Lightning McGreen

Faster than Lightning

Contributors – Team 1: Nathan Carels, Drew Clark, Keith Dunn, Lori Glenn, Sydney Gyurek, and William Moore

Revision	Description
1	General formatting and setup, scope, abbreviations, overview - LG
2	Added scope - SG
3	Added Section 3 overview functionality description and block diagram - NC
4	Added Micro-controller and Power systems - LG
5	Added section about IR Sensor & Motor Controller - SG
6	Added section 3 simple block descriptions - NC
7	Updated abbreviations and table of contents/fixed grammatical and spelling errors/Added Software - KD
8	Added Section 6 regarding Testing Processes - DC
9	Added Section 4.6 for the User Interface and proofread and worked on formatting the document - WM
10	Made some corrections after return of first document - SG
11	General edits after feedback from first document -LG
12	Edits to sections 3 as guided by feedback and added subsystem diagrams - NC
13	Added Figures/Flow chart, added a bit to the scope and conclusion -SG
14	Added Section 6.2 Power Switch Test and Section 6.3 Analog Discovery Test -DC

Team Members

Nathan Carels
Drew Clark
Keith Dunn
Lori Glenn
Sydney Gyurek
William Moore

Originator: Lori Glenn

Checked: 04/30/2021 | Released: 04/30/2021

Filename: ece_306_report_final.docx

Title: **Lightning McGreen Faster than Lightning**

Date:	Document Number:	Rev:	Sheet:
04/30/2021	**5**	**39**	**1 of 71**

Revision	Description
15	Added Figures, updated table of contents & section headers, editing & formatting - LG
16	General Updates, updated Section 4.3 and 4.4 - WM
17	Minor spelling and grammatical corrections - KD
18	Added sections 6.4 FET Board Power Test and 6.5 Emitter and Detector Test - DC
19	Made revisions based on Grading - WM
20	Added back template for sections 7,8,9 and added ports section - SG
21	Added ADC initialization and ISR code, descriptions, and flowcharts - LG
22	Added Main flowchart, code, and description - NC
23	Updated section 3 to reflect the IR Dectection system that is now in use -NC
24	Fixed miscellaneous errors - DC
25	Added Interrupts flowchart, code, and description. Updated Table of Contents and Figures. - WM
26	Updated Abbreviations, corrected spelling and grammatical errors, added clarifications. -KD
27	Edits based on feeback, serial initialization/ISR code & description & flowcharts - LG
28	Addition of serial communications testing protocol 6.6– NC
29	Addition of images to testing process sections 6.1-6.6 - DC
30	Addition of Section 10 Conclusion - WM
31	Measured and added section 5 wit,h power analysis - NC
32	Added conclusion to TOC, updated ports for project 8, fixed grammatical/spelling errors - SG
33	Made necessary changes to section 3 and added the communications system and susbsystem diagram – NC
34	Fixed magic numbers from Project 8 corrections, added sections 7.6, 8.8, and 9.8 for switches - WM
35	Added commands sections 7.7, 8.9, and 9.9 - DC
36	Added black line following flowchart, description, and code; updated table of contents - LG
37	Wrote conclusion and updated scope - SG
38	Added LCD Display functions description, flowchart and code - NC
39	Fixed revision number, spelling and grammatical errors, made additions to abbreviations - KD

Table of Contents

Figures

1. Scope

As the world enters a new era of technology, there are endless possibilities. Not long ago, the first vehicle was built from society's need for faster and cheaper transportation. Overtime, the original design evolved due to improving technology and expanding needs to produce the plethora of vehicles on the market today. One of the newest shifts in this expansion is toward autonomously run vehicles that perform programmed functions without the need of a human driver. After sending commands through a IOT module, Lightening McGreen can travel through an eight station obstacle course and intercept, travel, and follow a black line around a circle twice.

2. Abbreviations

Abbreviation	Definition
AD2	Analog Discovery 2
ADC	Analog to Digital Converter
API	Application Programming Interface
DAC	Digital to Analog Converter
DMA	Direct Memory Access
FRAM	Ferroelectric Random-Access Memory
IOT	Internet of Things
IR	Infrared
ISR	Interrupt Service Routine
LED	Light-Emitting Diode
LCD	Liquid Crystal Display
MSP430	Microcontroller Board from TI
PCB	Printed Circuit Board
RAM	Random-Access Memory
RTC	Real Time Clock
PWM	Pulse-Width Modulation
UI	User Interface

3. Overview

Lightning McGreen is constructed of interconnected blocks that each perform a specific function that together create a working vehicle. The major blocks are the Power System, Motor Controllers, User Interface, and the MSP403FR2355 Microcontroller.

The system is a remote controllable car with variable speed control. The system relies upon the MSP403FR2355 Microcontroller to communicate with the Motor Controller to drive the wheels in accordance with the instructions received. Additionally, it relies upon the microcontroller to provide sensor analysis and steer the car independent of outside instruction when in autonomous mode. The car

communicates using 2 serial connections and an IOT module capable of receiving TCP commands from any TCP client when using the correct command structure. Finally, the Microcontroller controls the UI handling all button or thumbwheel inputs and delivering the proper outputs to the LCD screen which allows them to be seen by the end user. The Power Board delivers the power to the microcontroller and the wheels via a buck boost converter to maintain two voltage lines of 3.3V and 5V to power the motors and the MSP430 from 4 AA batteries. The power board also provides voltages to be read by the microcontroller. The motor controller provides the voltage needed to drive the wheels in a specific direction according to the microcontroller.

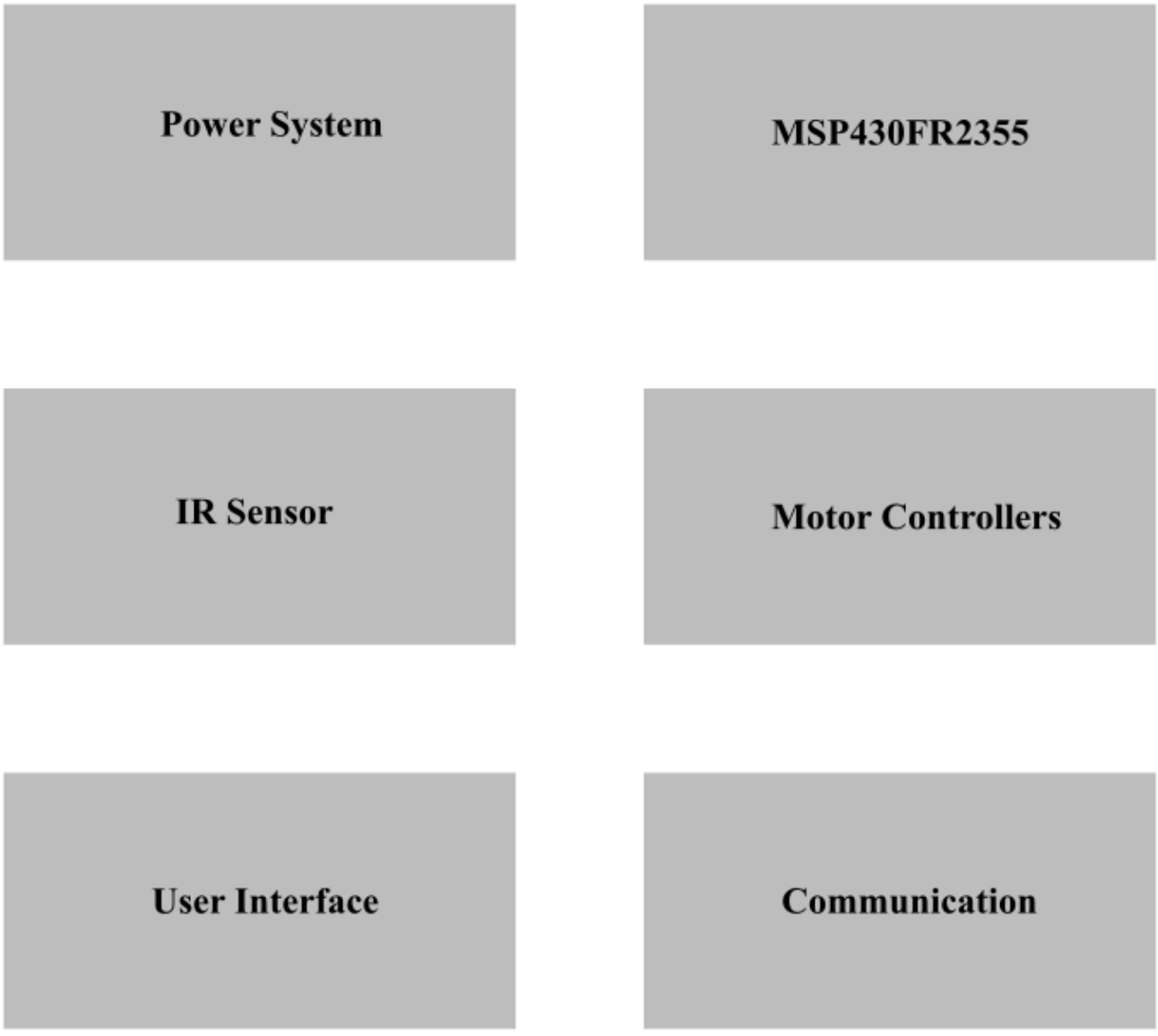


Figure 3 Block Diagram

3.1 Power System

Figure 3.1 Power System Block Diagram

The power system provides power to all the other components at a regulated voltage using a buck boost converter to maintain power stability. The LCD screen is mounted on this module and it connects directly to the microcontroller's board and provides some pathways for the microcontroller to be able to read the voltages supplied.

3.1.1 Batteries

The power system draws its power from 4 AA batteries providing 1.5 volts each when fully charged. In series they provide 6 volts that is then converted into 5 volts and 3.3 volts for the FRAM as well as used to power the motors directly.

3.1.2 Boost converter

The boost converter is the heart of the power system. It takes the unregulated battery voltage and turns it into the 3.3v and 5v regulated power lines.

3.1.3 Power rails

The power rails are what deliver the power to the different components. They come from the boost converter.

3.2 Microcontroller

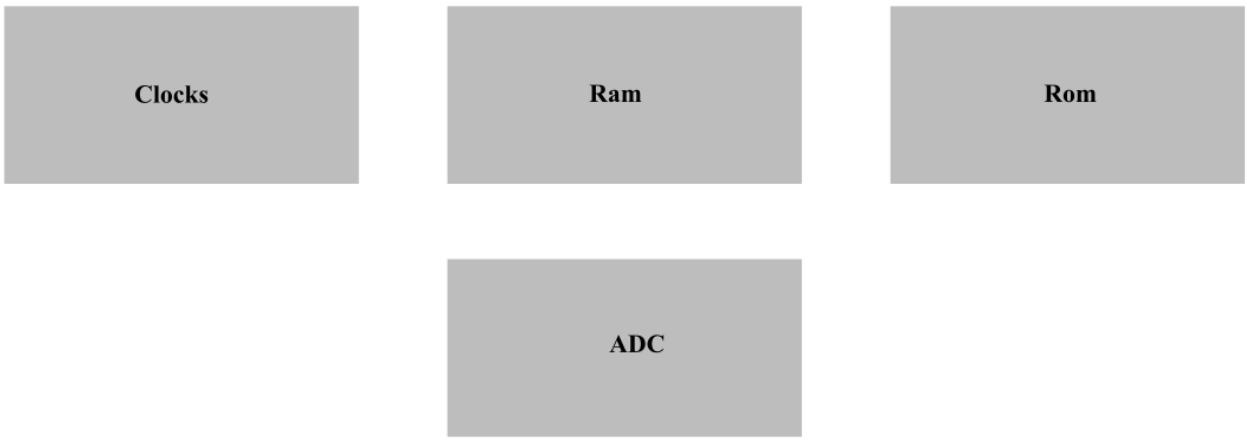


Figure 3.2 Microcontroller Block Diagram

The microcontroller controls all the data analysis and handles all inputs and interrupts that the car needs. It is in control of the motors, LCD, and reads in all the switches and controls the car.

3.2.1 Clocks

The clocks of the microcontroller control all peripheral and system clocks that determine the throughput of the microcontroller and are what the timers are based on.

3.2.2 RAM

The RAM of the microcontroller contains all the working memory for variables to be stored in during runtime.

3.2.3 ADC

The analog to digital converter allows analog voltage signals to be converted into digital signal which can be read and processed by the microcontroller.

3.3 Motor Control

Figure 3.3 Motor Control Block Diagram

The motor controller controls the voltage supplies to the motors to control their speed. This is done with DAC and PWM to allow a consistent drive to be achieved at a wide range of speeds and turning angles. This is controlled by the microcontroller.

3.3.1 DAC

The DAC board will supply a constant specified voltage to the drive train regardless of the battery's voltage (will not function if the batteries can no longer supply power).

3.3.2 Timers

The timers are a part of the microcontroller but provide PWM to control the speed at which the motors drive.

3.3.3 Motors

The 2 electric motors drive the car and are its sole form of propulsion. They do not operate uniformly, and this is compensated for in software.

3.4 User Interface

The user interface on the car itself consists of the LCD screen, 2 switches, and on/off switch and a thumb wheel. These will be used to select and control various car functions and internal values that the operator might want to adjust. The switches are located on the microcontroller board and the thumb wheel is set beneath the LCD screen.

Figure 3.4 User Interface

3.4.1 LCD Screen

The LCD screen is the only optical output device and is controlled by the microcontroller. It has 4 lines on which it can display 10 characters of text each. It also has a 12V converter inside which is routed to an external testing pin.

3.4.2 Switches

The switches are buttons on either side of the MSP403FR2355 board and presses are handled using negative edge triggering interrupts that can be changed to perform a desired function, like starting a drive cycle or toggling the LCD's backlight.

3.4.3 Thumb Wheel

The thumb wheel is a source of analog input to the car. It must be read through the ADC of the microcontroller for its input to be in a usable state.

3.5 IR Sensor

Figure 3.5 IR Sensor System

The IR sensor is a simple combination of an IR emitter and two IR detectors that give the car the ability to detect which side of the car is over a black line on a white background. This system requires ADC functionality in order to read in the analog values that the detectors provide.

3.5.1 IR Emitter

The IR Emitter is controlled by the MSP430 via a FET and is responsible for providing the IR light source that the detectors require to function.

3.5.2 IR Detector

The IR Detectors are placed close to each other in line with the drive axle of the car with one slightly on either side. The voltage across them can be measured by the ADC and used to determine the location of the car relative to a black line.

