0;
unsigned int timer = 0;

```

```

void main(void){
//-----
// Main Program
// This is the main routine for the program. Execution of code starts here.
// The operating system is Back Ground Fore Ground.
//
//-----
Init_Ports();           // Initialize Ports
// Disable the GPIO power-on default high-impedance mode to activate
// previously configured port settings
PM5CTL0 &= ~LOCKLPM5;
Init_Clocks();          // Initialize Clock System
Init_Conditions();      // Initialize Variables and Initial Conditions
Init_Timers();          // Initialize Timers
Init_ADC();
Init_DAC();
Init_LCD();             // Initialize LCD           // Initialize Switches

```



```

display_mode = ORIGINAL;
display_adc = 1;
//
// strcpy(display_line[FIRST], " ECE-306 ");
// update_string(display_line[FIRST], FIRST);
// strcpy(display_line[SECOND], " Voltages ");
// update_string(display_line[SECOND], SECOND);
// strcpy(display_line[THIRD], "BAT  CONV");
// update_string(display_line[THIRD], THIRD);
// strcpy(display_line[FORTH], "      ");
// update_string(display_line[FORTH], FORTH);
// strcpy(display_line[3], "L  R  ");
// update_string(display_line[3], 3);
// display_changed = 1;
update_display = ON;
running = OFF;

//-----
// Begining of the "While" Operating System
//-----

while(ALWAYS) {           // Can the Operating system run

    if(Last_Time_Sequence != Time_Sequence){
        Last_Time_Sequence = Time_Sequence;
        cycle_time++;
        time_change = 1;

        if (Last_Time_Sequence % 2 == 0){
            timer = timer + 1;
        }
        if (Last_Time_Sequence % 1 == 0){

```

```

        wheel_timer = wheel_timer + 1;
    }
}
Display_Process();

```

```

if (timer <= 10){
    BLACK = 0;
}
else if (timer <= 500){
    Forward_On_Slow();
    if (BLACK){
        Wheel_Off();
        timer = 500;
    }
}
else if (timer <= 530){
    Wheel_Off();
}
else if (timer <= 535)
{
    Wheel_Off();
    P6OUT |= L_FORWARD;
}
else {
    Wheel_Off();
    break;
}
}
}

```

9.2. System_Init.c

```

//=====
=====

```

```
// File Name : init.c
//
// Description: This file contains the initialization of conditions, timers,
// DAC, ADC, and REF
//
// Author: William Brazil
// Date: October 2020
// Compiler: Built with IAR Embedded Workbench Version: 7.20.1
//=====
=====

#include "macros.h"
#include "msp430.h"
#include "functions.h"
#include <string.h>

extern char display_line[DISPLAY_ROWS][DISPLAY_COLUMNS];
extern char *display[DISPLAY_ROWS];
extern volatile unsigned int update_display_count;
extern volatile unsigned char update_display;
extern volatile unsigned int Time_Sequence;
unsigned int wheel_on_count;
char wheels_on;
extern volatile unsigned int update_display_count;

void Init_Conditions(void){

//=====
=====

/*int i;

for(i=INIT;i<11;i++){
    display_line[LINE1][i] = RESET_STATE;
```

```

    display_line[LINE2][i] = RESET_STATE;
    display_line[LINE3][i] = RESET_STATE;
    display_line[LINE4][i] = RESET_STATE;
}
display_line[LINE1][TEN] = INIT;
display_line[LINE2][TEN] = INIT;
display_line[LINE3][TEN] = INIT;
display_line[LINE4][TEN] = INIT;

display[LINE1] = &display_line[LINE1][INIT];
display[LINE2] = &display_line[LINE2][INIT];
display[LINE3] = &display_line[LINE3][INIT];
display[LINE4] = &display_line[LINE4][INIT];
update_display = FALSE;
update_display_count = INIT;
*/

// Interrupts are disabled by default, enable them.
enable_interrupts();

wheels_on = FALSE;
wheel_on_count = INIT;

//=====
=====
}

void Init_Timers(void){
    //Init_Timer_A0();
    //Init_Timer_A1();
    //Init_Timer_A2();
    //Init_Timer_A3();
    Init_Timer_B0();
    //Init_Timer_B1();

```

```

Init_Timer_B3();
}

void Init_ADC(void){
//-----
// V_DETECT_L (0x04) // Pin 2 A2
// V_DETECT_R (0x08) // Pin 3 A3
// V_THUMB (0x20) // Pin 5 A5
//-----
// ADCCTL0 Register
    ADCCTL0 = 0; // Reset
    ADCCTL0 |= ADCSHT_2; // 16 ADC clocks
    ADCCTL0 |= ADCMSC; // MSC
    ADCCTL0 |= ADCON; // ADC ON

// ADCCTL1 Register
    ADCCTL2 = 0; // Reset
    ADCCTL1 |= ADCSHS_0; // 00b = ADCSC bit
    ADCCTL1 |= ADCSHP; // ADC sample-and-hold SAMPCON signal from sampling timer.
    ADCCTL1 &= ~ADCISSH; // ADC invert signal sample-and-hold.
    ADCCTL1 |= ADCDIV_0; // ADC clock divider - 000b = Divide by 1
    ADCCTL1 |= ADCSSEL_0; // ADC clock MODCLK
    ADCCTL1 |= ADCCONSEQ_0; // ADC conversion sequence 00b = Single-channel single-
conversion
// ADCCTL1 & ADCBUSY identifies a conversion is in process

// ADCCTL2 Register
    ADCCTL2 = 0; // Reset
    ADCCTL2 |= ADCPDIV0; // ADC pre-divider 00b = Pre-divide by 1
    ADCCTL2 |= ADCRES_2; // ADC resolution 10b = 12 bit (14 clock cycle conversion time)
    ADCCTL2 &= ~ADCFD; // ADC data read-back format 0b = Binary unsigned.
    ADCCTL2 &= ~ADCSR; // ADC sampling rate 0b = ADC buffer supports up to 200 ksp/s

```

```

// ADCMCTL0 Register
ADCMCTL0 |= ADCSREF_0; // VREF - 000b = { VR+ = AVCC and VR- = AVSS }
ADCMCTL0 |= ADCINCH_5; // V_THUMB (0x20) Pin 5 A5

ADCIE |= ADCIE0; // Enable ADC conv complete interrupt
ADCCTL0 |= ADCENC; // ADC enable conversion.
ADCCTL0 |= ADCSC; // ADC start conversion.
}

void Init_DAC(void){
    DAC_data = 1000; // Value between 0x000 and 0x0FFF
    SAC3DAT = DAC_data; // Initial DAC data
    SAC3DAC = DACSREF_1; // Select int Vref as DAC reference
    SAC3DAC |= DACLSEL_0; // DAC latch loads when DACDAT written
    SAC3DAC |= DACIE; // generate an interrupt
    SAC3DAC |= DACEN; // Enable DAC
    SAC3OA = NMUXEN; // SAC Negative input MUX control
    SAC3OA |= PMUXEN; // SAC Positive input MUX control
    SAC3OA |= PSEL_1; // 12-bit reference DAC source selected
    SAC3OA |= NSEL_1; // Select negative pin input
    SAC3OA |= OAPM; // Select low speed and low power mode
    SAC3PGA = MSEL_1; // Set OA as buffer mode
    SAC3OA |= SACEN; // Enable SAC
    SAC3OA |= OAEN; // Enable OA
}

void Init_REF(void){
    // Configure reference module
    PMMCTL0_H = PMMPW_H; // Unlock the PMM registers
    PMMCTL2 = INTREFEN; // Enable internal reference
    PMMCTL2 |= REFVSEL_2; // Select 2.5V reference
    while(!(PMMCTL2 & REFGENRDY)); // Poll till internal reference settles

```

}

9.3. Interrupt.c

9.3.1 Interrupt_timers.c

```
//-----
//
// Description: This file contains the interrupts for timers B0_0, B0_1,
//   and B0_2.
//
//
// Sloane Cox
// Jan 2020
// Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//-----

//-----

#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"

// Global Variables
unsigned int current;
unsigned int current2;
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
unsigned int switch1_debounce;
unsigned int switch2_debounce;
unsigned int switch1_db_count;
unsigned int switch2_debounce_count;
unsigned int traveltimer = 0;
int waiting = 1;
int statetimer = 0;

// Function Prototypes
void both_wheels_off(void);
```

```
//=====
// Function name: Timer0_B0_ISR
//
// Description: This function is an interrupt for timer B0_0 that updates the
//      display and backlight at desired intervals.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: Time_Sequence
//      current
//      current2
//      update_display
//      statetimer
//
// Author: Sloane Cox
// Date: Spring 2020
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//=====
```