3.6 Communication System

Figure 3.6 Communication System

The IOT Wi-Fi module allows the operator to control the vehicle wirelessly. It handles the wireless communication to the external computer as well as communicates with the microcontroller. It communicates with the FRAM via UCA 0 serial port. Additionally, UCA 1 is a serial port that can communicate the PC. UCA 1 is specifically used as a back door to communicate with the IOT module and to debug IOT communications during development.

3.6.1 UCA 0

UCA 0 is a serial port that communicates with the IOT module from the FRAM. This connection is used to transmit startup and configuration commands that the IOT modules needs to connect and receive signals. It also passes commands received from the IOT that the FRAM processed.

3.6.2 UCA 1

UCA 1 is a serial port that communicates with a serial terminal on a PC. This is used for sending some commands such as changing UCA 0 baud, debugging the IOT and initial configuration of the IOT via a backdoor pass-through.

3.6.3 IOT Module

The IOT module is how the car communicates with the PC wirelessly. This receives commands that are passed onto the FRAM that allow the car to navigate a course with user interaction via a TCP terminal or client. The module is configured using the UCA 1 USB serial port.

4 Hardware

4.1 Power System

The power system is driven by 4 AA batteries, each supplying about 1.5V. This supplied voltage is run though a buck boost converter to regulate the supplied voltage to a near constant DC value within the ideal operating conditions of the FRAM with maximum voltage of 3.6 V and minimum voltage of 1.8 V. This DC voltage is then used to power the other blocks including the Microcontroller, motors, and user interface.

Figure 4.1: Power System 4 AA batteries each supplying 1.5V

4.2 Microcontroller

The Microcontroller used in this product is the MSP430FR2355, or MSP430. Within the MSP430, there are various subsystems that work to power the clocks, timers, ports, and data flow throughout the rest of the systems in the product. The major subsystems within the microcontroller include memory, communication system, and timing system.

Within the memory subsystem, there are general purpose registers, RAM, FRAM, and a DMA controller. The user's code is imported to the microcontroller and is stored within the registers, RAM, and FRAM. All input and output values are stored within this memory subsystem. The DMA controller allows the microcontroller to store and access data in memory efficiently.

The communication subsystem contains digital ports and analog ports. These ports are used to input and export digital and analog data throughout the MSP430 as well as to the other blocks within the product. Data is read into the microcontroller as analog input voltages and are then processed and converted into digital values that are used in the other systems such as the LCD display.

The timing subsystem includes timers and clocks. The timers and clocks set and maintained within the microcontroller are used to set the pace at which the whole device operates.

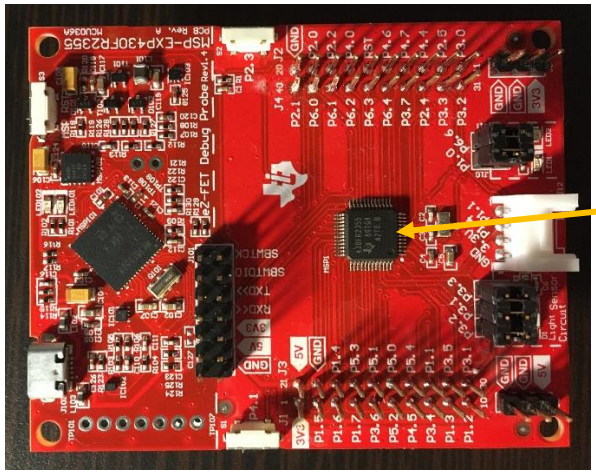


Figure 4.2 FRAM Board

4.3 Powertrain

The powertrain is what makes the car move in the direction, speed, and duration specified by the code in our microcontroller. The main component of the powertrain is the two motors, one for each side. They are connected by wire to our FET board. They are manufactured by LEGO and there is a LEGO wheel and tire on each one. The port pins on our FRAM board that correspond to the powertrain control are 6.0 through 6.3 which control Right Forward, Left Forward, Right Reverse, and Left Reverse, respectively.

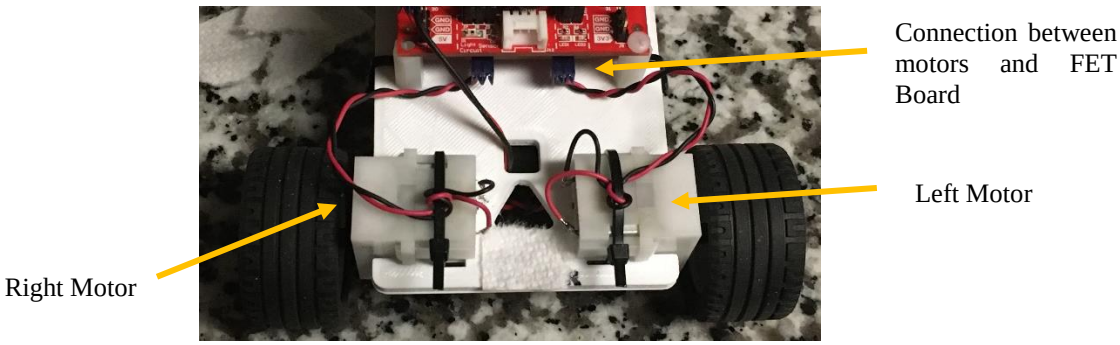


Figure 4.3 Motors

4.4 User Interface

There are five methods of user interface. These include the LCD Display, Thumbwheel, On/Off Switch, and 2 Buttons labeled SW1 and SW2. The On/Off Switch is the method of control to turn off the battery power supply to the circuit boards. Once the Switch is in the On position, the LCD screen lights up and displays “NCSU Wolfpack ECE306”. The buttons control what shape is to be executed by the car. By pressing SW2, the display transitions from “NCSU Wolfpack ECE306” to “CIRCLE” and the car executes two consecutive circles. Then by pressing SW2 again, the display changes to “FIGURE 8” and executes two consecutive figure eights. Another press of SW2 and the display will change to “TRIANGLE” and execute two consecutive equilateral triangles. This process can be done in reverse by pressing SW1 and transitioning to TRIANGLE, FIGURE 8, and CIRCLE in that order. The thumbwheel is installed on the board and is used as a potentiometer.

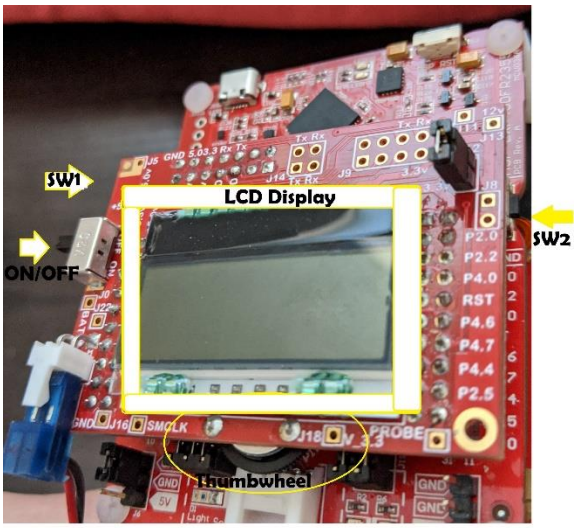


Figure 4.4: User Interface (Thumbwheel, LCD Display, SW1, SW2, and On/Off switch)

5 Power Analysis

The power analysis was done by measuring the current flowing between power board 3.3v and 3.3v_PB knowing that the system voltage is maintained at 3.3V. The following table shows the power for each component. The system was running at 8Mhz and was not utilizing any power saving functionality during the test.

Component	Current used by component in mA (at 3.3V)	Power Consumed (in mW)
IOT, FRAM, LCD Backlight, DAC, ADC, IR LED (full system)	210.5	694.65
FRAM	7.57	24.981
LCD Backlight	59.77	197.24
IR LED	24.10	79.53
ADC	0.20	.66
DAC	0.30	.99
IOT	118.70	391.71

The total system power, with every functionality fully active, is 694.65 mW which will allow the car to last an estimated 24.7 hrs off a fresh set of batteries excluding drive time which was excluded due to the high variability of power consumption but will result in degradation of use time.

6 Test Process

In order to ensure proper functionality of the vehicle during the creation process, multiple methods of testing the hardware and software were utilized.

6.1 Power Board Test

The testing process for powering the Power Board includes applying proper grounds and sending power to the Power Board. An AD2 was used to set the input voltage to 4.5 volts and measure the output voltage which was expected and confirmed to be 3.3 volts. Note that alternatively, a power supply unit can apply power and an oscilloscope or multimeter can measure the voltage.

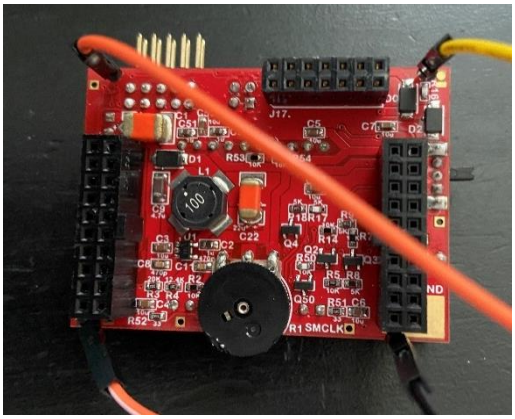


Figure 6.1a Power Board Layout

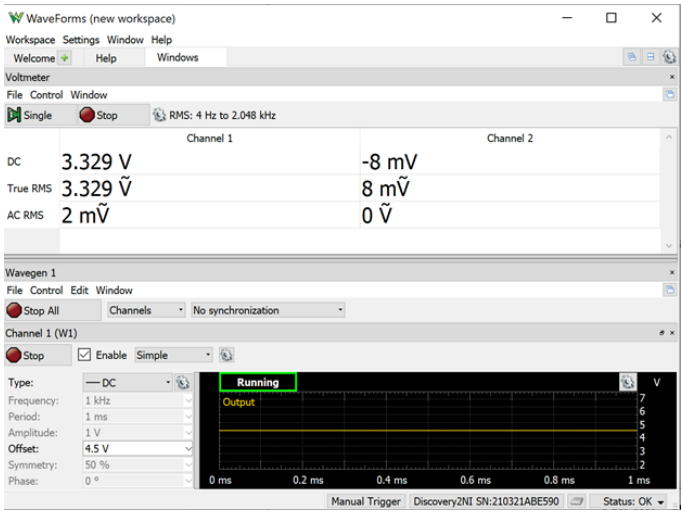


Figure 6.1b Waveforms Setup

6.2 Power Switch Test

After soldering the power switch to the DAC board, a resistance test was needed to ensure the switch works correctly and which way is on or off. This was carried out by using a multimeter to measure the resistance through the switch when the switch is open and closed. The expected outcome was that the closed switch state would be low in resistance and the open switch would have infinite resistance, an open circuit. This proved to be the case and our power switch works correctly.

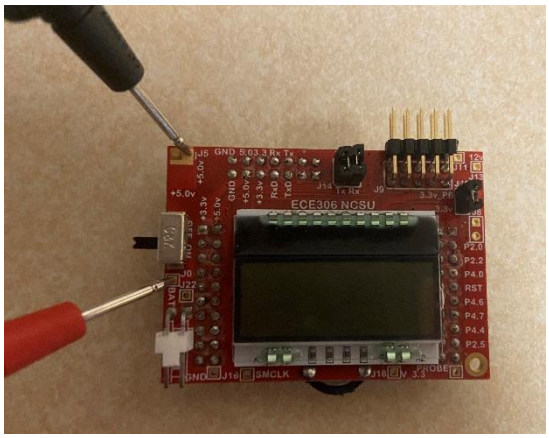


Figure 6.2 Power Switch Layout

6.3 Analog Discovery Test

When implementing code to control the forward and reverse port pins, careful testing must be done to avoid hardware damage. If the port pins for forward and reverse are both set high for a single wheel, the FETs will be damaged by overheating. In fact, if there is too little delay between a motor being set to forward and reverse, the FETs will heat up and may cause irreversible damage over time. To combat this, we used the Analog Discovery to read the port pins 6.0, 6.1, 6.2, and 6.3 digitally. These port pins are used for right forward, left forward, right reverse, and left reverse, respectively. We configured WaveForms, the AD2 software, to give us an error if the port pins for forward and reverse are both set

high for a single wheel. After running the Analog Discovery Test, we can be confident that we are not going to blow our FETs when we try to drive our car.

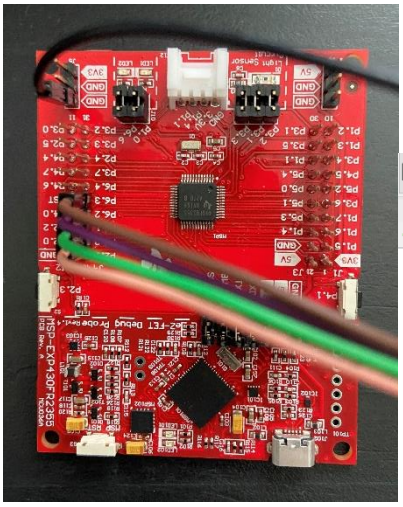


Figure 6.3a FRAM Layout

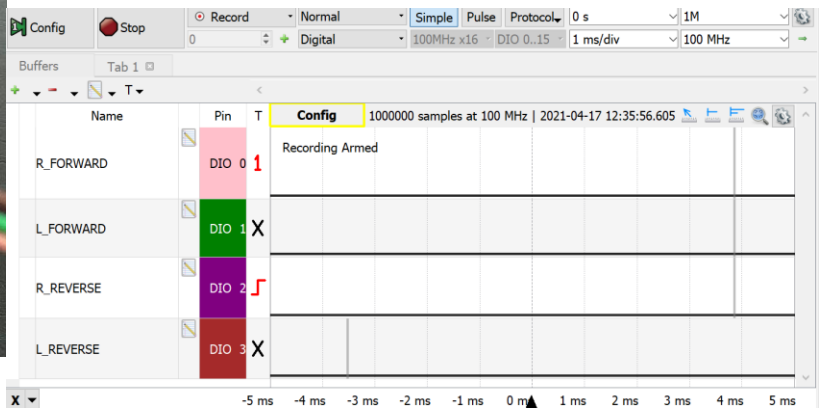


Figure 6.3b Waveforms Error Setup

6.4 FET Board Power Test

After soldering all of the N-FETs and P-FETS onto the FET board, test points TP1-TP8 should be measured with the Analog Discovery or a multimeter. When the port pins for reverse and forward are off, TP1-TP8 should all equal battery voltage. When the pins are turned on, TP1, TP2, TP5, and TP6 will be approximately half battery voltage and the remaining test points TP3, TP4, TP7, and TP8 should be close to ground potential. These test points have been measured and proved to be at expected voltages.