```
#pragma vector = TIMER0_B0_VECTOR
__interrupt void Timer0_B0_ISR(void){
//-----
// TimerB0 0 Interrupt handler
//-----

    Time_Sequence++;
    one_time = TRUE;
    if (current == DESIRED_NOI) {
        P6OUT &= ~LCD_BACKLITE;
        current = RESET;
    }
    else {
        current++;
    }
    if (current2 == WAIT_MSEC) {
        update_display = TRUE;
        current2 = RESET;
    }
    else {
```



```

        current2++;
    }
    statetimer++;
    TB0CCR0 += TB0CCR0_INTERVAL;    // Add Offset to TBCCR0

//-----

}

//=====

// Function name: Timer0_B1_ISR
//
// Description: This function enables or disables capture control registers
// associated with timers B0_1 and B0_2 by debouncing the depressed switch.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: switch1_debounce
//          switch2_debounce
//          switch1_db_count
//          switch2_db_count
//
// Author: Sloane Cox
// Date: Spring 2020
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//=====

#pragma vector=TIMER0_B1_VECTOR
__interrupt void TIMER0_B1_ISR(void){
//-----

// TimerB0 1-2, Overflow Interrupt Vector (TBIV) handler
//-----

switch(__even_in_range(TB0IV,OVERFLOW)){
case NO_INTERRUPT: break;    // No interrupt
case CCR1_UNUSED:    // CCR1 not used
    TB0CCTL0 &= ~CCIE;    // CCR0 disable interrupt
    TB0CCTL2 |= CCIE;    // CCR2 enable interrupt
    traveltimer++;
    TB0CCR1 += TB0CCR1_INTERVAL;    // Add Offset to TBCCR1

```

```

    break;
case CCR2_UNUSED:           // CCR2 not used
    if (switch1_debounce == TRUE) {
        if (switch1_db_count == 0) {
            TB0CCTL0 &= ~CCIE;      // CCR0 disable interrupt
            TB0CCTL1 |= CCIE;       // CCR1 enable interrupt
        }
        switch1_db_count++;
    }
    if (switch1_db_count == 8) {
        switch1_db_count = RESET;
        switch1_debounce = RESET;
    }
    if (switch2_debounce == TRUE) {
        if (switch2_db_count == 0) {
            TB0CCTL0 &= ~CCIE;      // CCR0 disable interrupt
            TB0CCTL1 |= CCIE;       // CCR1 enable interrupt
        }
        switch2_db_count++;
    }
    if (switch2_db_count == 8) {
        switch2_db_count = RESET;
        switch2_debounce = RESET;
    }
    if (switch1_debounce == 0 && switch2_debounce == 0) {
        TB0CCTL0 |= CCIE;          // CCR0 enable interrupt
    }
    TB0CCR2 += TB0CCR2_INTERVAL;    // Add Offset to TBCCR2
    break;
case OVERFLOW:              // overflow
    break;
default: break;
}
//-----
}

```

9.3.2 Timers.c

```

//-----
//
// Description: This file contains the initialization for timer

```

```
// B0_0, B0_1, B0_2, and B3.
//
//
// Sloane Cox
// Jan 2020
// Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//-----

//-----

#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"

//=====
=====
// Function name: Init_Timers
//
// Description: This function initializes timer B0_0, B0_1, B0_2, and overflow.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no variables used
//
// Author: Sloane Cox
// Date: Spring 2020
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//=====
=====

//-----
```

// Timer B0 initialization sets up both B0_0, B0_1-B0_2 and overflow

void Init_Timers(void) {

TB0CTL = TBSSSEL__SMCLK; // SMCLK source

TB0CTL |= TBCLR; // Resets TB0R, clock divider, count direction

TB0CTL |= MC__CONTINUOUS; // Continuous up

TB0CTL |= ID__2; // Divide clock by 2

TB0EX0 = TBIDEX__8; // Divide clock by an additional 8

TB0CCR0 = TB0CCR0_INTERVAL; // CCR0

TB0CCTL0 |= CCIE; // CCR0 enable interrupt

TB0CCR1 = TB0CCR1_INTERVAL; // CCR1

TB0CCTL1 |= CCIE; // CCR1 enable interrupt

TB0CCR2 = TB0CCR2_INTERVAL; // CCR2

TB0CCTL2 |= CCIE; // CCR2 enable interrupt

TB0CTL |= TBIE; // Enable Overflow Interrupt

TB0CTL &= ~TBIFG; // Clear Overflow Interrupt flag

}

//-----

//=====

// Function name: Init_Timer_B3

//

// Description: This function initializes timer B3.

//

// Passed : no variables passed

// Locals: no variables declared

// Returned: no values returned

// Globals: no variables used

```
//
// Author: Sloane Cox
// Date: Spring 2020
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//=====
=====
void Init_Timer_B3(void) {
//-----
// SMCLK source, up count mode, PWM Right Side
// TB3.1 P6.0 R_FORWARD
// TB3.2 P6.1 L_FORWARD
// TB3.3 P6.2 R_REVERSE
// TB3.4 P6.3 L_REVERSE
//-----
TB3CTL = TBSEL__SMCLK;      // SMCLK
TB3CTL |= MC__UP;          // Up Mode
TB3CTL |= TBCLR;           // Clear TAR

TB3CCR0 = WHEEL_PERIOD;     // PWM Period

TB3CCTL1 = OUTMOD_7;        // CCR1 reset/set
RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM duty cycle

TB3CCTL2 = OUTMOD_7;        // CCR2 reset/set
LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM duty cycle

TB3CCTL3 = OUTMOD_7;        // CCR3 reset/set
RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM duty cycle

TB3CCTL4 = OUTMOD_7;        // CCR4 reset/set
LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM duty cycle
//-----
}
```

9.3.4 interrupt_eUSCI.c

```
void out_character(char character);
```

```
#pragma vector=EUSCI_A0_VECTOR
__interrupt void eUSCI_A0_ISR(void){
    unsigned int temp;
    switch(__even_in_range(UCA0IV,0x08)){
        case 0: // Vector 0 - no interrupt
            break;
        case 2: // Vector 2 - RXIFG
            if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
                usb_rx_ring_wr = BEGINNING; // Circular buffer back to beginning
            }
            temp = usb_rx_ring_wr++;
            USB_Char_Rx[temp] = UCA0RXBUF; // RX -> USB_Char_Rx character
            out_character(USB_Char_Rx[temp]);
```

```
        display_line[0][0] = ' ';
        display_line[0][1] = 'R';
        display_line[0][2] = 'e';
        display_line[0][3] = 'c';
        display_line[0][4] = 'e';
        display_line[0][5] = 'i';
        display_line[0][6] = 'v';
        display_line[0][7] = 'e';
        display_line[0][8] = 'd';
        display_line[0][9] = ' ';
```

```
        display_changed = 1;
        update_string(display_line[FIRST], FIRST);
```

```
        break;
        case 4: // Vector 4 - TXIFG
            switch(UCA0_index++){
                case 0:
                case 1:
```

```


    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 8:
        UCA0TXBUF = test_command[UCA0_index];
        break;
    case 9:
        UCA0TXBUF = 0x0D;
        break;
    case 10:          // Vector 0 - no interrupt
        UCA0TXBUF = 0x0A;
        break;
    default:
        UCA0IE &= ~UCTXIE;    // Disable TX interrupt
        break;

}
break;
default: break;
}
}

#pragma vector=EUSCI_A1_VECTOR
__interrupt void eUSCI_A1_ISR(void){
    unsigned int temp;
    switch(__even_in_range(UCA1IV,0x08)){
        case 0: // Vector 0 - no interrupt
            break;
        case 2: // Vector 2 - RXIFG
            if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
                usb_rx_ring_wr = BEGINNING; // Circular buffer back to beginning
            }
            temp = usb_rx_ring_wr++;
            USB_Char_Rx[temp] = UCA1RXBUF; // RX -> USB_Char_Rx character

```

```

        out_character(USB_Char_Rx[temp]);
    break;
case 4: // Vector 4  TXIFG
    switch(UCA1_index++){
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
            UCA1TXBUF = test_command[UCA0_index];
            break;
        case 9:
            UCA1TXBUF = 0x0D;
            break;
        case 10: // Vector 0 - no interrupt
            UCA1TXBUF = 0x0A;
            break;
        default:
            UCA1IE &= ~UCTXIE; // Disable TX interrupt
            break;
    }
    break;
default: break;
}
}

```

9.4. ADC.c

```

//
=====
=====
//

```



```
// Description: This file contains the ADC Initialization and Hex to BCD Conversion
//
// William Brazil
// Nov 2020
// Built with IAR Embedded Workbench Version: V7.20.1
//
```