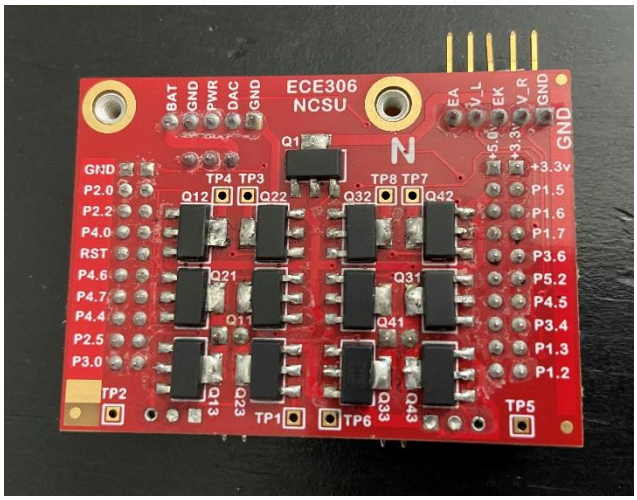


Figure 6.4 FET Tests TP1-TP8

6.5 Emitter and Detector Test

After soldering the emitter and detectors to the PCB, the board can be held above a white surface and a black surface to verify there is a substantial difference in infrared light detection. When converting the readings to a 10-bit digital reading, the white surface should yield a detection between 0-200 and the black surface should yield between 600-1000. These values were measured correctly, proving that the detection circuit works as expected.

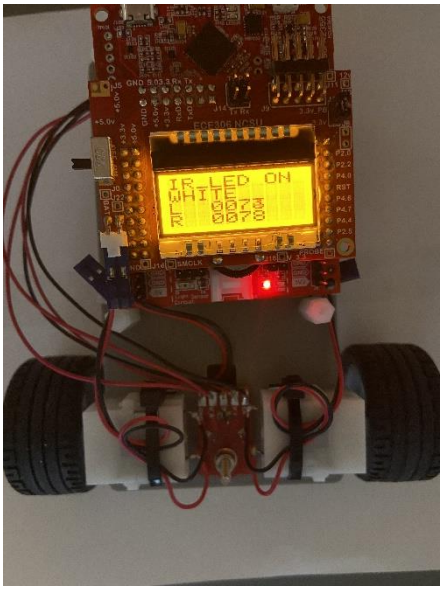


Figure 6.5a White 0-200

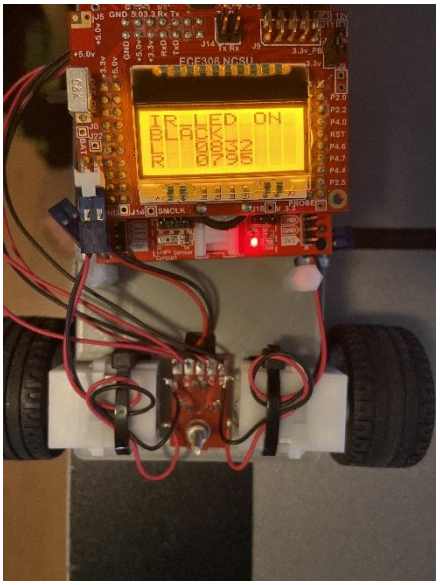
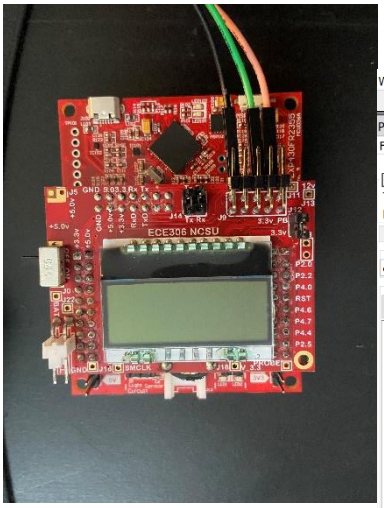


Figure 6.5b Black 600-1000

6.6 Serial Communication Testing

After configuring and initializing the serial ports to the correct baud rate, the ports need to be verified independently. For UCA1, this means connecting to the FRAM via USB and test the serial communication with an external serial terminal, when configured to a matching baud rate. It was then used to test the transmit and receive interrupts independently. After verification, they were then placed in loopback to verify functionality. Similarly, the UCA0 was tested with the AD2 and the use of its protocol function to test the transmit and receive interrupts independently. Again, like UCA1, UCA0 was placed in loopback to verify functionality.



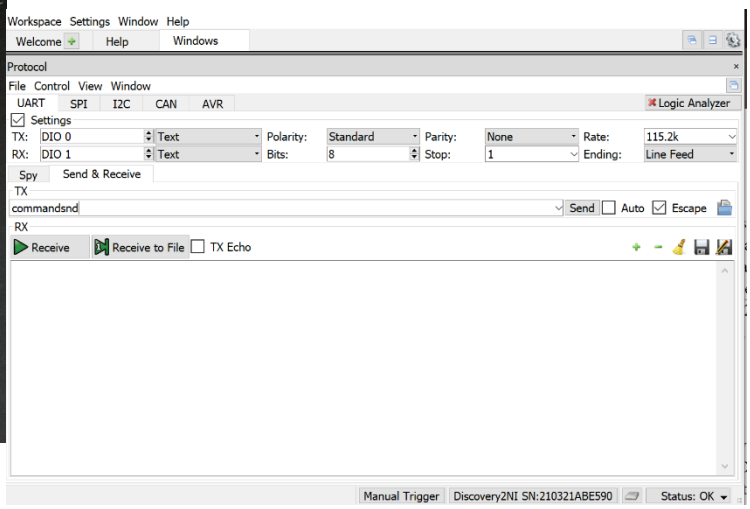


Figure 6.6a Serial Setup

Figure 6.6b Waveforms Protocol Setup

7 Software

7.1 Main

The main method of the program handles all of the execution that doesn't occur within an interrupt. It handles the initialization of the Ports, Clock, Variables and Conditions, Timers, LCD, and the ADC. The main method also contains a while (ALWAYS) loops that acts as the operating system for the car. This loop will always run and inside it resides the state machine that gives the car functionality. The state machine handles the waiting, searching, detection and orientation of the car on the black line. Halting main will prevent the car from preforming its tasks.

7.2 Ports

The structure of this code is to initialize all the ports. It initializes ports 1 through 6 when called. This sets pins to GP I/O or as a function. For project 8 the main changes that needed to be made to the ports were to pins that involved UCA. In port 1 all the select 0 bits for the transmit needed to be 0 and for select 1 the bits needed to be a 1. The same values needed to occur for ports 4 pin 2 and 3. These would be the pins for the USB transmit and receive. The main purpose of ports is to change the output, direction, or select of pins for different uses and to set them as either a 1 or 0.

7.3 Timers and Interrupts

This section of code is used to create interrupts and timers. The set up TimerB3 manages Pulse Width Modulation for the speed of the motors. The other timers such as TimerB0, TimerB1, and TimerB2, can all be configured and used with interrupts for timing in the sequence our car goes.

The timer durations are based off the clock signals, either SMCLK or ACLK, provided by the MSP430. These clock sources can be set and changed as well. Each timer used will be a base of the clock and derived to achieve an optimal time increment. Each of the timers (B0, B1, B2, and B3) can be used with each having three intervals that can be set and used within them for various car processes.

These timers can be used for interrupts that will run in the foreground with main running in the background. They are used for our detectors, switches, and duration timers for the processes that can be done by the car. Interrupts allow for the car to change states based on a given input. The interrupts are implemented alongside the timer code.

7.4 ADC and Interrupt Files

The ADC initialization and ISR code is broken up between two files for readability. The initialization function prepares the ADC for use by determining the clock used for the ADC, setting the sampling rate, establishing the resolution, setting the reference voltage, and enabling the ADC to start conversions.

The ADC ISR sequentially reads and processes data from the left detector, right detector, and thumb wheel. The value read from the thumb wheel is used to set the IR LED on or off to save power when values from the detectors are not needed. After iterating through each of the three channels, a timer is started to put a buffer between the ADC readings to allow other processes to run efficiently.

7.5 Serial Initialization and Interrupt Files

The serial initialization and ISR code are broken up between two files for readability. The initialization function configures both the UCA0 and UCA1 serial ports. The UCA0 port corresponds to the receiver and transmitter used to send and receive data using the IOT. The UCA1 port corresponds to the receiver and transmitter used to send and receive data using the USB. In both initialization functions, the baud rate and clocks are configured for proper serial communication.

The serial ISRs contain the code that receives, processes, and transmits data through each of the serial ports. As characters are received by the receiver, they are put into a ring buffer for further processing. The transmitter steps through an array and transmits the contents character by character.

7.6 Switch Interrupts File

The switch ISRs are contained within this file for SW1 and SW2. There are separate ISRs for each. Each one initializes a timer interrupt upon button press to control the debouncing of the mechanical switch. After the debounce timer is over, each switch has some lines of code that is the functionality of the switch. SW1 handles the calibration mechanism. SW2 is used for sending commands through the IOT to connect to the network. Upon SW1's first press, the white calibration is initiated and completes just a couple seconds. Upon a second press of SW1, the black line calibration is initiated and completes in just a couple seconds and then calls a calculate threshold function to complete the calibration for line threshold. When SW2 is pressed, the commands that are sent to the IOT to establish connection are used to leave the connection open for an extended period of time, retrieve connection information, and set up a port for TCP communication for receiving commands.

7.7 Command Files

The command interpretation files are used to parse through our received command for elements that determine what to execute as a result of the command. When sending a command, a start symbol and an end symbol is used. For example, a command such as $1299F1000! can be sent from a TCP application to the IOT. A command array will search through what is received to the FRAM from the IOT, putting 1299F1000 into the array. 1299 is used for the password, F indicates the direction forward, and 1000 indicates the time to execute the command in milliseconds.

7.8 Follow the Black Line File

This file contains the algorithm to maneuver the vehicle along a black line based on the ADC left and right detector values. The ADC detectors read in values that differ based on if the vehicle is positioned over a black or white surface. When both the left and right ADC detectors are positioned over the black line, the

car moves forward at a constant pace. Once either the left or right detector is no longer on the black line, the car adjusts the speed of both wheels to shift the detectors to be back over the black line. Since this adjustment is done by turning the car back towards the line, a second adjustment is completed to straighten the car back out so that it can continue moving forward with both detectors on the black line.

7.9 LCD Display Functions

These functions contain an easy encapsulation of the LCD API that can be used to put a null terminated string to the LCD screen and update the screen at the same time. In order to prevent the LCD from being updated too often they were only called every 200ms. The functions accept a pointer to a null terminated string. It also includes a function to clear the LCD by setting all the lines to " " characters.

8 Flow Chart

8.1 Main Flow Chart

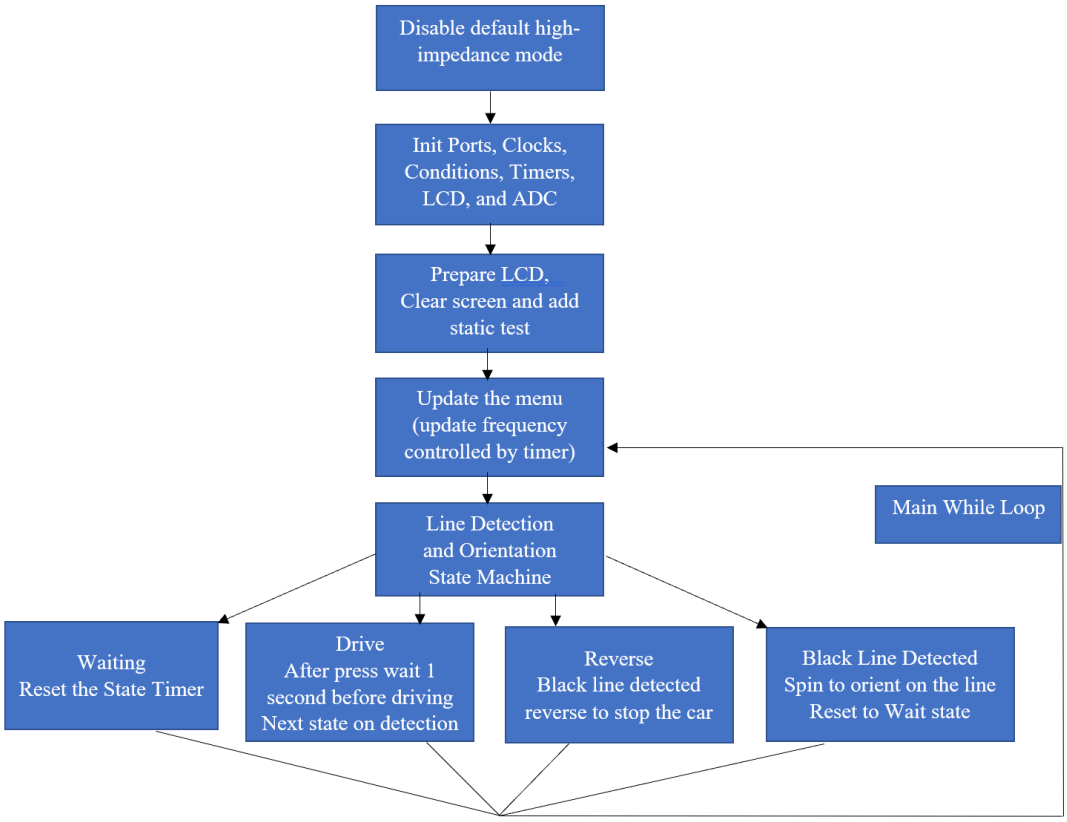


Figure 8.1 Main Flow Chart

Figure 8.1.1 shows the flow chart in main that handles the initialization calls, menu updates and the state machine used to drive, detect, and align the car to prepare for black line navigation.