```
=====
=====
```

```
#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"
```

```
char adc_char[DISPLAY_COLUMNS] = "      ";
```

```
void HEXtoBCD(int hex_value, int side){
```

```
//=====
=====
```

```
// Function Name: HEXtoBCD
```

```
//
```

```
// Description: Convert a Hex number to a BCD for display on an LCD or monitor
```

```
//
```

```
// Local Variables: value, hex_value
```

```
// Global Variables: adc_char[DISPLAY_COLUMNS]
```

```
//
```

```
// William Brazil
```

```
// Oct 2020
```

```
// Built with IAR Embedded Workbench Version: V(7.20.1)
```

```
//=====
=====
```

```

int value = INIT;

switch(side){
case L:
    adc_char[Thousands] = ZERO;
    while (hex_value >= ONETHOUSAND){
        hex_value = hex_value - ONETHOUSAND;
        value++;
        adc_char[Thousands] = ASCII_CONV + value;
    }
    value = INIT;
    adc_char[Hundreds] = ZERO;
    while (hex_value >= ONEHUNDRED){
        hex_value = hex_value - ONEHUNDRED;
        value++;
        adc_char[Hundreds] = ASCII_CONV + value;
    }
    value = INIT;
    adc_char[Tens] = ZERO;
    while (hex_value >= TEN){
        hex_value = hex_value - TEN;
        value++;
        adc_char[Tens] = ASCII_CONV + value;
    }
    adc_char[Ones] = ZERO;
    adc_char[Ones] = ASCII_CONV + hex_value;
    break;
case R:
    adc_char[Thousands+6] = ZERO;
    while (hex_value >= ONETHOUSAND){
        hex_value = hex_value - ONETHOUSAND;
        value++;

```

```

    adc_char[Thousands+6] = ASCII_CONV + value;
}
value = INIT;
adc_char[Hundreds+6] = ZERO;
while (hex_value >= ONEHUNDRED){
    hex_value = hex_value - ONEHUNDRED;
    value++;
    adc_char[Hundreds+6] = ASCII_CONV + value;
}
value = INIT;
adc_char[Tens+6] = ZERO;
while (hex_value >= TEN){
    hex_value = hex_value - TEN;
    value++;
    adc_char[Tens+6] = ASCII_CONV + value;
}
adc_char[Ones+6] = ZERO;
adc_char[Ones+6] = ASCII_CONV + hex_value;
break;
default: break;
}
}

```

```

void ADC_Process(void){

```

```

//=====
=====

```

```

// Function Name: ADC_Process

```

```

//

```

```

// Description: Holds program in place while ADC conversions are being made

```

```

//

```

```

// Local Variables: None

```

```

// Global Variables: None

```

```
//  
// William Brazil  
// Oct 2020  
// Built with IAR Embedded Workbench Version: V(7.20.1)  
  
//=====
```

```
while(ADCCTL1 && ADCBUSY);  
ADCCTL0 |= ADCENC;      // ADC enable conversion.  
ADCCTL0 |= ADCSC;       // ADC start conversion.  
}
```

9.5. Clocks.c

```
//  
=====
```

//

// Description: This file contains the Clock Initialization

//

// Jim Carlson

// Jan 2016

// Built with IAR Embedded Workbench Version: V7.12.1

//

```
=====
```

```
#include "functions.h"  
#include "msp430.h"  
#include "macros.h"
```

```
void Init_Clocks(void);  
void Software_Trim(void);
```

```
void Init_Clocks(void){
```

```
// -----
// Clock Configurtaions
// This is the clock initialization for the program.
// Initial clock configuration, runs immediately after port configuration.
// Disables 1ms watchdog timer,
// Configure MCLK for 8MHz and XT1 sourcing ACLK and FLLREF.
//
// Description: Configure ACLK = 32768Hz,
//             MCLK = DCO + XT1CLK REF = 8MHz,
//             SMCLK = MCLK/2 = 4MHz.
// Toggle LED to indicate that the program is running.
//
// -----

WDTCTL = WDTPW | WDTHOLD; // Disable watchdog

do{
    CSCTL7 &= ~XT1OFFG;    // Clear XT1 fault flag
    CSCTL7 &= ~DCOFFG;     // Clear DCO fault flag
    SFRIFG1 &= ~OFIFG;
} while (SFRIFG1 & OFIFG); // Test oscillator fault flag
__bis_SR_register(SCG0); // disable FLL

CSCTL1 = DCOFTRIMEN_1;
CSCTL1 |= DCOFTRIM0;
CSCTL1 |= DCOFTRIM1;    // DCOFTRIM=3
CSCTL1 |= DCORSEL_3;    // DCO Range = 8MHz

CSCTL2 = FLLD_0 + 243;  // DCODIV = 8MHz

CSCTL3 |= SELREF__XT1CLK; // Set XT1CLK as FLL reference source
__delay_cycles(3);
__bic_SR_register(SCG0); // enable FLL
```

```

Software_Trim();      // Software Trim to get the best DCOFTRIM value

CSCTL4 = SELA__XT1CLK;  // Set ACLK = XT1CLK = 32768Hz
CSCTL4 |= SELMS__DCOCLKDIV; // DCOCLK = MCLK and SMCLK source

CSCTL5 |= DIVM__1;      // MCLK = DCOCLK = 8MHZ,
CSCTL5 |= DIVS__1;      // SMCLK = DCOCLK = 8MHz

PM5CTL0 &= ~LOCKLPM5;  // Disable the GPIO power-on default high-impedance mode
                        // to activate previously configured port settings
}

void Software_Trim(void){
// --COPYRIGHT--,BSD_EX
// Copyright (c) 2014, Texas Instruments Incorporated
// All rights reserved.
    unsigned int oldDcoTap = 0xffff;
    unsigned int newDcoTap = 0xffff;
    unsigned int newDcoDelta = 0xffff;
    unsigned int bestDcoDelta = 0xffff;
    unsigned int csCtl0Copy = 0;
    unsigned int csCtl1Copy = 0;
    unsigned int csCtl0Read = 0;
    unsigned int csCtl1Read = 0;
    unsigned int dcoFreqTrim = 3;
    unsigned char endLoop = 0;
    do{
        CSCTL0 = 0x100;          // DCO Tap = 256
        do{
            CSCTL7 &= ~DCOFFG;    // Clear DCO fault flag
        }while (CSCTL7 & DCOFFG); // Test DCO fault flag
        // Wait FLL lock status (FLLUNLOCK) to be stable

```

```

// Suggest to wait 24 cycles of divided FLL reference clock
__delay_cycles((unsigned int)3000 * MCLK_FREQ_MHZ);
while((CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)) &&
      ((CSCTL7 & DCOFFG) == 0));
csCtl0Read = CSCTL0;          // Read CSCTL0
csCtl1Read = CSCTL1;          // Read CSCTL1
oldDcoTap = newDcoTap;        // Record DCOTAP value of last time
newDcoTap = csCtl0Read & 0x01ff; // Get DCOTAP value of this time
dcoFreqTrim = (csCtl1Read & 0x0070)>>4; // Get DCOFTRIM value
if(newDcoTap < 256){          // DCOTAP < 256
    newDcoDelta = 256 - newDcoTap; // Delta value between DCPTAP and 256
    if((oldDcoTap != 0xffff) &&
        (oldDcoTap >= 256)){      // DCOTAP cross 256
        endLoop = 1;              // Stop while loop
    }else{
        dcoFreqTrim--;
        CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
    }
}else{                        // DCOTAP >= 256
    newDcoDelta = newDcoTap - 256; // Delta value between DCPTAP and 256
    if(oldDcoTap < 256){          // DCOTAP cross 256
        endLoop = 1;              // Stop while loop
    }else{
        dcoFreqTrim++;
        CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
    }
}
if(newDcoDelta < bestDcoDelta){ // Record DCOTAP closest to 256
    csCtl0Copy = csCtl0Read;
    csCtl1Copy = csCtl1Read;
    bestDcoDelta = newDcoDelta;
}

```

```

}while(endLoop == 0);          // Poll until endLoop == 1
CSCTL0 = csCtl0Copy;          // Reload locked DCOTAP
CSCTL1 = csCtl1Copy;          // Reload locked DCOFTRIM
while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked
}

```

9.6. Display.c

```

//
=====
=====
//
// Description: This file contains several function s regarding the operation of
//             the display
//
// William Brazil
// Nov 2020
// Built with IAR Embedded Workbench Version: V7.20.1
//
=====
=====

#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"

extern char display_line[DISPLAY_ROWS][DISPLAY_COLUMNS];
extern volatile unsigned char display_changed;
extern char adc_char[DISPLAY_COLUMNS];
extern volatile unsigned char update_display;
extern char string2display[ELEVEN];
extern char baudrate[TEN];
extern char process_buffer[50][2];

```