8.2 Port Initialization Flow Chart

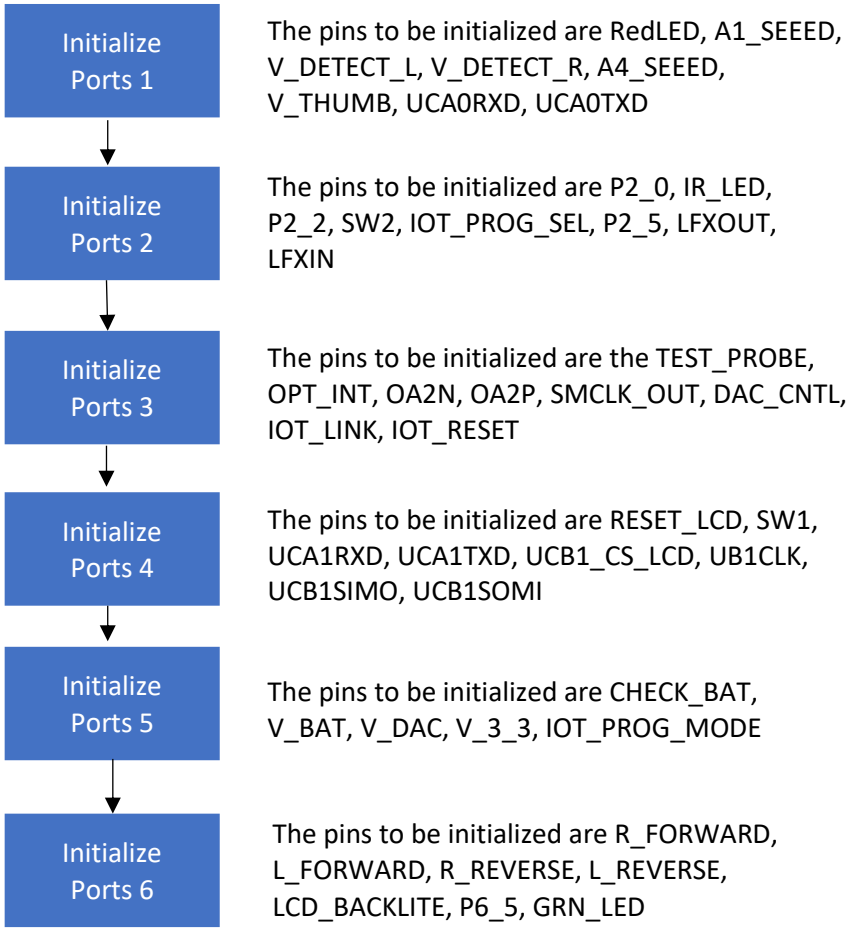


Figure 8.2 Initialization Flow Chart of Ports

This flowchart shows the order the ports are initialized as well as each pin that was initialized for the ports. The main changes to ports for project 8 involved making sure the UCA receive and transmit were GP I/O.

8.3 Timers and Interrupts Flow Chart

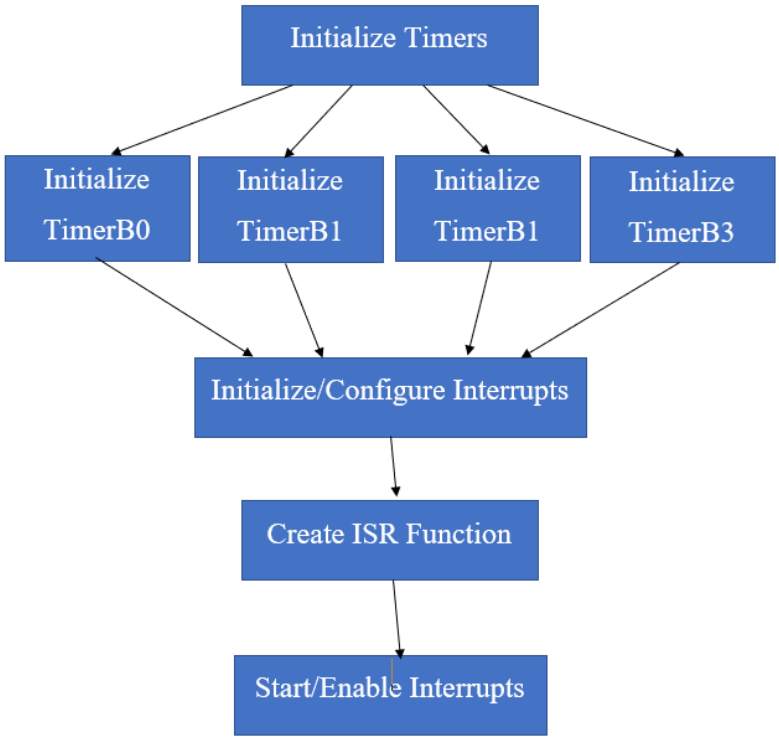


Figure 8.3 Timers and Interrupts Flow Chart

This section of the code is where the Pulse Width Modulation Timer and the Timers used for processes within our code that involve timers are enabled. The interrupts are also within this section as well as the ISRs.

8.4 ADC Initialization Flow Chart

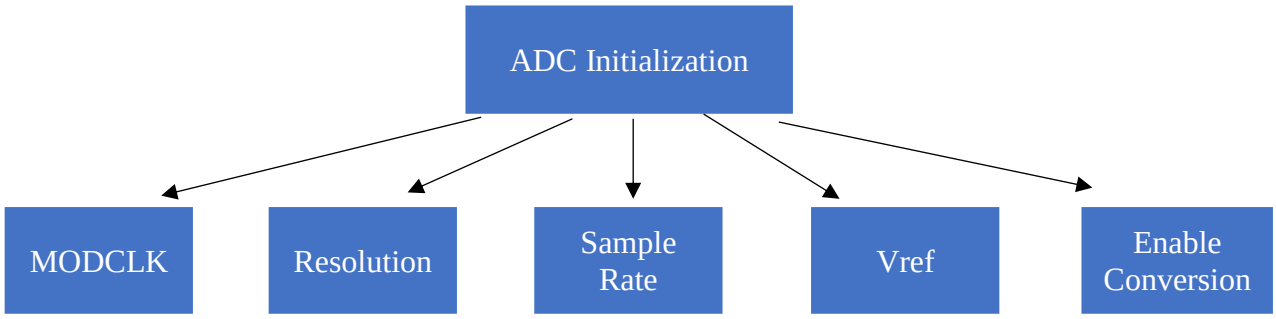


Figure 8.4 ADC Initialization Flow Chart

This flow chart shows the initialization process to configure and enable the ADC to take readings.

8.5 ADC ISR Flow Chart

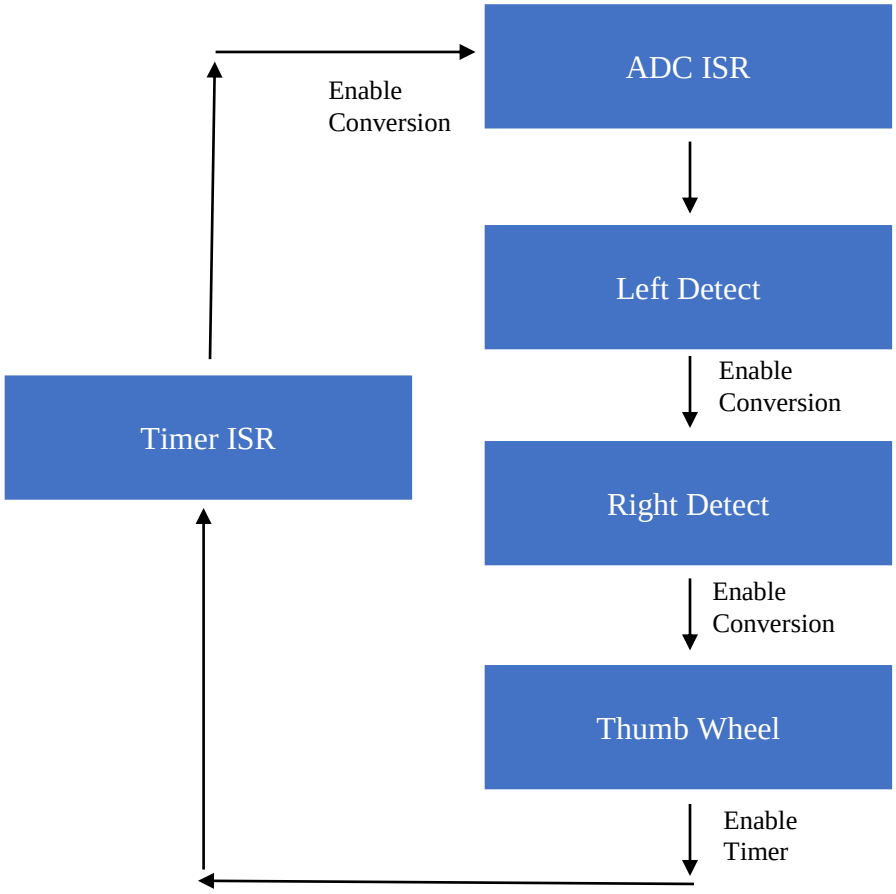


Figure 8.5 ADC ISR Flow Chart

This figure shows the process of the ADC ISR to take and convert readings from the left detector, right detector, and thumb wheel. Between each set of three readings, there is a wait period before the next reading is enabled.

8.6 Serial Initialization Flow Chart

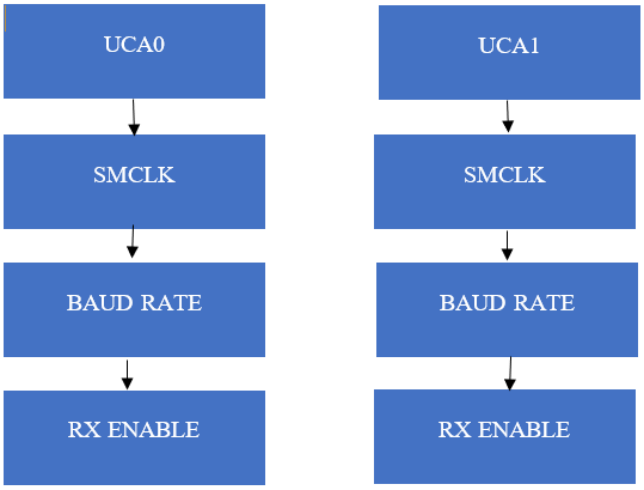


Figure 8.6 Serial Initialization Flow Chart

This flow chart shows the initialization process to configure the UCA0 and UCA1 serial ports.

8.7 Serial ISR Flow Chart

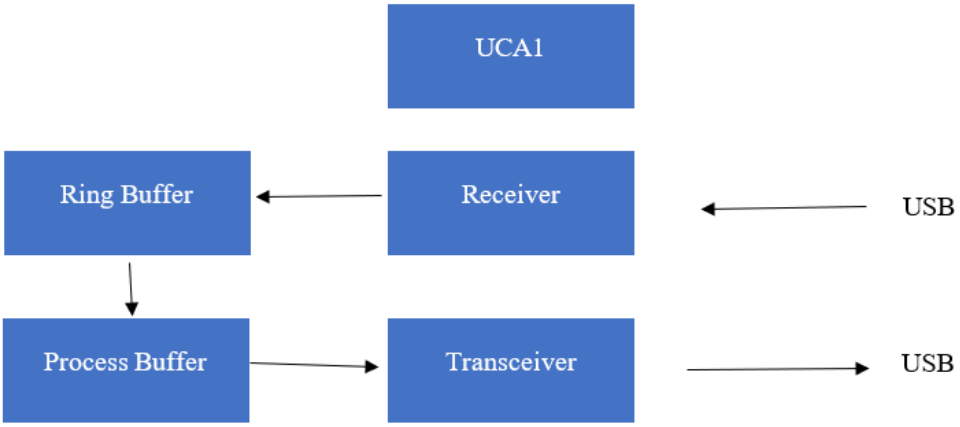


Figure 8.7 Serial ISR Flow Chart

This flow chart shows the process of the UCA1 ISR to receive characters from the USB and place them character by character into the ring buffer. The characters are then processed using a process buffer and sent back to the USB through the transceiver.

8.8 Switches Interrupts Flow Chart

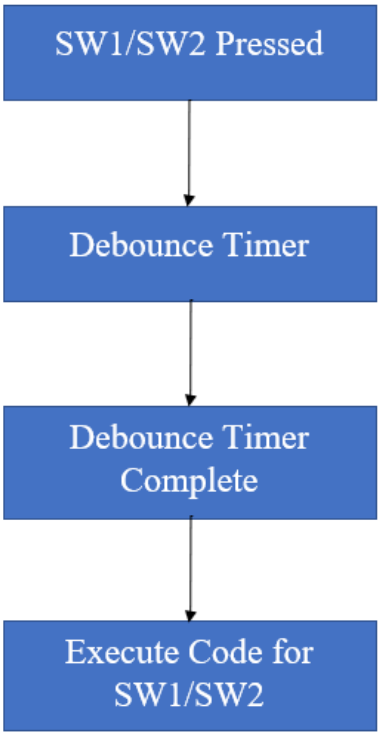


Figure 8.8 Switches Interrupts Flow Chart

This flow chart shows the process of a switch being pressed and how the debounce timer is initiated and then executes to allow for only one press of the switch to occur when the switch is pressed each time and then the commands execute.

8.9 Commands Flow Chart

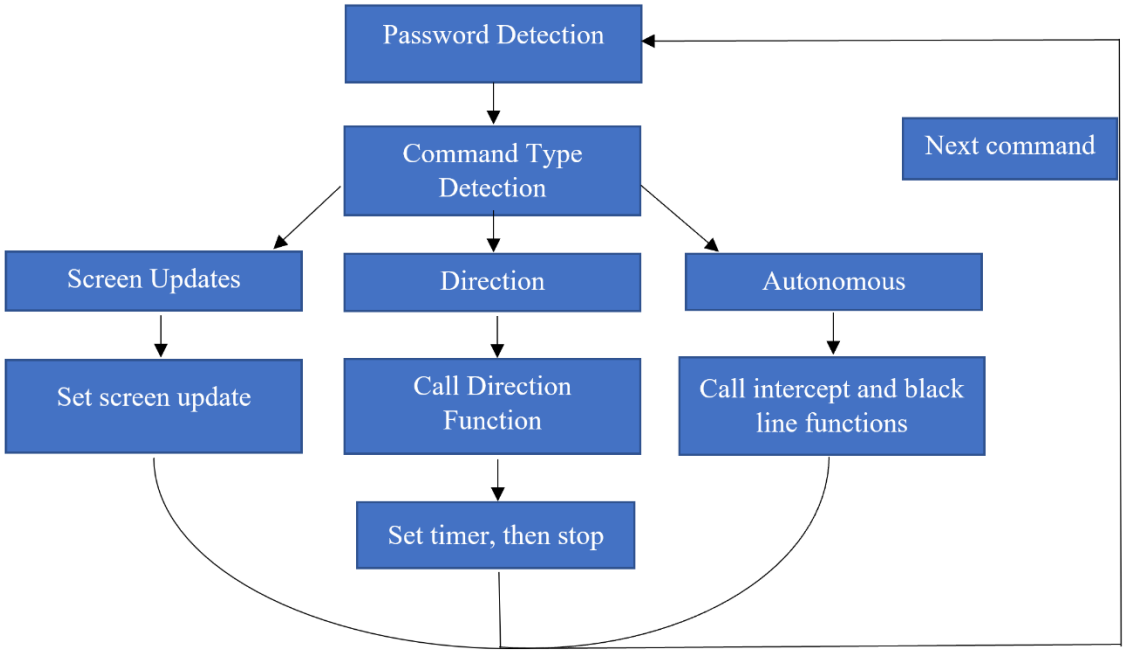


Figure 8.9 Commands Flow Chart

This flow chart shows the process of interpreting the commands being received by the IOT module and how there are different command types and durations.