```
extern char ssid[ELEVEN];
extern char ip1[ELEVEN];
extern char ip2[ELEVEN];
extern char TCP_command[20];
extern char arrivedstring[ELEVEN];
extern char displaytimer[ELEVEN];
```

```
void Display(void){
    if(update_display){
        update_display = INIT;
        Display_Update(0,0,0,0);
    }
}
```

```
void BootText(void){
    strcpy(display_line[LINE1], "Connecting");
    update_string(display_line[LINE1], LINE1);
    strcpy(display_line[LINE2], "      ");
    update_string(display_line[LINE2], LINE2);
    strcpy(display_line[LINE3], "      ");
    update_string(display_line[LINE3], LINE3);
    strcpy(display_line[LINE4], "      ");
    update_string(display_line[LINE4], LINE4);
```

```
    display_changed = TRUE;
}
```

```
void Forward_Text(void){
    strcpy(display_line[LINE1], " Forward ");
    update_string(display_line[LINE1], LINE1);
```

```
    display_changed = TRUE;
```

```

}

void Reverse_Text(void){
    strcpy(display_line[LINE1], " Reverse ");
    update_string(display_line[LINE1], LINE1);

    display_changed = TRUE;
}

void Stop_Text(void){
    strcpy(display_line[LINE1], " STOP ");
    update_string(display_line[LINE1], LINE1);

    display_changed = TRUE;
}

void Right_Text(void){
    strcpy(display_line[LINE1], " RIGHT ");
    update_string(display_line[LINE1], LINE1);

    display_changed = TRUE;
}

void Left_Text(void){
    strcpy(display_line[LINE1], " LEFT ");
    update_string(display_line[LINE1], LINE1);

    display_changed = TRUE;
}

void clockwise_text(void){
    strcpy(display_line[LINE1], " Spin ");

```

```
update_string(display_line[LINE1], LINE1);
strcpy(display_line[LINE2], " Clockwise");
update_string(display_line[LINE2], LINE2);
strcpy(display_line[LINE3], "      ");
update_string(display_line[LINE3], LINE3);

display_changed = TRUE;
}
```

```
void counterclockwise_text(void){
    strcpy(display_line[LINE1], " Spin ");
    update_string(display_line[LINE1], LINE1);
    strcpy(display_line[LINE2], " Counter ");
    update_string(display_line[LINE2], LINE2);
    strcpy(display_line[LINE3], " Clockwise");
    update_string(display_line[LINE3], LINE3);

    display_changed = TRUE;
}
```

```
void Reset_Text(void){
    strcpy(display_line[LINE1], " NCSU ");
    update_string(display_line[LINE1], LINE1);
    strcpy(display_line[LINE2], " WOLFPACK ");
    update_string(display_line[LINE2], LINE2);
    strcpy(display_line[LINE3], " ECE306 ");
    update_string(display_line[LINE4], LINE4);

    display_changed = TRUE;
}
```

```
void emitter_state(int state){
```

```

switch(state){
case ON:
    strcpy(display_line[LINE2], "IR: ON  ");
    update_string(display_line[LINE2], LINE2);
    break;
case OFF:
    strcpy(display_line[LINE2], "IR: OFF  ");
    update_string(display_line[LINE2], LINE2);
    break;
default:
    break;
}
display_changed = TRUE;

}

```

```

void adc_line4(char location){
    strcpy(display_line[LINE3], "L  R  ");
    update_string(display_line[LINE3], LINE3);
    strcpy(display_line[LINE4],adc_char);
    update_string(display_line[LINE4], LINE4);

    display_changed = TRUE;
}

```

```

void clear_display(void){
    strcpy(display_line[LINE1], "      ");
    update_string(display_line[LINE1], LINE1);
    strcpy(display_line[LINE2], "      ");
    update_string(display_line[LINE2], LINE2);
    strcpy(display_line[LINE3], "      ");
    update_string(display_line[LINE3], LINE3);
}

```

```
strcpy(display_line[LINE4], "      ");  
update_string(display_line[LINE4], LINE4);
```

```
display_changed = TRUE;  
}
```

```
void display_received(void){  
    strcpy(display_line[LINE1], " Recieved ");  
    update_string(display_line[LINE1], LINE1);  
    strcpy(display_line[LINE4], string2display);  
    update_string(display_line[LINE4], LINE4);  
    display_changed = TRUE;  
}
```

```
void display_transmitted(void){  
    strcpy(display_line[LINE1], " Transmit ");  
    update_string(display_line[LINE1], LINE1);  
    strcpy(display_line[LINE2], string2display);  
    update_string(display_line[LINE2], LINE2);  
    display_changed = TRUE;  
}
```

```
void display_connected(void){  
    strcpy(display_line[LINE1], "Connected ");  
    update_string(display_line[LINE1], LINE1);  
    display_changed = TRUE;  
}
```

```
void display_ssid(void){  
    strcpy(display_line[LINE1], ssid);  
    update_string(display_line[LINE1], LINE1);  
    display_changed = TRUE;
```

```
}

void display_ip(void){
    strcpy(display_line[LINE1], " Waiting ");
    update_string(display_line[LINE1], LINE1);
    strcpy(display_line[LINE2], ip1);
    update_string(display_line[LINE2], LINE2);
    strcpy(display_line[LINE3], ip2);
    update_string(display_line[LINE3], LINE3);
    strcpy(display_line[LINE4], displaytimer);
    update_string(display_line[LINE4], LINE4);

    display_changed = TRUE;
}

void display_tcp(void){
    strcpy(display_line[LINE3], TCP_command);
    update_string(display_line[LINE3], LINE3);

    display_changed = TRUE;
}

void display_arrived(void){
    strcpy(display_line[LINE1], arrivedstring);
    update_string(display_line[LINE1], LINE1);

    display_changed = TRUE;
}

void display_timer(void){
    strcpy(display_line[LINE4], displaytimer);
```

```

update_string(display_line[LINE4], LINE4);

display_changed = TRUE;
}

```

9.7. Ports

9.7.1 Port1.c

```

//=====
==
// File Name : port1.c
//
// Description: This file contains the initialization of port 1
//
// Author: William Brazil
// Date: August 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
==
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"

void Init_Port1(void){
    P1OUT = LOW;           //P1 set Low
    P1DIR = OUTPUT;        //Set P1 direction to output

    //P1 Pin 0
    P1SEL0 &= ~RED_LED;    //RED_LED GPIO operation
    P1SEL1 &= ~RED_LED;    //RED_LED GPIO operation
    P1OUT &= ~RED_LED;      //Set LED off
    P1DIR |= RED_LED;       //RED_LED direction output

```

//P1 Pin 1

P1SEL0 |= A1_SEEED; //ADC input for A1_SEED

P1SEL1 |= A1_SEEED; //ADC input for A1_SEED

//P1 Pin 2

P1SEL0 |= V_DETECT_L; //ADC input for V_DETECT_L

P1SEL1 |= V_DETECT_L; //ADC input for V_DETECT_L

//P1 Pin 3

P1SEL0 |= V_DETECT_R; //ADC input for V_DETECT_R

P1SEL1 |= V_DETECT_R; //ADC input for V_DETECT_R

//P1 Pin 4 (NOT SURE)

P1SEL0 |= A4_SEEED; //ADC input for A4_SEEED

P1SEL1 |= A4_SEEED; //ADC input for A4_SEEED

//P1 Pin 5 (NOT SURE)

P1SEL0 |= V_THUMB; //ADC input for V_THUMB

P1SEL1 |= V_THUMB; //ADC input for V_THUMB

//P1 Pin 6

P1SEL0 |= UCA0RXD; //set UCA0RXD operation

P1SEL1 &= ~UCA0RXD; //set UCA0RXD operation

//P1 Pin 7

P1SEL0 |= UCA0TXD; //set UCA0TXD operation

P1SEL1 &= ~UCA0TXD; //set UCA0TXD operation

}

9.7.2 Port2.c

```
//=====
//
// File Name : port2.c
//
// Description: This file contains the initialization of port 2
//
// Author: William Brazil
// Date: August 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
//=====
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"

void Init_Port2(void){
//=====
=====
    P2OUT = LOW;      // P2 set Low
    P2DIR = OUTPUT;    // Set P2 direction to output

    //P2 Pin 0
    P2SEL0 &= ~P2_0;   // P2_0 GPIO operation
    P2SEL1 &= ~P2_0;   // P2_0 GPIO operation
    P2DIR &= ~P2_0;    // Direction = input