8.10 Follow the Black Line Flow Chart

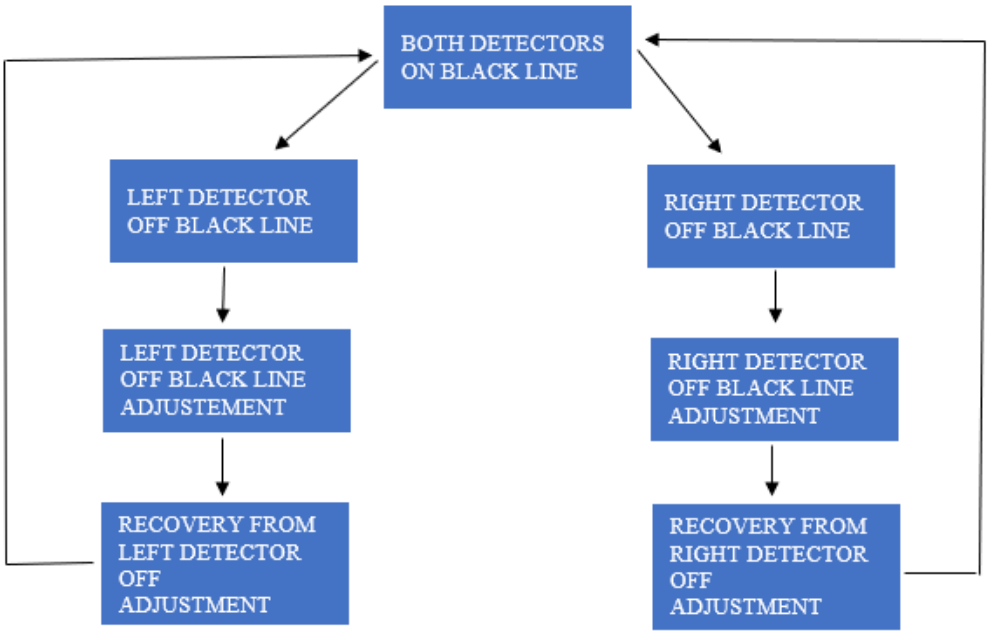


Figure 8.10 Follow the Black Line Flow Chart

This flow chart shows the process of how the ADC detectors are used to maneuver the vehicle along the black line. The right and left ADC detector values are used to determine if the car is moving off the black line and in which direction. Based on which ADC detector is off the black line, the vehicle turns to get that detector back onto the black line. After making the necessary adjustment, a second adjustment is then made to straighten the car out so that it can continue to move along the black line.

8.11 LCD Display Flow Chart

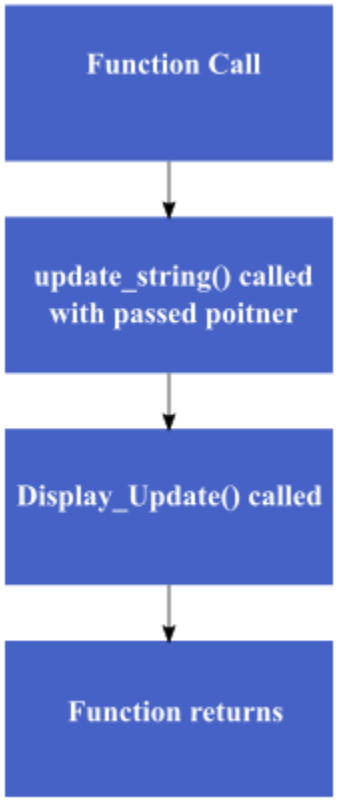


Figure 8.11 LCD Display Flow Chart

The LCD display functions work very simply with the LCD API. It calls the update_string() function and passes a string pointer and then simply updates the line with Display_Update() in order to change the display to the string referenced by the current pointer.

9 Software Listing

9.1 Main.c

```
/-------------------------------------------------------------------------
// Description: This file contains the Main Routine - "While" Operating System
//
//File Name:    main.c
//Date:         2/6/2021
//Build:        Built with IAR Embedded Workbench Version: V7.20.1.997 (7.20.1)
//-------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------
#include "functions.h"
#include "msp430.h"
#include "macros.h"
// Global Variables

 //LCD Display
 char state = WAIT;             //sets the default state to WAIT
 unsigned int state_count;      //the state counter that will get updated every 5ms
                                //a timer interrupt

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


 // Static Display text
  clear_lcd();                  //clears the LCD screen
  lcd_line1("L,R,Thumb1");      //Puts "L,R,Thumb1" to the screen

//-------------------------------------------------------------------------
// Begining of the "While" Operating System
//-------------------------------------------------------------------------
  while(ALWAYS) {               // Operating system

    menu();
    wheel_polarity_error();     //Error checking for wheel direction to prevent FET damage.

    switch(state){
    case WAIT:                  //Waiting for switch to trigger the state machine.
      state_count = RESET_COUNT;
      break;
    case DRIVE:                 //Drive until black line is intercepted.
      if(state_count < ONE_SECOND)
        break;
                                //Detection of the black line.
      if((return_vleft_average()>BLACK_LINE) && (return_vright_average() > BLACK_LINE)){
```

```
      R_stop();L_stop();        //Stop the wheels and advance the state and reset the count.
      state = REVERSE_STATE;
      state_count = RESET_COUNT;
     }else{
      R_forward(SLOW);L_forward(SLOW); //Not detected so continue forward.
     }
     break;
   case REVERSE_STATE:   //Once detected pulse reverse to stop forward progress.
     if(state_count > REVERSE_TIME){
      R_stop();L_stop();
      state = BLACK_LINE_DETECTED;    //Once reverse time is up advance state and reset count.
      state_count = RESET_COUNT;
     }else{
      R_reverse(SLOW);L_reverse(SLOW);
     }
     break;
   case BLACK_LINE_DETECTED:          //Car is now on the black line and must orient itself
     if(state_count > SPIN_TIME){             //If the car is done spinning stop and enter wait state
      R_stop();L_stop();
      state = WAIT;
     }else{
      R_forward(SLOW);L_reverse(SLOW); //Otherwise spin to orient.
     }
     break;
   default:break;
   }
  } // End of While Always
//--------------------------------------------------------------------------------
} // End of main();
```

9.2 Ports.c

```
//===========================================================================
==
// File Name : ports.c
//
// Description: This file contains the Initialization for all port pins
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//===========================================================================
==

//---------------------------------------------------------------------
#include "macros.h"
#include  "functions.h"
#include  "msp430.h"
#include <string.h>
#include "ports.h"

extern unsigned int useaclock;

// Function Prototypes
void Init_Ports(void);
void Init_Port1(void);
void Init_Port2(void);
void Init_Port3(void);
void Init_Port4(void);
void Init_Port5(void);
void Init_Port6(void);

//===========================================================================
==
// Function name: ports
//
// Description: This function contains the port function calls for all 6 ports.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//===========================================================================
==

void Init_Ports(void){
```

```
  Init_Port1();
  Init_Port2();
  Init_Port3();
  Init_Port4();
  Init_Port5();
  Init_Port6();
}
//=====================================================================
==
// Function name: Initialization of Ports 1
//
// Description: This function initalizes the pins in the first port.
// The pins to be intialized are the Red LED, A1_SEEED, V_DETECT_L, V_DETECT_R,
// A4_SEEED, V_THUMB, UCA0RXD, UCA0TXD
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
//---------------------------------------------------------------------
// Init_Port1
// Purpose: Initialize Port 1
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCBOSTE
// 1 0 0 - SMCLK
// 1 0 1 - VSS
// 1 1 X - Analog input A0 - ADC, Comparator_D input CD0, Vref- External applied reference
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCBOCLK
// 0 1 1 - ACLK
// 1 0 1 - TA1 input clock
// 1 0 0 - VSS
// 1 1 X - Analog input A1 - ADC, Comparator_D input CD1, Input for an external reference voltage to
the ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCB0SIMo/UCB0SDA
// 1 0 0 - TBOTRG
// 1 1 X - Analog input A2 - ADC, Input for an external reference voltage to the ADC
```

```
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCB0SOMI/UCB0SCL
// 1 1 X - Analog input A3 - ADC
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA0STE
// 1 1 X - Analog input A4 - ADC
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA0CLK
// 1 1 X - Analog input A5 - ADC, OA10
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA0RXD/UCA0SOMI
// 1 0 0 - TB0.CCI1A
// 1 0 1 - TB0.1
// 1 1 X - Analog input A6 - ADC, OA1-
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA0TXD/UCA0SIMO
// 1 0 0 - TB0.CCI2A
// 1 0 1 - TB0.2
// 1 1 X - Analog input A7 - ADC, Input for an external reference voltage to the ADC, OA1+
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
==
 void Init_Port1(void){
   P1OUT = port_reset;              // P1 set Low
   P1DIR = port_reset;              // Set P1 direction to output
  //P1 PIN 0
   P1SEL0 &= ~RED_LED;              // REDLED GPIO operation
   P1SEL1 &= ~RED_LED;              // REDLED GPIO operation
   P1OUT &= ~RED_LED;               // Initial Value = Low / Off
   P1DIR |= RED_LED;                // Direction = output
  //P1 PIN 1
```

```
    P1SELC |= A1_SEEED;              // A1_SEEED operation
     //P1 PIN 2
    P1SELC |= V_DETECT_L;
     //P1 PIN 3
    P1SELC |= V_DETECT_R;
  //P1 PIN 4
    P1SELC |= A4_SEEED;
     //P1 PIN 5
    P1SELC |= V_THUMB;
     //P1 PIN 6
    P1SEL0 |= UCA0RXD;
    P1SEL1 &= ~UCA0RXD;
  //P1 PIN 7
    P1SEL0 |= UCA0TXD;
    P1SEL1 &= ~UCA0TXD;
  }

//=========================================================================
==
// Function name: Initialization of Ports 2
//
// Description: This function initalizes the pins in the first port.
// The pins to be intialized are the Port 2 pin 0, IR_LED, port 2 pin 2, Switch 2,
// IOT program select, port 2 pin 5, LFXOUT, LFXIN
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
// Init_Port2
// Purpose: Initialize Port 2
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1.CCI1A: TB1 CCR1 capture: CCI1A input
// 0 1 1 - TB1.1: TB1 CCR1 compare: Out1
// 1 0 1 - Comparator_D input CD0
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1CCI2A: TB1 CCR2 capture: CCI2A input
// 0 1 1 - TB1.2: TB1 CCR2 compare: Out2
// 1 0 1 - Comparator_D input CD1
//
// Various options for Pin 2
```

```
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1CLK
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1TRG
// 0 1 1 - VSS
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Comparator_D input CD4
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Comparator_D input CD5
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - MCLK
// 0 1 1 - VSS
// 1 0 X - XOUT *Slave in, master out - eUSCI_B0 SPI mode, I2C data - eUSCI_B0 I2C mode?
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB0CLK
// 0 1 1 - VSS
// 1 0 X - XIN: Slave out, master in - eUSCI_B0 SPI mode, I2C clock - eUSCI_B0 I2C mode?
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
==
  void Init_Port2(void){
   P2OUT = port_reset;              // P2 set Low
   P2DIR = port_reset;              // 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 = Low / 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
    //P2OUT |= SW2;               // Configure pullup resistor
    P2PUD |= SW2;                 // Configure pullup resistor
    P2DIR &= ~SW2;                // Direction = input
    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 &= ~IOT_PROG_SEL;       // IOT_PROG_SEL GPIO operation
   P2SEL1 &= ~IOT_PROG_SEL;       // IOT_PROG_SEL GPIO operation
   P2OUT &= ~IOT_PROG_SEL;        // Initial Value = Low / Off
   P2DIR |= IOT_PROG_SEL;         // 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
 }

//=========================================================================
==
// Function name: Initialization of Ports 3
//
// Description: This function initalizes the pins in the first port.
//
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
// Init_Port3
// Purpose: Initialize Port 3
//
```

```
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - MCLK
// 0 1 1 - VSS
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA2O
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA2-
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA2+
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 0 1 - SMCLK
// 0 1 0 - VSS
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA3O
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA3-
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - OA3+
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
==
  void Init_Port3(void){
    P3OUT = port_reset;                 // P3 set Low
```

```
  P3DIR = port_reset;              // 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;            // Direction = output
 //P3 PIN 1
  P3SEL0 &= ~OPT_INT;              // TEST_PROBE GPIO operation
  P3SEL1 &= ~OPT_INT;              // TEST_PROBE GPIO operation
  P3DIR &= ~OPT_INT;               // Direction = input
 //P3 PIN 2
  P1SEL0 |= OA2N;
  P1SEL1 |= OA2N;
 //P3 PIN 3
  P1SEL0 |= OA2P;
  P1SEL1 |= OA2P;
 //P3 PIN 4
    if(useaclock==USE_SMCLK){
  P3SEL0 |= SMCLK_OUT;             // SMCLK_OUT GPIO operation
  P3SEL1 &= ~SMCLK_OUT;            // SMCLK_OUT GPIO operation
  P3DIR |= SMCLK_OUT;              // Direction = input
  }
  else{                            //otherwise it will be GPI/O
  P3SEL0 &= ~SMCLK_OUT;            // SMCLK_OUT GPIO operation
  P3SEL1 &= ~SMCLK_OUT;            // SMCLK_OUT GPIO operation
  P3DIR &= ~SMCLK_OUT;             // Direction = input
  }
 //P3 PIN 5
  P3SEL0 &= ~DAC_CNTL;             // SMCLK_OUT GPIO operation
  P3SEL1 &= ~DAC_CNTL;             // SMCLK_OUT GPIO operation
 P3DIR &= ~DAC_CNTL;               // Direction = input
 //P3 PIN 6
  P3SEL0 &= ~IOT_LINK;             // SMCLK_OUT GPIO operation
  P3SEL1 &= ~IOT_LINK;             // SMCLK_OUT GPIO operation
  //P3OUT &= ~IOT_LINK;            // SMCLK_OUT = Low / Off
  P3DIR &= ~IOT_LINK;              // Direction = input
 //P3 PIN 7
  P3SEL0 &= ~IOT_RESET;            // IOT_RESET GPIO operation
  P3SEL1 &= ~IOT_RESET;            // IOT_RESET GPIO operation
  //P3OUT &= ~IOT_RESET;           // Set IOT_RESET On [Low]
  P3DIR &= ~IOT_RESET;             // Set IOT_RESET direction to output
 }

//=====================================================================
==
// Function name: Initialization of Ports 4
//
// Description: This function initalizes the pins in the first port.
// The pins to be intialized are the RESET_LCD, Switch 1, UCA1RXD, UCA1TXD,
```

```
// UCB1_CS_LCD, UB1CLK, UCB1SIMO, UCB1SOMI
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
// Init_Port4
// Purpose: Initialize Port 4
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA1STE
// 1 0 0 - UCA1RXD, TB3.CCI2B
// 1 0 1 - UCA1TXD logic-AND TB3.2B
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA1CLK
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA1RXD/UCA1SOMI
// 1 0 X - (UCA1RXD)'
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCA1TXD/UCA1SIMO
// 1 0 X -  (UCA1TXD)'
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UB1STE
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UB1CLk
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCB1SIMO/UCB1SDA
```

```
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 X - UCB1SOMI/UCB1SCL
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
==

  void Init_Port4(void){
P4OUT = port_reset;                 // P1 set Low
P4DIR = port_reset;                 // Set P1 direction to output

// P4 PIN 0
P4SEL0 &= ~RESET_LCD;               // RESET_LCD GPIO operation
P4SEL1 &= ~RESET_LCD;               // RESET_LCD GPIO operation
P4OUT &= ~RESET_LCD;                // Set RESET_LCD On [Low]
P4DIR |= RESET_LCD;                 // Set RESET_LCD direction to output

// P4 PIN 1
P4SEL0 &= ~SW1;                     // SW1 GPIO operation
P4SEL1 &= ~SW1;                     // SW1 GPIO operation
//P4OUT |= SW1;                     // Configure pullup resistor
P4PUD |= SW1;                       // Configure pullup resistor
P4DIR &= ~SW1;                      // Direction = input
P4REN |= SW1;                       // Enable pullup resistor
 P4IES |= SW1;                      // P4.0 Hi/Lo edge interrupt
 P4IFG &= ~SW1;                     // Clear all P4.6 interrupt flags
 P4IE |= SW1;                       // P4.6 interrupt enabled

// P4 PIN 2
  P4SEL0 |= UCA1TXD;                // USCI_A1 UART operation
  P4SEL1 &= ~UCA1TXD;               // USCI_A1 UART operation

// P4 PIN 3
  P4SEL0 |= UCA1RXD;                // USCI_A1 UART operation
  P4SEL1 &= ~UCA1RXD;               // 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 SPI_CS_LCD Off [High]
  P4DIR |= UCB1_CS_LCD;             // Set SPI_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
//-------------------------------------------------------------------
}

//=====================================================================
==
// Function name: Initialization of Ports 5
//
// Description: This function initalizes the pins in the first port.
// The pins to be intialized are the Check Battery, V_BAT, V_DAC, V_3_3,
// IOT Program Mode
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no global variables
//
// Init_Port5
// Purpose: Initialize Port 5
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 I - TB2.CCI1A: TA0 CCR1 capture: CCI1A input
// 0 1 O - TB2.1
// 1 0 X - MFM.RX
// 1 1 X - Analog input A8 - ADC
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 I - TB2.CCI2A: TA0 CCR2 capture: CCI2A input,
// 0 1 O - TB2.2: TA0 CCR1 compare: Out1
// 1 0 X - MFM.TX
// 1 1 X - Analog input A9 - ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 I - TB2CLK
```

```
// 0 1 O - VSS
// 1 1 X - Analog input A10 - ADC
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 I - TB2TRG
// 0 1 O - VSS
// 1 1 X - Analog input A11 - ADC
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
//
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
==
void Init_Port5(void){
  P5OUT = port_reset;                 // P5 set Low
  P5DIR = port_reset;                 // Set P5 direction to output
  //P5 PIN 0
    P5SEL0 &= ~CHECK_BAT;             // CHECK_BAT GPIO operation
    P5SEL1 &= ~CHECK_BAT;             // CHECK_BAT GPIO operation
   P5DIR &= ~CHECK_BAT;               // Set CHECK_BAT direction to output
  //P5 PIN 1
    P5SELC |= V_BAT;                  // V_BAT operation
   //P5 PIN 2
    P5SEL0 |= V_DAC;                  // V_DAC operation
    P5SEL1 |= V_DAC;                  // V_DAC operation
  //P5 PIN 3
    P5SELC |= V_3_3;                  // V_3_3 operation
   //P5 PIN 4
    P5SEL0 &= ~IOT_PROG_MODE; // IOT_PROG_MODE operation
    P5SEL1 &= ~IOT_PROG_MODE; // IOT_PROG_MODE operation
    P5DIR &= ~IOT_PROG_MODE;   // Set IOT_PROG_MODE direction to output
}

//=============================================================================
==
// Function name: Initialization of Ports 6
//
// Description: This function initalizes the pins in the first port.
// The pins to be intialized are the Right wheel forward, Left Wheel forward, Right wheel reverse
// Left wheel reverse, LCD Backlite, port 6 pin 5, Green LED
//
// Passed : no variables passed
// Locals: no variables declared
```

```
// Returned: no values returned
// Globals: no global variables
//
// Init_Port6
// Purpose: Initialize Port 6
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI1A: TB3 CCR1 capture: CCI1A input
// 0 1 1 - TB3.1: TB3 CCR1 compare: Out1
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI2A: TB3 CCR2 capture: CCI2A input
// 0 1 1 - TB3.2: TB3 CCR2 compare: Out2
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI3A: TB3 CCR3 capture: CCI13A input
// 0 1 1 - TB3.3: TB3 CCR3 compare: Out3
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI4A: TB3 CCR4 capture: CCI14A input
// 0 1 1 - TB3.4: TB3 CCR4 compare: Out4
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI5A: TB3 CCR5 capture: CCI15A input
// 0 1 1 - TB3.5: TB3 CCR5 compare: Out5
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB3.CCI6A: TB3 CCR6 capture: CCI16A input
// 0 1 1 - TB3.6: TB3 CCR6 compare: Out6
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 1 - TB3CLK
// 0 1 0 - VSS
//
```

```
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
==
void Init_Port6(void){
  P6OUT = port_reset;                 // P1 set Low
  P6DIR = port_reset;                 // Set P1 direction to output
  //P6 PIN 0
    P6SEL0 |= R_FORWARD;              // R_FORWARD operation
    P6SEL1 &= ~R_FORWARD;             // R_FORWARD operation
   P6DIR |= R_FORWARD;                // Set R_FORWARD direction to output
  //P6 PIN 1
    P6SEL0 |= L_FORWARD;              // L_FORWARD operation
    P6SEL1 &= ~L_FORWARD;             // L_FORWARD operation
   P6DIR |= L_FORWARD;                // Set L_FORWARD direction to output
  //P6 PIN 2
    P6SEL0 |= R_REVERSE;              // R_REVERSE operation
    P6SEL1 &= ~R_REVERSE;             // R_REVERSE operation
   P6DIR |= R_REVERSE;                // Set R_REVERSE direction to output
  //P6 PIN 3
    P6SEL0 &= ~L_REVERSE;             // L_REVERSE operation
    P6SEL1 &= ~L_REVERSE;             // L_REVERSE operation
    P6DIR |= L_REVERSE;               // Set L_REVERSE direction to output
  //P6 PIN 4
    P6SEL0 &= ~LCD_BACKLITE;          // LCD_BACKLITE operation
    P6SEL1 &= ~LCD_BACKLITE;          // LCD_BACKLITE operation
    P6OUT |= LCD_BACKLITE;            // Set LCD_BACKLITE Off [Low]
    P6DIR |= LCD_BACKLITE;            // Set LCD_BACKLITE direction to output
  //P6 PIN 5
    P6SEL0 &= ~P6_5;                  // P6_5 GPIO operation
    P6SEL1 &= ~P6_5;                  // P6_5 GPIO operation
    P6DIR &= ~P6_5;                   // Direction = input
  //P6 PIN 6
   // P6SEL0 &= ~GRN_LED;              // GRNLED GPIO operation
    //P6SEL1 &= ~GRN_LED;             // GRNLED GPIO operation
    //P6OUT &= ~GRN_LED;              // Initial Value = Low / Off
    //P6DIR |= GRN_LED;               // Direction = output
}
```