    //P2 Pin 1
    P2SEL0 &= ~IR_LED; // IR_LED GPIO operation
    P2SEL1 &= ~IR_LED; // IR_LED GPIO operation
    P2OUT &= ~IR_LED;  // Initial Value = OFF
    P2DIR |= IR_LED;   // Direction = output
```

//P2 Pin 2

P2SEL0 &= ~P2_2; // P2_2 GPIO operation

P2SEL1 &= ~P2_2; // P2_2 GPIO operation

P2DIR &= ~P2_2; // Direction = input

//P2 Pin 3

P2SEL0 &= ~SW2; // SW2 Operation

P2SEL1 &= ~SW2; // SW2 Operation

P2DIR &= ~SW2; // Direction = input

P2PUD |= SW2; // Configure pullup resistor

P2REN |= SW2; // Enable pullup resistor

P2IES |= SW2; // P2.0 Hi/Lo edge interrupt

P2IFG &= ~SW2; // Clear all P2.6 interrupt flags

P2IE |= SW2; // P2.6 interrupt enabled

//P2 Pin 4

P2SEL0 &= ~P2_4; // P2_4 GPIO operation

P2SEL1 &= ~P2_4; // P2_4 GPIO operation

P2DIR &= ~P2_4; // Direction = input

//P2 Pin 5

P2SEL0 &= ~P2_5; // P2_5 GPIO operation

P2SEL1 &= ~P2_5; // P2_5 GPIO operation

P2DIR &= ~P2_5; // Direction = input

//P2 Pin 6

P2SEL0 &= ~LFXOUT; // LFXOUT Clock operation

P2SEL1 |= LFXOUT; // LFXOUT Clock operation

//P2 Pin 7

P2SEL0 &= ~LFXIN; // LFXIN Clock operation

P2SEL1 |= LFXIN; // LFXIN Clock operation

```
//=====
=====
}
```

9.7.3 Port3.c

```
//=====
==
```

// File Name : port3.c

//

// Description: This file contains the initialization of port 3

//

// Author: William Brazil

// Date: August 2020

// Compiler: Built with IAR Embedded Workbench Version: V7.20.1

```
//=====
==
```

#include "msp430.h"

#include "functions.h"

#include <string.h>

#include "macros.h"

void Init_Port3(void){

P3OUT = LOW; // P3 set Low

P3DIR = OUTPUT; // Set P3 direction to output

//P3 Pin 0

P3SEL0 &= ~TEST_PROBE; //TEST_PROBE GPIO operation

P3SEL1 &= ~TEST_PROBE; //TEST_PROBE GPIO operation

P3DIR &= ~TEST_PROBE; //TEST_PROBE input

//P3 Pin 1

P3SEL0 &= ~CHECK_BAT; //CHECK_BAT GPIO operation

P3SEL1 &= ~CHECK_BAT; //CHECK_BAT GPIO operation

P3DIR &= ~CHECK_BAT; //CHECK_BAT input

//P3 Pin 2

P3SEL0 &= ~OA2N; //OA2N GPIO operation

P3SEL1 &= ~OA2N; //OA2N GPIO operation

P3DIR &= ~OA2N; //OA2N input

//P3 Pin 3

P3SEL0 &= ~OA2P; //OA2P GPIO operation

P3SEL1 &= ~OA2P; //OA2P GPIO operation

P3DIR &= ~OA2P; //OA2P input

//P3 Pin 4

P3SEL0 &= ~SMCLK_OUT; //SMCLK_OUT GPIO operation

P3SEL1 &= ~SMCLK_OUT; //SMCLK_OUT GPIO operation

P3DIR &= ~SMCLK_OUT; //SMCLK_OUT input

//P3 Pin 5

P3SEL0 &= ~DAC_CNTL; //DAC_CNTL GPIO operation

P3SEL1 &= ~DAC_CNTL; //DAC_CNTL GPIO operation

P3DIR &= ~DAC_CNTL; //DAC_CNTL input

//P3 Pin 6

P3SEL0 &= ~IOT_LINK; //IOT_LINK GPIO operation

P3SEL1 &= ~IOT_LINK; //IOT_LINK GPIO operation

P3DIR &= ~IOT_LINK; //IOT_LINK input

//P3 Pin 7

P3SEL0 &= ~P3_7; //P3_7 GPIO operation

P3SEL1 &= ~P3_7; //P3_7 GPIO operation

P3DIR &= ~P3_7; //P3_7 input

}

9.7.4 Port4.c

```
//=====
//
// File Name : port4.c
//
// Description: This file contains the initialization of port 4
//
// Author: William Brazil
// Date: August 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
//
```

```
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"
```

```
void Init_Port4(void){
    P4OUT = LOW;      //P4 set Low
    P4DIR = OUTPUT;   //Set P4 direction to output

    //P4 Pin 0
    P4SEL0 &= ~RESET_LCD; //RESET_LED GP I/O operation
    P4SEL1 &= ~RESET_LCD; //RESET_LED GP I/O operation
    P4OUT &= ~RESET_LCD; //Set RESET_LED on (inital value 0)
    P4DIR |= RESET_LCD;  //RESET_LED direction output

    //P4 Pin 1
    P4SEL0 &= ~SW1; // SW1 set as I/O
```

```
P4SEL1 &= ~SW1; // SW1 set as I/O
P4DIR &= ~SW1; // SW1 Direction = input
P4PUD |= SW1; // Configure pull-up resistor SW1
P4REN |= SW1; // Enable pull-up resistor SW1
P4IES |= SW1; // SW1 Hi/Lo edge interrupt
P4IFG &= ~SW1; // IFG SW1 cleared
P4IE |= SW1; // SW1 interrupt Enabled
```

//P4 Pin 2

```
P4SEL0 |= UCA1RXD; //USCI_A1 UART operation
P4SEL1 &= ~UCA1RXD; //USCI_A1 UART operation
```

//P4 Pin 3

```
P4SEL0 |= UCA1TXD; //USCI_A1 UART operation
P4SEL1 &= ~UCA1TXD; //USCI_A1 UART operation
```

//P4 Pin 4

```
P4SEL0 &= ~UCB1_CS_LCD; //UCB1_CS_LCD GPIO operation
P4SEL1 &= ~UCB1_CS_LCD; //UCB1_CS_LCD GPIO operation
P4OUT |= UCB1_CS_LCD; //Set UCB1_CS_LCD off (initial value 1)
P4DIR |= UCB1_CS_LCD; //UCB1_CS_LCD direction to output.
```

//P4 Pin 5

```
P4SEL0 |= UCB1CLK; //UCB1CLK SPI BUS operation
P4SEL1 &= ~UCB1CLK; //UCB1CLK SPI BUS operation
```

//P4 Pin 6

```
P4SEL0 |= UCB1SIMO; //UCB1SIMO SPI BUS operation
P4SEL1 &= ~UCB1SIMO; //UCB1SIMO SPI BUS operation
```

//P4 Pin 7

```
P4SEL0 |= UCB1SOMI; //UCB1SOMI SPI BUS operation
```

```
P4SEL1 &= ~UCB1SOMI; //UCB1SOMI SPI BUS operation
}
```

9.7.5 Port5.c

```
//=====
==
```

```
// File Name : port5.c
```

```
//
```

```
// Description: This file contains the initialization of port 5
```

```
//
```

```
// Author: William Brazil
```

```
// Date: August 2020
```

```
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
```

```
//=====
==
```

```
#include "msp430.h"
```

```
#include "functions.h"
```

```
#include <string.h>
```

```
#include "macros.h"
```

```
void Init_Port5(void){
```

```
    P5OUT = LOW;      //P5 set low
```

```
    P5DIR = OUTPUT;   //Set P5 direction to output
```

```
    //P5 pin 0
```

```
    P5SEL0 &= ~IOT_RESET; //IOT_RESET GPIO operation
```

```
    P5SEL1 &= ~IOT_RESET; //IOT_RESET GPIO operation
```

```
    P5DIR &= ~IOT_RESET; //IOT_RESET input
```

```
    //P5 pin 1
```

```
    P5SEL0 |= V_BAT;   //ADC input for V_BAT
```

```
    P5SEL1 |= V_BAT;   //ADC input for V_BAT
```

```
//P5 pin 2
P5SEL0 &= ~IOT_PROG_SEL; //IOT_PROG_SEL GPIO operation
P5SEL1 &= ~IOT_PROG_SEL; //IOT_PROG_SEL GPIO operation
P5DIR &= ~IOT_PROG_SEL; //IOT_PROG_SEL input

//P5 pin 3
P5SEL0 &= ~V_3_3; //V_3_3 GPIO operation
P5SEL1 &= ~V_3_3; //V_3_3 GPIO operation
P5DIR &= ~V_3_3; //V_3_3 input

//P5 pin 4
P5SEL0 &= ~IOT_PROG_MODE; //IOT_PROG_MODE GPIO operation
P5SEL1 &= ~IOT_PROG_MODE; //IOT_PROG_MODE GPIO operation
P5DIR &= ~IOT_PROG_MODE; //IOT_PROG_MODE input
}
```

9.7.6 Port6.c

```
//=====
//
// File Name : port6.c
//
// Description: This file contains the initialization of port 6
//
// Author: William Brazil
// Date: August 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
//

#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"
```



```

void Init_Port6(void){
    P6OUT = LOW;      //P6 set low
    P6DIR = INPUT;    //Set P6 direction to output

    //P6 pin 0
    P6SEL0 |= R_FORWARD; //R_FORWARD GPIO operation
    P6SEL1 &= ~R_FORWARD; //R_FORWARD GPIO operation
    P6DIR |= R_FORWARD; //R_FORWARD output

    //P6 pin 1
    P6SEL0 |= ~L_FORWARD; //L_FORWARD GPIO operation
    P6SEL1 &= ~L_FORWARD; //L_FORWARD GPIO operation
    P6DIR |= L_FORWARD; //L_FORWARD output

    //P6 pin 2
    P6SEL0 |= ~R_REVERSE; //R_REVERSE GPIO operation
    P6SEL1 &= ~R_REVERSE; //R_REVERSE GPIO operation
    P6DIR |= R_REVERSE; //R_REVERSE output

    //P6 pin 3
    P6SEL0 |= ~L_REVERSE; //L_REVERSE GPIO operation
    P6SEL1 &= ~L_REVERSE; //L_REVERSE GPIO operation
    P6DIR |= L_REVERSE; //L_REVERSE output

    //P6 pin 4
    P6SEL0 &= ~LCD_BACKLITE; //LCD_BACKLITE GPIO operation
    P6SEL1 &= ~LCD_BACKLITE; //LCD_BACKLITE GPIO operation
    P6OUT = LCD_BACKLITE; //LCD_BACKLITE ON
    P6DIR |= LCD_BACKLITE; //LCD_BACKLITE output

    //P6 pin 5
    P6SEL0 &= ~P6_5;    //P6_5 GPIO operation

```

```
P6SEL1 &= ~P6_5;    //P6_5 GPIO operation
P6DIR &= ~P6_5;     //P6_5 input
```

```
//P6 pin 6
P6SEL0 &= ~GRN_LED; //GRN_LED GPIO operation
P6SEL1 &= ~GRN_LED; //GRN_LED GPIO operation
P6OUT &= ~GRN_LED;  //GRN_LED on
P6DIR |= GRN_LED;   //GRN_LED output
}
```

9.7.7 Ports.c

```
//=====
==
// File Name : ports.c
//
// Description: This file contains the Initialization for all port pins
//
// Author: William Brazil
// Date: August 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
==
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"

void Init_Ports(void){

//=====
==
// Init_Ports
// Purpose: Initialize all Ports
```

```
//=====
==
Init_Port1();
Init_Port2();
Init_Port3();
Init_Port4();
Init_Port5();
Init_Port6();
}
```

9.8. States.c

```
//=====
==
// File Name : states.c
//
// Description: This file contains different states for wheel movement
//
// Author: William Brazil
// Date: September 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
==
#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"

char state;
extern unsigned int cycle_time;
extern unsigned int time_change;
unsigned int right_motor_count;
unsigned int left_motor_count;
unsigned int segment_count;
```

```

unsigned int delay_start;
extern char event;
unsigned int count;
unsigned int left_time;
unsigned int right_time;
unsigned int rev;           //Time spent doing an action
unsigned int repeat;       //number of times the shape will repeat
unsigned int loopcount;    //Each shape has segments (i.e. Figure 8 has two loops, triangle has 3 sides
                           //and 3 corners)

                           //Loopcount is an indicator of how many segements there are in said shape

```

```

//STRAIGHT=====

```

```

void Run_Straight(void){

```

```

    switch(state){

```

```

        case WAIT:

```

```

            wait_case();

```

```

            break;

```

```

        case START:

```

```

            start_case();

```

```

            break;

```

```

        case RUN:    // Run

```

```

            straight_case();

```

```

            break;

```

```

        case END:    // End

```

```

            end_case();

```

```

            break;

```

```

        default: break;

```

```

    }

```

```

}

```

```

//=====

```

```

//CIRCLE=====

```

```

void Run_Circle(void){           //CIRCLE

```

```

switch(state){
  case WAIT:
    wait_case();
    break;
  case START:
    start_case();
    break;
  case RUN:    // Run
    circle_case();
    break;
  case END:    // End
    end_case();
    break;
  default: break;
}
}

//=====

```

```

//FIGURE8=====
void Run_Figure8(void){
  switch(state){
    case WAIT:
      wait_case();
      break;
    case START:
      start_case();
      break;
    case RUN:    // Run
      figure8_case();
      break;
    case END:    // End
      end_case();

```

```

    break;
default: break;
}
}

//=====

//TRIANGLE=====

void Run_Triangle(void){
    switch(state){
        case WAIT:
            wait_case();
            break;
        case START:
            start_case();
            break;
        case RUN:    // Run
            triangle_case();
            break;
        case END:    // End
            end_case();
            break;
        default: break;
    }
}

//=====

void wait_case(void){
    if(time_change){
        time_change = 0;
        if(delay_start++ >= WAITING2START){
            delay_start = 0;
            state = START;
        }
    }
}

```

```

    }
}
loopcount = 0;
}

```

```

void start_case(void){
    cycle_time = 0;
    right_motor_count = 0;
    left_motor_count = 0;
    left_time = (LEFT_COUNT_TIME/7)-1;
    right_time = (RIGHT_COUNT_TIME/7)-1;
    Forward_On();
    segment_count = 0;
    state = RUN;
}

```

```

void straight_case(void){           //STRAIGHT
    if(time_change){
        time_change = 0;
        if(segment_count <= TRAVEL_DISTANCE){
            if(right_motor_count++ >= RIGHT_COUNT_TIME){
                P6OUT &= ~R_FORWARD;
            }
            if(left_motor_count++ >= LEFT_COUNT_TIME){
                P6OUT &= ~L_FORWARD;
            }
            if(cycle_time >= WHEEL_COUNT_TIME){
                cycle_time = 0;
                right_motor_count = 0;
                left_motor_count = 0;
                segment_count++;
                Forward_Move();
            }
        }
    }
}

```

```

    }
}else{
    state = END;
}
}
}

void circle_case(void){           //CIRCLE
    rev = TRAVEL_DISTANCE*63;      // how long I want to be turning for

    right_turn(rev);               //the turn itself

    if(segment_count > rev && repeat > 0){ //repeats the number of times indicated in switches.c
        repeat--;
        state = START;
    }else if(segment_count > rev && repeat == 0){
        state = END;
    }
}

void figure8_case(void){          //FIGURE8

    rev = TRAVEL_DISTANCE*63;     //how long I want to be turning for

    switch(loopcount){ //The actual turns
        case 0:
            left_turn(rev);
            break;
        case 1:
            right_turn(rev);
            break;
        default: break;
    }
}

```



```

}

if(segment_count > rev && loopcount < 1){ //This checks whether its turned both left and right or not
    loopcount++;
    segment_count = 0;
}
if (segment_count > rev && loopcount == 1){
    if(repeat != 0){
        repeat--;
        loopcount = 0;
        state=START;
    }else if(repeat == 0){
        state=END;
    }
}
}
}

```

```

void triangle_case(void){ //TRIANGLE
    rev = TRAVEL_DISTANCE*21; // how long I want to be turning for

    switch(loopcount){ //The actual turns
        case 0:
            straight(rev);
            break;
        case 1:
            left_turn(rev);
            break;
        case 2:
            straight(rev);
            break;
        case 3:

```

```

    left_turn(rev);
    break;
case 4:
    straight(rev);
    break;
case 5:
    left_turn(rev);
    break;
default: break;
}          //the turn itself

if(segment_count > rev && loopcount < 5){ //This checks whether its turned both left and right or not
    loopcount++;
    segment_count = 0;
}
if (segment_count > rev && loopcount == 5){
    if(repeat != 0){
        repeat--;
        loopcount = 0;
        state=START;
    }else if(repeat == 0){
        state=END;
    }
}
}

void end_case(void){
    Forward_Off();
    state = WAIT;
    event = NONE;
}

```

```

void left_turn(unsigned int turntime){
    if(time_change){
        time_change = 0;
        if(segment_count <= turntime){
            if(right_motor_count++ >= RIGHT_COUNT_TIME){
                P6OUT &= ~R_FORWARD;
            }
            if(left_motor_count++ >= left_time){
                P6OUT &= ~L_FORWARD;
            }
            if(cycle_time >= WHEEL_COUNT_TIME){
                cycle_time = 0;
                right_motor_count = 0;
                left_motor_count = 0;
                segment_count++;
                Forward_Move();
            }
        }
    }
}

```