9.3 Timers.c

```
//-------------------------------------------------------------------------------
//
//  Description: This file contains the Clock Initialization and Operation
//
//  Spring 2021
//  Built with IAR Embedded Workbench Version: V7.20.1
//-------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include "macros.h"

// Variable Delcarations
unsigned int TB0_50ms_count;
extern unsigned int count_debounce_SW1;
extern unsigned int count_debounce_SW2;
extern unsigned int sw1DB;
extern unsigned int sw2DB;
extern unsigned int sw1_position;
extern unsigned int sw2_position;
extern volatile unsigned int mcc_timer;
extern volatile unsigned int SW1_event;
extern volatile unsigned int SW2_event;
volatile unsigned int delay_duration;


// Function Prototypes


// Timer Function Calls
void Init_Timers(void){
  Init_Timer_B0();
  Init_Timer_B3();
}


//-------------------------------------------------------------------------------
// Timer B0 initialization sets up both B0_0 , B0_1 B0_2 and overflow
void Init_Timer_B0 (void){
   TB0CTL = TBSSEL__SMCLK;             // SMCLK source
   TB0CTL |= TBCLR;                    // Resets TB0R , clock divider, count direction
   TB0CTL |= MC__CONTINOUS;            // Continuous up
   TB0CTL |= ID_1;                     // Divide clock by 2

   TB0EX0 = TBIDEX_7;                  // 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 ;                    // Disable Overflow Interrupt
   TB0CTL &= ~TBIFG ;                   // Clear Overflow Interrupt flag
}
//------------------------------------------------------------------------------


#pragma vector = TIMER0_B0_VECTOR
__interrupt void Timer0_B0_ISR(void){
//------------------------------------------------------------------------------
// TimerB0 0 Interrupt handler
//------------------------------------------------------------------------------
// GLOBAL VARIABLES - TB0_50ms_count

// Add What you need happen in the interrupt
  TB0_50ms_count++;
  TB0CCR0 += TB0CCR0_INTERVAL;      // Add Offset to TBCCR0
  if(TB0_50ms_count >= delay_duration){    // pause delay for each sequences
    TB0_50ms_count = FALSE;
    TB0CCTL0 &= ~CCIE;                  // disable timer interrupt
    TB0CCR0 = FALSE;
  }
//------------------------------------------------------------------------------
}

#pragma vector = TIMER0_B1_VECTOR
__interrupt void TIMER0_B1_ISR(void){
//------------------------------------------------------------------------------
// TimerB0 1-2, Overflow Interrupt Vector (TBIV) handler
//------------------------------------------------------------------------------
// GLOBAL VARIABLES - count_debounce_SW1, count_debounce_SW2, sw1DB, sw2DB
//            sw1_position, sw2_position, SW1_event, SW2_event
switch(__even_in_range (TB0IV,14)){
   case FALSE: break;                   // No interrupt
   case CCR1:  // CCR1 to control Switch Debounce
// Add What you need happen in the interrupt
     TB0CCR1 += TB0CCR1_INTERVAL;  // Add Offset to TBCCR1
     count_debounce_SW1++;
     if(count_debounce_SW1 >= DEBOUNCE_TIME){
       sw1DB = OKAY;                    // switch has debounced
       P4IE |= SW1;                     // interrupt enabled
```

```
      sw1_position = RELEASED;            // switch has been released
      //disable CCR1 and clear
      TB0CCTL1 &= ~CCIE;
      TB0CCR1 = FALSE;
      count_debounce_SW1 = DEBOUNCE_RESTART;
      SW1_event = FALSE;                  // for testing debounce
     }
     count_debounce_SW2++;
     if(count_debounce_SW2 >= DEBOUNCE_TIME){
      sw2DB = OKAY;                       // switch has debounced
      P2IE |= SW2;                        // interrupt enabled
      sw2_position = RELEASED;            // switch has been released
      // disable CCR1 and clear
      TB0CCTL1 &= ~CCIE;
      TB0CCR1 = FALSE;
      count_debounce_SW2 = DEBOUNCE_RESTART;
      SW2_event = FALSE;                  // for testing debounce
      if(SW1_event == FALSE){             // to disable timer interrupt if neither switch is using it
       TB0CCTL0 |= CCIE;                  // CCR0 enable BACKLITE interrupt
      }
     }
     break;
   case CCR2:  // CCR2 to time for motor controls
// Add What you need happen in the interrupt
     TB0CCR2 += TB0CCR2_INTERVAL; // Add Offset to TBCCR2
     mcc_timer++;
     if(mcc_timer >= SAMP_RATE){
      mcc_timer = FALSE;
      TB0CCTL2 &= ~CCIE;
      TB0CCR2 = FALSE;
      ADCCTL0 |= ADCSC;                   // start next adc sample
     }
          break;
   case OVERFLOW:                         // overflow
// Add What you need happen in the interrupt
     break;
   default: break;
  }
//------------------------------------------------------------------------------
}


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

9.4 ADC.c

```
//===============================================================================
//  File Name : ADC.c
//  Description: This file contains the initialization function for the ADC
//
//  Globals: none
//
//  Lori Glenn
//  March 2021
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//===============================================================================
#include  "functions.h"
#include "macros.h"
#include  "msp430.h"
#include <string.h>

//===============================================================================
// Function name: Init_ADC
//
// Description: This function contains the initialization settings for the ADC
//
// Passed: no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: none
//
// Author: Lori Glenn
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//===============================================================================
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 = RESET_STATE           // Reset
  ADCCTL0 |= ADCSHT_2;            // 16 ADC clocks
  ADCCTL0 |= ADCMSC;              // MSC
  ADCCTL0 |= ADCON;               // ADC ON

  // ADCCTL1 Register
  ADCCTL2 = RESET_STATE;          // 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 = RESET_STATE;                // Reset
  ADCCTL2 |= ADCPDIV0;                  // ADC pre-divider 00b = Pre-divide by 1
  ADCCTL2 |= ADCRES_1;                  // ADC resolution 10b = 10 bit (14 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 |= ADCSREF_0;                // VREF - 000b = {VR+ = AVCC and VR– = AVSS }
  ADCMCTL0 |= ADCINCH_2;                // V_DETECT_L (0x04) Pin 2 A2
  ADCIE |= ADCIE0;                      // Enable ADC conv complete interrupt
  ADCCTL0 |= ADCENC;                    // ADC enable conversion.
  ADCCTL0 |= ADCSC;                     // ADC start conversion.
}
```

9.5 ADC_interrupts.c

```
//===================================================================
//  File Name : ADC_interrupts.c
//  Description: This file contains the ISRs for reading and converting values
//               from the thumb wheel, left detect, and right detect
//               using the ADC
//  Globals: ADC_Thumb_Wheel, ADC_Left_Detect, ADC_Right_Detect,
//               ADC_Channel, ir_led_is_on, on_black_line_left,
//               on_black_line_right, display_line, display,
//               display_changed
//
//  Lori Glenn
//  March 2021
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//===================================================================
#include  "functions.h"
#include "macros.h"
#include  "msp430.h"
#include <string.h>

// Globals
volatile int ADC_Thumb_Wheel;
volatile int ADC_Left_Detect;
volatile int ADC_Right_Detect;
volatile int ADC_Channel;
char ir_led_is_on;
char on_black_line_left;
char on_black_line_right;
extern char display_line[DISPLAY_HEIGHT][DISPLAY_WIDTH_PLUS_ONE];
extern char *display[DISPLAY_HEIGHT];
extern volatile unsigned char display_changed;
char adc_char_R[LENGTH_OF_DISPLAY_ARRAY];
char adc_char_L[LENGTH_OF_DISPLAY_ARRAY];
char adc_char[LENGTH_OF_DISPLAY_ARRAY];

//===================================================================
// Function name: ADC_ISR
//
// Description: This ISR reads and converts the left detect, right
//              detect, and thumb wheel values
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: ADC_Left_Detect, on_black_line_left, ADC_Right_Detect,
//          on_black_line_right, ADC_Thumb_Wheel, ir_led_is_on,
//          ADC_Channel
```

```
//
// Author: Lori Glenn
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//=============================================================================
//-----------------------------------------------------------------------------
#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void){
  switch(__even_in_range(ADCIV,ADCIV_ADCIFG)){
    case ADCIV_NONE:
      break;
    case ADCIV_ADCOVIFG:                // When a conversion result is written to the
                                        //      ADCMEM0 before its previous conversion
                                        //      result was read
      break;
    case ADCIV_ADCTOVIFG:               // ADC conversion-time overflow
      break;
    case ADCIV_ADCHIIFG:                // Window comparator interrupt flags
      break;
    case ADCIV_ADCLOIFG:                // Window comparator interrupt flag
      break;
    case ADCIV_ADCINIFG:                // Window comparator interrupt flag
      break;
    case ADCIV_ADCIFG:                  // ADCMEM0 memory register with the conversion
                                        //      result
      ADCCTL0 &= ~ADCENC;               // Disable ENC bit
      switch(ADC_Channel++){
        case CHANNEL_2:                         // channel 2 interrupt
          ADC_Left_Detect = ADCMEM0;    // Move result into Global
          ADCMCTL0 &= ~ADCINCH_2;       // Disable Last - channel A3
          ADCMCTL0 |= ADCINCH_3;
          HEXtoBCD_L(ADC_Left_Detect);  // Convert result to String
          if(ADC_Left_Detect > LINE_THRESHOLD) {on_black_line_left = TRUE;}
          else{on_black_line_left = FALSE;}
          ADCCTL0 |= ADCENC;            // Enable Conversions
          ADCCTL0 |= ADCSC;
          break;
        case CHANNEL_3:                         // channel 3 interrupt
          ADC_Right_Detect = ADCMEM0;   // Move result into Global
          ADCMCTL0 &= ~ADCINCH_3;       // Disable Last - channel A3
          ADCMCTL0 |= ADCINCH_5;        // Enable Next - channel A5
          HEXtoBCD_R(ADC_Right_Detect); // Convert result to String
          if(ADC_Right_Detect > LINE_THRESHOLD) {on_black_line_right = TRUE;}
          else{on_black_line_right = FALSE;}
          ADCCTL0 |= ADCENC;            // Enable Conversions
          ADCCTL0 |= ADCSC;
          break;
        case CHANNEL_5:                         // channel 5 interrupt
```

```
            ADC_Thumb_Wheel = ADCMEM0;          // Channel A5
            if(ADC_Thumb_Wheel > THUMB_WHEEL_THRESHOLD) {
                  P2OUT  |= IR_LED;
                  ir_led_is_on = TRUE;
           }
            else{
                  P2OUT &= ~IR_LED;
                  ir_led_is_on = FALSE;
           }
            ADCMCTL0 &= ~ADCINCH_5;             // Disable Last - channel A5
            ADCMCTL0 |= ADCINCH_2;              // Enable Next - channel A2
            HEXtoBCD(ADC_Thumb_Wheel);
            ADCCTL0 &= ~ADCSC;
            TB1CCR0 = TB1CCR0_INTERVAL;
            TB1CCTL0 |= CCIE;                    // enable timer interrupt for B1 CCR0
            ADC_Channel = RESET_STATE;
           break;
         }
       // ADCCTL0 |= ADCENC;                    // Enabled in Timer ISR after all three reads
       // ADCCTL0 |= ADCSC;                     // Enabled in Timer ISR after all three reads
        default:
        break;
    }
}
```

9.6 Serial.c

```
//=====================================================================
// Function name: Init_Serial_UCA0
// Description: This function initializes the serial port UCA0
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: USB_Char_Rx, usb_rx_ring_wr, usb_rx_ring_rd
//
// Author: Lori Glenn
// Date: April 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//=====================================================================
void Init_Serial_UCA0(void){
  int i;
  for(i=BEGINNING; i<SMALL_RING_SIZE; i++){
    USB_Char_Rx[i] = NULL_CHAR;                    // USB Rx Buffer
  }
  usb_rx_ring_wr = BEGINNING;
  usb_rx_ring_rd = BEGINNING;
  for(i=BEGINNING; i<LARGE_RING_SIZE; i++){
    USB_Char_Tx[i] = NULL_CHAR;                    // USB Tx Buffer
  }
  usb_tx_ring_wr = BEGINNING;
  usb_tx_ring_rd = BEGINNING;

  // Configure UART 0
  UCA0CTLW0 = RESET_STATE;                          // Use word register
  UCA0CTLW0 |= UCSWRST;                             // Set Software reset enable
  UCA0CTLW0 |= UCSSEL__SMCLK;                       // Set SMCLK as fBRCLK

  UCA0BRW = BAUD_RATE_FOR_115200;                   // 9,600 Baud
  UCA0MCTLW = BAUD_CONCAT_FOR_115200;
  UCA0CTLW0 &= ~ UCSWRST;                           // Set Software reset enable
  UCA0TXBUF = NULL_CHAR;
  UCA0IE |= UCRXIE;                                 // Enable RX interrupt
}

//=====================================================================
// Function name: Init_Serial_UCA1
// Description: This function initializes the serial port UCA1
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: USB_Char_Rx, usb_rx_ring_wr, usb_rx_ring_rd
```

```
//
// Author: Lori Glenn
// Date: April 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//-------------------------------------------------------------------------------
void Init_Serial_UCA1(void){
  received_from_pc = OFF;
  int i;
  for(i=BEGINNING; i<SMALL_RING_SIZE; i++){
    USB_Char_Rx[i] = NULL_CHAR;                    // USB Rx Buffer
  }
  usb_rx_ring_wr = BEGINNING;
  usb_rx_ring_rd = BEGINNING;
  for(i=BEGINNING; i<LARGE_RING_SIZE; i++){
    USB_Char_Tx[i] = NULL_CHAR;                    // USB Tx Buffer
  }
  usb_tx_ring_wr = BEGINNING;
  usb_tx_ring_rd = BEGINNING;

  // Configure UART 1
  UCA1CTLW0 = RESET_STATE;                         // Use word register
  UCA1CTLW0 |= UCSWRST;                            // Set Software reset enable
  UCA1CTLW0 |= UCSSEL__SMCLK;                      // Set SMCLK as fBRCLK

  UCA1BRW = BAUD_RATE_FOR_115200;                  // 9,600 Baud
  UCA1MCTLW = BAUD_CONCAT_FOR_115200;
  UCA1CTLW0 &= ~ UCSWRST;                          // Set Software reset enable
  UCA1TXBUF = NULL_CHAR;
  UCA1IE |= UCRXIE;                                // Enable RX interrupt
}
```

9.7 Serial_Interrupts.c

```
//=====================================================================
// Function name: eUSCI_A0_ISR
// Description: This is the ISR to transmit and receive for UCA0
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: USB_Char_Rx, usb_rx_ring_wr, UCA0_index, process_buffer_Rx
//
// Author: Lori Glenn
// Date: April 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//=====================================================================
#pragma vector=EUSCI_A0_VECTOR
__interrupt void eUSCI_A0_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA0IV,0x08)){
    case UCA_ISR_ONE:                           // Vector 0 - no interrupt
      break;
    case UCA_ISR_TWO:                           // Vector 2 - RXIFG
     // receiving = TRUE;
      temp = usb_rx_ring_wr++;
      USB_Char_Rx[temp] = UCA0RXBUF;            // RX -> USB_Char_Rx character
      if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
        usb_rx_ring_wr = BEGINNING;             // Circular buffer back to beginning
      }
      break;
    case UCA_ISR_THREE:                         // Vector 4 – TXIFG
      switch(UCA0_index++){
        case UCA_CHAR1:
        case UCA_CHAR2:
        case UCA_CHAR3:
        case UCA_CHAR4:
        case UCA_CHAR5:
        case UCA_CHAR6:
        case UCA_CHAR7:
        case UCA_CHAR8:
        case UCA_CHAR9:
          UCA0TXBUF = process_buffer_Rx[UCA0_index];
          break;
        case UCA_CHAR10:
          UCA0TXBUF = CR_CHAR;
          break;
        case UCA_CHAR11:
          UCA0TXBUF = LF_CHAR;
          break;
```

```
      default:
        UCA0IE &= ~UCTXIE;                          // Disable TX interrupt
        //UCA0_index++;
        break;
     }
     break;
   default: break;
  }
}

//=============================================================================
// Function name: eUSCI_A1_ISR
// Description: This is the ISR to transmit and receive for UCA1
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: USB_Char_Rx, usb_rx_ring_wr, UCA1_index, process_buffer_Rx
//
// Author: Lori Glenn
// Date: April 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//=============================================================================
#pragma vector=EUSCI_A1_VECTOR
__interrupt void eUSCI_A1_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA1IV,0x08)){
    case UCA_ISR_ONE:                               // Vector 0 - no interrupt
      break;
    case UCA_ISR_TWO:                               // Vector 2 - RXIFG
      temp = usb_rx_ring_wr++;
      USB_Char_Rx[temp] = UCA1RXBUF;                // RX -> USB_Char_Rx character
      if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
        usb_rx_ring_wr = BEGINNING;                 // Circular buffer back to beginning
      }
      break;
    case UCA_ISR_THREE:
      switch(UCA1_index++){
        case UCA_CHAR1:
        case UCA_CHAR2:
        case UCA_CHAR3:
        case UCA_CHAR4:
        case UCA_CHAR5:
        case UCA_CHAR6:
        case UCA_CHAR7:
        case UCA_CHAR8:
        case UCA_CHAR9:
           UCA1TXBUF = process_buffer_Rx[UCA1_index];
```

```
        break;
      case UCA_CHAR10:
        UCA1TXBUF = CR_CHAR;
        break;
      case UCA_CHAR11:
        UCA1TXBUF = LF_CHAR;
        break;
      default:
        UCA1IE &= ~UCTXIE;                          // Disable TX interrupt
        break;
    }
    break;
  default: break;
 }
}
```

9.8 Switch.c

```
// -------------------------------------------------------------------------
//
//  Description: This file contains instructions for managing the switches
//
//  William Moore
//  Spring 2021
//  Built with IAR Embedded Workbench Version: V7.20.1
// -------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include "macros.h"
#include <string.h>

// Function Prototypes
void low_baud_rate(void);
void high_baud_rate(void);

// Variable Declarations
unsigned int sw1_position;
unsigned int sw2_position;
unsigned int count_debounce_SW1;
unsigned int count_debounce_SW2;
unsigned int sw1DB;
unsigned int sw2DB;
extern volatile unsigned int mcc_timer;
volatile unsigned int SW1_PRESS;
volatile unsigned int SW2_PRESS;
volatile unsigned int SW1_event;
volatile unsigned int SW2_event;
unsigned int EMITTER_STATE;
extern volatile unsigned int display_timer;
extern volatile unsigned int time_display;
extern char test_command0[LCD_WIDTH];
extern unsigned int baud_determination;
extern volatile unsigned char display_changed;
extern char display_line[DISPLAY_HEIGHT][DISPLAY_WIDTH];
extern unsigned int SW2_BAUD;
extern char IOT2PC_DISPLAY[LCD_WIDTH];
extern char line1[LCD_WIDTH];
extern char line2[DISPLAY_WIDTH];
extern char line3[DISPLAY_WIDTH];
extern unsigned int connect_command;
extern unsigned int SW1_CALIBRATE;


// Code Section
```

```
#pragma vector = PORT4_VECTOR
__interrupt void switchP4_interrupt(void){
// Switch 1
  if(P4IFG & SW1){
    P4IFG &= ~SW1;                          // IFG SW1 cleared
    sw1_position = PRESSED;                 // switch has been pressed
    sw1DB = NOT_OKAY;                       // switch is being debounced
    count_debounce_SW1 = DEBOUNCE_RESTART;      // reset count required of debounce
    P4IE &= ~SW1;                           // interrupt disabled
    // enable CCR1
    TB0CCR1 = TB0CCR1_INTERVAL;     // CCR1
    TB0CCTL1 |= CCIE;                       // CCR1 enable interrupt

// do what you want with button press
    SW1_PRESS = TRUE;
    SW1_event = TRUE;
    P2OUT |= IR_LED;
    EMITTER_STATE = TRUE;
    display_timer = FALSE;
    time_display = FALSE;
    SW1_CALIBRATE++;

    UCA0IE |= UCTXIE;                       // Enable TX interrupt
  }
}


#pragma vector = PORT2_VECTOR
__interrupt void switchP2_interrupt(void){
// Switch 2
  if(P2IFG & SW2){
    P2IFG &= ~SW2;                          // IFG SW2 cleared
    sw2_position = PRESSED;                 // switch has been pressed
    sw2DB = NOT_OKAY;                       // switch is being debounced
    count_debounce_SW2 = DEBOUNCE_RESTART;      // reset count required of debounce
    P2IE &= ~SW2;                           // interrupt disabled
    // enable CCR1
    TB0CCR1 = TB0CCR1_INTERVAL;     // CCR1
    TB0CCTL1 |= CCIE;                       // CCR1 enable interrupt

// do what you want with button press
    SW2_PRESS++;
    // Connect to network, send these commands to UCA0TXBUF
    // AT+NSTAT 0x0d
    // AT+WSYNCINTRL=32000 0x0d
    // AT+PING=www.google.com 0x0d
    // AT+NSTCP=10899,1 0x0d
```