```

void right_turn(unsigned int turntime){
    if(time_change){
        time_change = 0;
        if(segment_count <= turntime+4) {
            if(right_motor_count++ >= right_time){
                P6OUT &= ~R_FORWARD;
            }
            if(left_motor_count++ >= LEFT_COUNT_TIME){
                P6OUT &= ~L_FORWARD;
            }
            if(cycle_time >= WHEEL_COUNT_TIME){

```

```

    cycle_time = 0;
    right_motor_count = 0;
    left_motor_count = 0;
    segment_count++;
    Forward_Move();
}
}
}
rev = rev+4;
}

void straight(unsigned int turntime){
    if(time_change){
        time_change = 0;
        if(segment_count <= turntime/6){
            if(right_motor_count++ >= RIGHT_COUNT_TIME){
                P6OUT &= ~R_FORWARD;
            }
            if(left_motor_count++ >= LEFT_COUNT_TIME){
                P6OUT &= ~L_FORWARD;
            }
            if(cycle_time >= WHEEL_COUNT_TIME){
                cycle_time = 0;
                right_motor_count = 0;
                left_motor_count = 0;
                segment_count++;
                Forward_Move();
            }
        }
    }
    rev = rev/7;
}

```

9.9. Switches.c

```
//=====
//
// File Name : switches.c
//
// Description: This file contains different switches process for SW1 and SW2
//
// Author: William Brazil
// Date: September 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
//
```

```
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"
```

```
extern char display_line[DISPLAY_ROWS][DISPLAY_COLUMNS];
extern volatile unsigned char display_changed;
extern volatile unsigned char display_changed;
extern char start_moving;
extern unsigned int moving;
extern char event;
extern unsigned int repeat;
extern volatile unsigned int SW1_PRESSED;
extern volatile unsigned int SW2_PRESSED;
extern volatile unsigned int SW1_DEBOUNCED;
extern volatile unsigned int SW2_DEBOUNCED;
extern volatile unsigned int SW1_DEBOUNCE_COUNT;
```

```

extern volatile unsigned int SW2_DEBOUNCE_COUNT;
int IR_State = ON;
extern int FoundLine;
extern char switch_states;
extern char port_command[16];
extern char start_command[11];

```

```

void Switches_Process(void){

```

```

//=====
=====

```

```

// Function Name: Swtiches_Process

```

```

//

```

```

// Description: Handles the SW1 debounce counter after the debounce delay is
// over it reenables the respective SW and re-enables the blinking LCD

```

```

//

```

```

// Local Variables: None

```

```

// Global Variables: SW1_DEBOUNCE_COUNT, SW1_DEBOUNCED, SW1_PRESSED,
//          display_line[DISPLAY_ROWS][DISPLAY_COLUMNS], display_changed

```

```

//

```

```

// William Brazil

```

```

// Sept 2020

```

```

// Built with IAR Embedded Workbench Version: V(7.20.1)

```

```

//=====
=====

```

```

if(switch_states & SW1){
    switch_states &= ~SW1;    //clear flag bit

```

```

//FUNCTION=====

```

```

for(int z = INIT; z<sizeof(port_command);z++)
    out_character(port_command[z]);

```

```

// for(int i=INIT;i<MAX_COUNT;i++)      //small delay
//   msec_sleep(FOUR*ONETHOUSAND);
//
//   FoundLine = FALSE;
//
//   //search for blkline
//   Search();
//   //stop
//   all_wheels_off();
//   //display
//   strcpy(display_line[LINE1], "Searching.");
//   update_string(display_line[LINE1], LINE1);
//   display_changed = TRUE;
//   //movement
//   forward_slow();
//
//   display_changed = TRUE;
}

if(switch_states & SW2){
    switch_states &= ~SW2;

//FUNCITON=====

switch(IR_State){
case OFF:
    P2OUT &= ~IR_LED;
    emitter_state(OFF);
    IR_State = ON;
    break;
case ON:

```

```
P2OUT |= IR_LED;
emitter_state(ON);
IR_State = OFF;
break;
default:
break;
}
}
}

void enable_switch_SW1(void){

}

void enable_switch_SW2(void){

}

void Init_Switches(void){

}
```

9.10. System.c

```
//=====
=====
//
// Description: This file contains the System Configurations
//
// Jim Carlson
// Jan 2016
// Built with IAR Embedded Workbench Version: V7.3.1.3987 (6.40.1)
//=====
=====
```



```
//=====
=====
```

```
#include "functions.h"
```

```
#include "msp430.h"
```

```
#include "macros.h"
```

```
void enable_interrupts(void);
```

```
//=====
=====
```

```
// System Configurations
```

```
// Tells the compiler to provide the value in reg as an input to an
// inline assembly block. Even though the block contains no instructions,
// the compiler may not optimize it away, and is told that the value
// may change and should not be relied upon.
```

```
//inline void READ_AND_DISCARD(unsigned int reg) __attribute__((always_inline));
```

```
//void READ_AND_DISCARD(unsigned int reg){
```

```
// asm volatile ("" : "=m" (reg) : "r" (reg));
```

```
//}
```

```
//inline void enable_interrupts(void) __attribute__((always_inline));
```

```
void enable_interrupts(void){
```

```
    __bis_SR_register(GIE);    // enable interrupts
```

```
// asm volatile ("eint \n");
```

```
}
```

```
//inline void disable_interrupts(void) __attribute__((always_inline));
```

```
//void disable_interrupts(void){
```

```
// asm volatile ("dint \n");
```

```
//}
```

9.11. Timers.c

```
//=====
//
// File Name : timers.c
//
// Description: This file contains the Initialization of TimerB0 and
// a 5ms delay function
//
// Author: William Brazil
// Date: September 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
//
```

```
#include "functions.h"
#include "msp430.h"
#include <string.h>
#include "macros.h"
```

```
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
volatile unsigned int FIFTY_MS_INTERVALS = INIT;
volatile unsigned int ONE_S_INTERVALS = INIT;
volatile unsigned int FIVE_MS_INTERVALS = INIT;
int DAC_data;
```

```
void Init_Timer_B0(void) {
```

```
//=====
=====
```

```
// Function Name: Init_Timer_B0
//
// Description: Initializes TimerB0 and sets up both B0_0, B0_1-B0_2
// and overflow
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Sept 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====

TB0CTL = TBSSEL__SMCLK;    // SMCLK source
TB0CTL |= TBCLR;          // Resets TB0R, clock divider, count direction
TB0CTL |= MC__2;          // Continuous up
TB0CTL |= ID__8;          // Divide clock by 8

TB0EX0 = TBIDEX__8;        // Divide clock by an additional 8

TB0CCR0 = TB0CCR0_INTERVAL; // CCR0
TB0CCR0 &= ~CCIFG;
TB0CCTL0 |= CCIE;          // CCR0 enable interrupt

TB0CCR2 &= ~CCIFG;
TB0CCTL2 |= CCIE;          //CCR2 disable interrupt

TB0CTL &= ~TBIE;          // Disable Overflow Interrupt
TB0CTL &= ~TBIFG;          // Clear Overflow Interrupt flag
//TB0CCR1 = TB0CCR1_INTERVAL; // CCR1
```

```

}

void Init_Timer_B1(void){

//=====
=====
// Function Name: Init_Timer_B1
//
// Description: Initializes TimerB1 and sets up both B1_0, B1_1-B1_2
// and overflow
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Oct 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====
TB1CTL = TBSEL__SMCLK;    // SMCLK source
TB1CTL |= TBCLR;         // Resets TB0R, clock divider, count direction
TB1CTL |= MC_2;          // Continuous up
TB1CTL |= ID__8;         // Divide clock by 8

TB1EX0 = TBIDEX__8;      // Divide clock by an additional 8

TB1CCR0 = TB1CCR0_INTERVAL; // CCR0
TB1CCR0 &= ~CCIFG;
TB1CCTL0 &= ~CCIE;      // CCR0 disables interrupt

TB1CCR1 = TB1CCR1_INTERVAL;
TB1CCR1 &= ~CCIFG;
TB1CCTL1 &= ~CCIE;      //CCR1 disables interrupt

```

```

TB1CCR2 = TB1CCR2_INTERVAL;
TB1CCR2 &= ~CCIFG;
TB1CCTL2 &= ~CCIE;

TB1CTL &= ~TBIE;          // Disable Overflow Interrupt
TB1CTL &= ~TBIFG;         // Clear Overflow Interrupt flag
}

void Init_Timer_B3(void){

//=====
=====
// Function Name: Init_Timer_B3
//
// Description: Initialized TimerB3 and CCR0, CCR1, CCR2, CCR3, and CCR4.
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Sept 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====