```
    connect_command++;
  }
}
```

9.9 Commands.c

```
//Commands function deciphers the commands sent to the FRAM
void commands(void){
 if(received_command[ZERO_CHAR] == '1' && received_command[ONE_CHAR] == '2' &&
received_command[TWO_CHAR] == '9' && received_command[THREE_CHAR] == '9' &&
new_command == TRUE && command_done == TRUE){
    if(received_command[FOUR_CHAR] == 'F'){
     //Car should go forward
     if(received_command[FIVE_CHAR] == '1'){
       command = GO_FORWARD1;
       command_done = FALSE;
     }
     if(received_command[FIVE_CHAR] == '5'){
       command = GO_FORWARD5;
       command_done = FALSE;
     }
     if(received_command[FIVE_CHAR] == '9'){
       command = GO_FORWARD9;
       command_done = FALSE;
     }
    }
    if(received_command[FOUR_CHAR] == 'B'){
     //Car should go backward
     if(received_command[FIVE_CHAR] == '1'){
       command = GO_BACKWARD1;
       command_done = FALSE;
     }
     if(received_command[FIVE_CHAR] == '3'){
       command = GO_BACKWARD3;
       command_done = FALSE;
     }
    }
    if(received_command[FOUR_CHAR] == 'R'){
     //Car should turn right
     if(received_command[FIVE_CHAR] == '6'){
       command = GO_RIGHT6;
       command_done = FALSE;
     }
     if(received_command[FIVE_CHAR] == '1'){
       command = GO_RIGHT1;
       command_done = FALSE;
     }
     if(received_command[FIVE_CHAR] == '2'){
       command = GO_RIGHT2;
       command_done = FALSE;
     }
    }
```

```
    if(received_command[FOUR_CHAR] == 'L'){
      //Car should turn left
      if(received_command[FIVE_CHAR] == '6'){
        command = GO_LEFT6;
        command_done = FALSE;
      }
      if(received_command[FIVE_CHAR] == '1'){
        command = GO_LEFT1;
        command_done = FALSE;
      }
      if(received_command[FIVE_CHAR] == '2'){
        command = GO_LEFT2;
        command_done = FALSE;
      }
    }
    if(received_command[FOUR_CHAR] == 'A'){
      num_location[9] = num_location[9] + TRUE;
      strcpy(display_line[LINE0], num_location);
      update_string(display_line[LINE0], LINE0);
      display_changed = TRUE;
      new_command = FALSE;
    }
    if(received_command[FOUR_CHAR] == '7'){
      num_location[9] = '7';
      strcpy(display_line[LINE0], num_location);
      update_string(display_line[LINE0], LINE0);
      display_changed = TRUE;
      new_command = FALSE;
    }
    if(received_command[FOUR_CHAR] == 'T'){
      //time_counter = FALSE;
      time_started = TRUE;
    }
    if(received_command[FOUR_CHAR] == 'S'){
      time_started = FALSE;
      new_command = FALSE;
    }
    if(received_command[FOUR_CHAR] == 'E'){
      time_counter = FALSE;
      time_counted = FALSE;
      time_count[5] = '0';
      time_count[6] = '0';
      time_count[7] = '0';
      time_count[9] = '0';
      strcpy(display_line[LINE3], time_count);
      update_string(display_line[LINE3], LINE3);
      display_changed = TRUE;
      new_command = FALSE;
```

```
    }
    if(received_command[FOUR_CHAR] == 'G'){
      go_autonomous = TRUE;
      drive_timer = FALSE;
      new_command = FALSE;
    }
    if(received_command[FOUR_CHAR] == 'D'){
      new_command = FALSE;
      drive_timer = FALSE;
      go_autonomous++;
    }
  }
  //Forward for 1 second
  if(command == GO_FORWARD1 && new_command == TRUE){
    if(forward_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      forward();
      forward_init = TRUE;
    }
    if(sec_timer >= TRUE){
      stop();
      forward_init = FALSE;
      if(received_command[FOUR_CHAR] == 'F'){
        new_command = FALSE;
      }
      command = FALSE;
      command_done = TRUE;
    }
  }
  //Forward for 5 seconds
  if(command == GO_FORWARD5 && new_command == TRUE){
    if(forward_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      forward();
      forward_init = TRUE;
    }
    if(sec_timer >= FIVE_CHAR){
      stop();
      forward_init = FALSE;
      if(received_command[FOUR_CHAR] == 'F'){
        new_command = FALSE;
      }
      command = FALSE;
      command_done = TRUE;
    }
  }
```

```
  //Forward for 9 seconds
  if(command == GO_FORWARD9 && new_command == TRUE){
    if(forward_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      forward();
      forward_init = TRUE;
    }
    if(sec_timer >= NINE_CHAR){
      stop();
      forward_init = FALSE;
      if(received_command[FOUR_CHAR] == 'F'){
        new_command = FALSE;
      }
      command = FALSE;
      command_done = TRUE;
    }
  }
  //Reverse for 1 second
  if(command == GO_BACKWARD1 && new_command == TRUE){
    if(reverse_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      reverse();
      reverse_init = TRUE;
    }
    if(sec_timer >= TRUE){
      stop();
      reverse_init = FALSE;
      if(received_command[FOUR_CHAR] == 'B'){
        new_command = FALSE;
      }
      command = FALSE;
      command_done = TRUE;
    }
  }
  //Reverse for 3 second
  if(command == GO_BACKWARD3 && new_command == TRUE){
    if(reverse_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      reverse();
      reverse_init = TRUE;
    }
    if(sec_timer >= THREE_CHAR){
      stop();
      reverse_init = FALSE;
      if(received_command[FOUR_CHAR] == 'B'){
```

```
      new_command = FALSE;
     }
     command = FALSE;
     command_done = TRUE;
   }
  }
  //Turn right for 0.6 seconds
  if(command == GO_RIGHT6 && new_command == TRUE){
   if(right_init == FALSE){
     sec_timer = FALSE;
     ms200_count = FALSE;
     turn_right();
     right_init = TRUE;
   }
   if(ms200_count >= THREE_CHAR){
     stop();
     right_init = FALSE;
     if(received_command[FOUR_CHAR] == 'R'){
       new_command = FALSE;
     }
     command = FALSE;
     command_done = TRUE;
   }
  }
  //Turn right for 1 second
  if(command == GO_RIGHT1 && new_command == TRUE){
   if(right_init == FALSE){
     sec_timer = FALSE;
     ms200_count = FALSE;
     turn_right();
     right_init = TRUE;
   }
   if(sec_timer >= TRUE){
     stop();
     right_init = FALSE;
     if(received_command[FOUR_CHAR] == 'R'){
       new_command = FALSE;
     }
     command = FALSE;
     command_done = TRUE;
   }
  }
  //Turn right for 2 seconds
  if(command == GO_RIGHT2 && new_command == TRUE){
   if(right_init == FALSE){
     sec_timer = FALSE;
     ms200_count = FALSE;
     turn_right();
```

```
    right_init = TRUE;
   }
   if(sec_timer >= TWO_CHAR){
    stop();
    right_init = FALSE;
    if(received_command[FOUR_CHAR] == 'R'){
     new_command = FALSE;
    }
    command = FALSE;
    command_done = TRUE;
   }
  }
  //Turn left 0.6 seconds
  if(command == GO_LEFT6 && new_command == TRUE){
   if(left_init == FALSE){
    sec_timer = FALSE;
    ms200_count = FALSE;
    turn_left();
    left_init = TRUE;
   }
   if(ms200_count >= THREE_CHAR){
    stop();
    left_init = FALSE;
    if(received_command[FOUR_CHAR] == 'L'){
     new_command = FALSE;
    }
    command = FALSE;
    command_done = TRUE;
   }
  }
  //Turn left 1 second
  if(command == GO_LEFT1 && new_command == TRUE){
   if(left_init == FALSE){
    sec_timer = FALSE;
    ms200_count = FALSE;
    turn_left();
    left_init = TRUE;
   }
   if(sec_timer >= TRUE){
    stop();
    left_init = FALSE;
    if(received_command[FOUR_CHAR] == 'L'){
     new_command = FALSE;
    }
    command = FALSE;
    command_done = TRUE;
   }
  }
```

```
  //Turn left 2 seconds
  if(command == GO_LEFT2 && new_command == TRUE){
    if(left_init == FALSE){
      sec_timer = FALSE;
      ms200_count = FALSE;
      turn_left();
      left_init = TRUE;
    }
    if(sec_timer >= TWO_CHAR){
      stop();
      left_init = FALSE;
      if(received_command[FOUR_CHAR] == 'L'){
        new_command = FALSE;
      }
      command = FALSE;
      command_done = TRUE;
    }
  }
}
```

9.10 Follow_The_Line.c

```
//=====================================================================
// Function name: follow_the_line
//
// Description: This function contains the algorithm to follow the black line
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
//  Globals: line_state, on_black_line_left, on_black_line_right, left_speed,
//           right_speed
//
// Author: Lori Glenn
// Date: March 2021
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (7.20.1)
//=====================================================================
void follow_the_line(void){
 if(previous_movement_time_loops != movement_time_loops){
  line_time_loops++;
  previous_movement_time_loops = movement_time_loops;
 }

 switch(line_state){
 case BOTH_ON:
    RIGHT_REVERSE_SPEED = WHEEL_OFF;
    LEFT_REVERSE_SPEED = WHEEL_OFF;

    P1OUT &= ~RED_LED;                    // NOTE: RIGHT WHEEL IN CODE IS LEFT
    P6OUT &= ~GRN_LED;                    //       WHEEL ON CAR
    P1OUT |= RED_LED;
    P6OUT |= GRN_LED;

    if(on_black_line_right == FALSE){          // LEFT DETECT OFF LINE

      line_state = RIGHT_OFF;
      line_time_loops = RESET_STATE;
    }

    if(on_black_line_left == FALSE){           // RIGHT DETECT OFF LINE

      line_state = LEFT_OFF;
      line_time_loops = RESET_STATE;
    }

    RIGHT_FORWARD_SPEED = CRUISE_R;
    LEFT_FORWARD_SPEED = CRUISE_L+MED_ADJUSTMENT;
```

```
    break;
  case LEFT_OFF:                          // LEFT DETECT IS OFF - SWING RIGHT

    P1OUT &= ~RED_LED;
    P6OUT &= ~GRN_LED;
    P1OUT |= RED_LED;

    RIGHT_FORWARD_SPEED = CRUISE_R + MED_ADJUSTMENT;
    LEFT_FORWARD_SPEED = CRUISE_L;

    if(on_black_line_left == TRUE){
      line_state = RECOVERY_FROM_LEFT_OFF;
      line_time_loops = RESET_STATE;
    }

   break;
  case RIGHT_OFF:                         // RIGHT DETECT IS OFF - SWING LEFT

    P1OUT &= ~RED_LED;
    P6OUT &= ~GRN_LED;
    P6OUT |= GRN_LED;

    RIGHT_FORWARD_SPEED = CRUISE_R + SMALL_ADJUSTMENT;
    LEFT_FORWARD_SPEED = CRUISE_L + LARGE_ADJUSTMENT;

    if(on_black_line_right == TRUE){
      line_state = RECOVERY_FROM_RIGHT_OFF;
      line_time_loops = RESET_STATE;
    }

   break;
  case RECOVERY_FROM_RIGHT_OFF:
    P1OUT &= ~RED_LED;
    P6OUT &= ~GRN_LED;
    P1OUT |= RED_LED;

    RIGHT_FORWARD_SPEED = CRUISE_R + MED_ADJUSTMENT;
    LEFT_FORWARD_SPEED = CRUISE_L;

    if(line_time_loops == CORRECTION_TIME_LOOPS_RIGHT){
      line_time_loops = RESET_STATE;
      line_state = BOTH_ON;
    }

   break;
  case RECOVERY_FROM_LEFT_OFF:
    P1OUT &= ~RED_LED;
    P6OUT &= ~GRN_LED;
```

```
    P6OUT |= GRN_LED;

    RIGHT_FORWARD_SPEED = CRUISE_R - MED_ADJUSTMENT;
    LEFT_FORWARD_SPEED = CRUISE_L + MED_ADJUSTMENT;

    if(line_time_loops == CORRECTION_TIME_LOOPS_LEFT){
      line_time_loops = RESET_STATE;
      line_state = BOTH_ON;
    }

   break;
  default: break;
  }

}
```

9.11 LCD_Display_Functions.c

```
//-------------------------------------------------------------------------------------------------
//Description:  This file contains the functions and variables that control the
//              display easily
//
//
//By:           Nathan Carels
//Date:         2/6/2021
//Build:        Built with IAR Embedded Workbench Version: V7.20.1.997 (7.20.1)
//-------------------------------------------------------------------------------------------------


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

//clear the lcd
void clear_lcd(void){
  char display_clear[MAX_LCD_LENGTH] = "          ";
  for(int i=LINE1;i<NUM_LINES;i++)
    update_string(display_clear,i);
  Display_Update(CHAR_0,CHAR_0,CHAR_0,CHAR_0);
}

//displays the string in line to LCD line 1
void lcd_line1(char* line){
  update_string(line,LINE1);
  Display_Update(CHAR_0,CHAR_0,CHAR_0,CHAR_0);
}

//displays the string in line to LCD line 2
void lcd_line2(char* line){
  update_string(line,LINE2);
  Display_Update(CHAR_0,CHAR_0,CHAR_0,CHAR_0);
}

//displays the string in line to LCD line 3
void lcd_line3(char* line){
  update_string(line,LINE3);
  Display_Update(CHAR_0,CHAR_0,CHAR_0,CHAR_0);
}

//displays the string in line to LCD line 4
void lcd_line4(char* line){
  update_string(line,LINE4);
  Display_Update(CHAR_0,CHAR_0,CHAR_0,CHAR_0);
}
```

10 Conclusion

Lightening McGreen is now a fully functioning autonomous car. After 14 weeks, Lightening McGreen can make its way through an obstacle course of eight stations while receiving commands from a user over the Wi-Fi and utilizing the IOT module. It is also able to follow a black line and exit a circle upon two commands total. This experience has taught us real-world experience of how to start and complete a project. As individuals, we faced a range of obstacles, such as soldering the Display and FRAM together to hardware failure (especially in the DAC, IOT modules, and IR Sensors). However, we were able to overcome these issues by trying a range of solutions and persistence, as well as knowing when to ask for help. We learned how best to debug our cars by both evaluating its performance and by looking at the code; understanding it could be either hardware or software related. Throughout each of the ten projects, our team realized the importance of understanding what we were trying to code and utilized the knowledge and resources at our disposal. Overall, the project contained many hurdles we had to overcome, including an eight-station obstacle course, but from this process we were able to get a better understanding of embedded systems. Lightening McGreen has fueled our motivation and drive to be innovative thinkers in life and to remember life is a highway.