TB3CTL = TBSSEL__SMCLK; // SMCLK
TB3CTL |= MC__UP; // Up Mode
TB3CTL |= TBCLR; // Clear TAR

TB3CCR0 = WHEEL_PERIOD; // PWM Period

TB3CCTL1 = OUTMOD_7; // CCR1 reset/set

```

RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM duty cycle

TB3CCTL2 = OUTMOD_7; // CCR2 reset/set

LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM duty cycle

TB3CCTL3 = OUTMOD_7; // CCR3 reset/set

RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM duty cycle

TB3CCTL4 = OUTMOD_7; // CCR4 reset/set

LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM duty cycle

}

void msec_sleep(unsigned int msec){

//=====

// Function Name: five_msec_sleep

//

// Description: Pauses the clock by doing a counter for x ms

//

// Local Variables: msec

// Global Variables: FIVE_MS_INTERVALS

//

// William Brazil

// Sept 2020

// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====

FIVE_MS_INTERVALS = INIT;

while((FIVE_MS_INTERVALS++) < msec);

```

}

void Init_ADC(void){

//=====
=====
// Function Name: Init_ADC
//
// Description: Initializes ADC conversion and Registers 0, 1, and 2.
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Oct 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====

//-----
// V_DETECT_L (0x04) // P1 Pin 2 A2
// V_DETECT_R (0x08) // P1 Pin 3 A3
// V_THUMB (0x02) // P5 Pin 1 A9
//-----

// ADCCTL0 Register
ADCCTL0 = RESET;           // Reset
ADCCTL0 |= ADCSHT_0;       // 4 ADC clocks
ADCCTL0 |= ADCMSC;         // MSC
ADCCTL0 |= ADCON;          // ADC ON

```

```
// ADCCTL1 Register
ADCCTL1 = RESET;           // Reset
ADCCTL1 |= ADCSHS_0;       // 00b = ADCSC bit
ADCCTL1 |= ADCSHP;         // ADC sample-and-hold SAMPCON signal from sampling timer.
ADCCTL1 &= ~ADCISSH;       // ADC invert signal sample-and-hold.
ADCCTL1 |= ADCDIV_0;       // ADC clock divider - 000b = Divide by 1
ADCCTL1 |= ADCSSEL_0;      // ADC clock SMCLK
ADCCTL1 |= ADCCONSEQ_0;    // ADC conversion sequence 00b = Single-channel single-
conversion
// ADCCTL1 & ADCBUSY identifies a conversion is in process
```

```
// ADCCTL2 Register
ADCCTL2 = RESET;           // Reset
ADCCTL2 |= ADCPDIV0;       // ADC pre-divider 00b = Pre-divide by 1
ADCCTL2 |= ADCRES_1;       // ADC resolution 10b = 10 bit (12 clock cycle conversion time)
ADCCTL2 &= ~ADCDF;         // ADC data read-back format 0b = Binary unsigned.
ADCCTL2 &= ~ADCSR;         // ADC sampling rate 0b = ADC buffer supports up to 200 ksps
```

```
// ADCMCTL0 Register
ADCMCTL0 = RESET;
ADCMCTL0 |= ADCSREF_0;     // VREF - 000b = {VR+ = AVCC and VR- = AVSS }
ADCMCTL0 |= ADCINCH_2;     // V_DETECT_L A2
ADCIE |= ADCIE0;
ADCCTL0 |= ADCENC;         // ADC enable conversion.
ADCCTL0 |= ADCSC;          // ADC start conversion.
}
```

```
void Init_DAC(void){
```

```
//=====
=====
// Function Name: Init_DAC
//
```



```
// Description: Initializes DAC conversion
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Oct 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====

//using 1000 sets it to 6.1V
DAC_data = 1000; // Value between 0x000 and 0x0FFF
SAC3DAT = DAC_data; // Initial DAC data
SAC3DAC = DACSREF_1; // Select int Vref as DAC reference
SAC3DAC |= DACLSEL_0; // DAC latch loads when DACDAT written
// SAC3DAC |= DACIE; // generate an interrupt
SAC3DAC |= DACEN; // Enable DAC
SAC3OA = NMUXEN; // SAC Negative input MUX control
SAC3OA |= PMUXEN; // SAC Positive input MUX control
SAC3OA |= PSEL_1; // 12-bit reference DAC source selected
SAC3OA |= NSEL_1; // Select negative pin input
SAC3OA |= OAPM; // Select low speed and low power mode
SAC3PGA = MSEL_1; // Set OA as buffer mode
SAC3OA |= SACEN; // Enable SAC
SAC3OA |= OAEN; // Enable OA
}

void Init_REF(void){

//=====
=====

// Function Name: Init_REF
```

```
//
// Description: Initializes Refernce Module
//
// Local Variables: None
// Global Variables: None
//
// William Brazil
// Oct 2020
// Built with IAR Embedded Workbench Version: V(7.20.1)

//=====
=====

// Configure reference module
PMMCTL0_H = PMMPW_H; // Unlock the PMM registers
PMMCTL2 = INTREFEN; // Enable internal reference
PMMCTL2 |= REFVSEL_2; // Select 2.5V reference
while(!(PMMCTL2 & REFGENRDY)); // Poll till internal reference settles
// Using a while statement is not usually recommended without an exit strategy.
// This while statement is the suggested operation to allow the reference to settle.
}
```

9.12. Wheels.c

```
//=====
==

// File Name : wheels.c
//
// Description: This file contains all wheel controls.
//
// Author: William Brazil
// Date: September 2020
// Compiler: Built with IAR Embedded Workbench Version: V7.20.1
//=====
==
```

```
#include "msp430.h"
#include "functions.h"
#include <string.h>
#include "macros.h"
```

```
extern int spin;
extern int Interval_Counter;
```

```
void all_wheels_off(void){
    //Set PWM for all wheels off.
    RIGHT_FORWARD_SPEED = WHEEL_OFF;
    LEFT_FORWARD_SPEED = WHEEL_OFF;
    RIGHT_REVERSE_SPEED = WHEEL_OFF;
    LEFT_REVERSE_SPEED = WHEEL_OFF;
}
```

```
void forward_slow(void){
    RIGHT_FORWARD_SPEED = WHEEL_ON_50;
    LEFT_FORWARD_SPEED = WHEEL_ON_50 + LEFT_DIFFERENTIAL;
}
```

```
void reverse(void){
    RIGHT_REVERSE_SPEED = WHEEL_ON_50;
    LEFT_REVERSE_SPEED = WHEEL_ON_50;
}
```

```
void reverse_slow(void){
    RIGHT_REVERSE_SPEED = WHEEL_ON_20;
    LEFT_REVERSE_SPEED = WHEEL_ON_20 + LEFT_DIFFERENTIAL;
}
```

```
void stop(void){
    RIGHT_FORWARD_SPEED = WHEEL_OFF;
    LEFT_FORWARD_SPEED = WHEEL_OFF;
    RIGHT_REVERSE_SPEED = WHEEL_OFF;
    LEFT_REVERSE_SPEED = WHEEL_OFF;
}
```

```
void left_wheel_forward_on(void){
    LEFT_FORWARD_SPEED = WHEEL_ON_20;
}
```

```
void left_wheel_reverse_on(void){
    LEFT_REVERSE_SPEED = WHEEL_ON_20;
}
```

```
void right_wheel_forward_on(void){
    RIGHT_FORWARD_SPEED = WHEEL_ON_20;
}
```

```
void right_wheel_reverse_on(void){
    RIGHT_REVERSE_SPEED = WHEEL_ON_20;
}
```

```
void NINETY_RIGHT(int delay){
    switch(delay){
    case FALSE:
        //enable interrupt to turn 90deg
        spin = TRUE;
        Interval_Counter = INIT;
        TB1CCTL2 |= CCIE;
        break;
    case TRUE:
```

```

//delay itself
for(int j=INIT;j<MAX_COUNT;j++)
    msec_sleep(TEN*ONETHOUSAND);
//make sure wheels are off
all_wheels_off();

//enable interrupt to turn 90deg
spin = TRUE;
Interval_Counter = INIT;
TB1CCTL2 |= CCIE;
}
}

void Forward_On(void){
    left_wheel_forward_on();
    right_wheel_forward_on();
}

void Forward_Move(void){
    forward();
}

void Forward_Off(void){
    RIGHT_FORWARD_SPEED = WHEEL_OFF;
    LEFT_FORWARD_SPEED = WHEEL_OFF;
}

```

10. Conclusion

The Electric Rover seems quite simple from the outside. However, when broken down into small pieces and looking at the small details the complexity of the device can be seen clearly. This document shows explains every component of the car from motor movements to ADC and DAC conversion. Over the course of development, the Electric Rover has truly proven to be a challenge and a very new and eye-opening experience. It showed what it truly means to create an embedded system and teaches us about

not just the importance of each component, but the importance of being able to bring all of them together to create one cohesive device.