RACECARLSSON C52

Car with Autonomous Navigation and Control via Wi-Fi

Team 2

Team Members

Rachel Williams Steffon Brigman Christopher Woedy	Originator: R.Williams, S.Brigman & C.Woedy			
	Checked: Dec. 4th, 2015	Released: Dec. 4th, 2015		
	Filename: RACECARLSSONC52 -			
	Title: **RACECARLSSON C52** **Car with Autonomous Navigation and Control via Wi-Fi**			
This document contains information that is **PRIVILEGED** and **CONFIDENTIAL**; you are hereby notified that any dissemination of this information is strictly prohibited.	Date: **12/04/2015**	Document Number: **0-0002-001-0001-02**	Rev: **Z**	Sheet: **1 of 112**

Revision	Description
A	The creation and initial formatting of the document including headings and structure.
B	Focused on the installation of the product’s power supply detailing components and process.
C	Formatting, grammar, and structure corrections to overall document.
D	Focused on details about the LCD Screen installation and configuration.
E	Detail corrections from revision D, focused on document formatting and products specifications.
F	Added details regarding the installation and implementation of the partial H-Bridge.
G	Formatting, grammar, and structure corrections to overall document.
H	Added information on the installation and configuration of the full H-Bridge.
I	Detail corrections from revision H, addressed footer formatting and revision descriptions.
J	Added sections on the ADC and the Thumb Wheel.
K	Formatting, grammar, and structure corrections to overall document.
L	Added details on the Emitter and the Detectors.
M	Formatting, grammar, and structure corrections to overall document.
N	Added Flowcharts for Main, Ports, Timers, ADC, and Interrupts source code files.
O	Formatting, grammar, and structure corrections to added flowchart section.
P	Added software descriptions for Main, Ports, Timers, ADC, and Interrupts source code files.
Q	Formatting, grammar, and structure corrections to the added software section.
R	Modified document to add details on serial communications.
S	Formatting, grammar, and structure corrections to overall document.
T	Added details on WiFi Module
U	Added details on Wireless control
V	Formatting, grammar, and structure corrections to overall document.
W	Added Power Analysis section
X	Formatting, grammar, and structure corrections to overall document.
Y	Added Conclusion
Z	Formatting, grammar, and structure corrections to overall document.

Table of Contents

Table of Figures

1. Scope

This document describes the overall process used to assemble our vehicle and the programming implemented to have the system controlled via Wi-Fi, or drive based on black line detection. The process began with the installation of the Buck-Boost converter circuit that is used to deliver power from the Control Board to the system. This circuit was designed and constructed to allow a separate battery pack power system on the vehicle. The Buck-Boost converter utilizes voltage from the battery pack and converts it to the appropriate system voltage, allowing for mobility of the device. Furthermore, the MSP430FR5739 FRAM Experimenter Board was modified to interconnect with the control board allowing for added functionality. An LCD Screen is mounted to the Control Board to provide feedback to the user while operating the device. This display allows for a TLI to guide users through menus and functions. N-FETs were installed onto the Control Board to allow for unidirectional forward motor control of the left and right motors on the vehicle. Afterwards, N-FETs and P-FETs were installed onto the Control Board to allow for both forward and reverse motor control for the left and right motors on the vehicle. The thumb wheel was added for analog signal testing and user input. An IR emitter and two detectors were connected to ADC channels so that the vehicle could detect IR radiation bouncing off the surface below the vehicle against the ambient IR lighting. The wireless module was added to allow for the vehicle's motors to be controlled by a client program over the wireless network.

2. Abbreviations

Abbreviations	Definitions
ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
CMOS	Complementary metal-oxide semiconductor
CPU	Central Processing Unit
DC	Direct Current
FRAM	Ferroelectric Random Access Memory
GPI/O	General Purpose Input / Output
IOT	Internet of Things
IR	Infrared
ISR	Interrupt Service Handler
LCD	Liquid Crystal Display
LED	Light Emitting Diode
N-FET	N-Type Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)
P-FET	P-Type Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)
PSU	Power Supply Unit
RX	Receive
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TLI	Text Line Interface
TX	Transmit
UART	Universal asynchronous receiver/transmitter
UDP	User Datagram Protocol
Wi-Fi	Wireless Fidelity

3. Overview

The purpose of this vehicle is to support both autonomous operation and direct control via IOT wireless device. These operational states are achieved by utilizing a Texas Instrument MSP430 Experimenter board, custom design ASIC Control board, several peripheral devices, and device-specific programming to drive hardware. The major components of the vehicle may be viewed in the block diagram below.

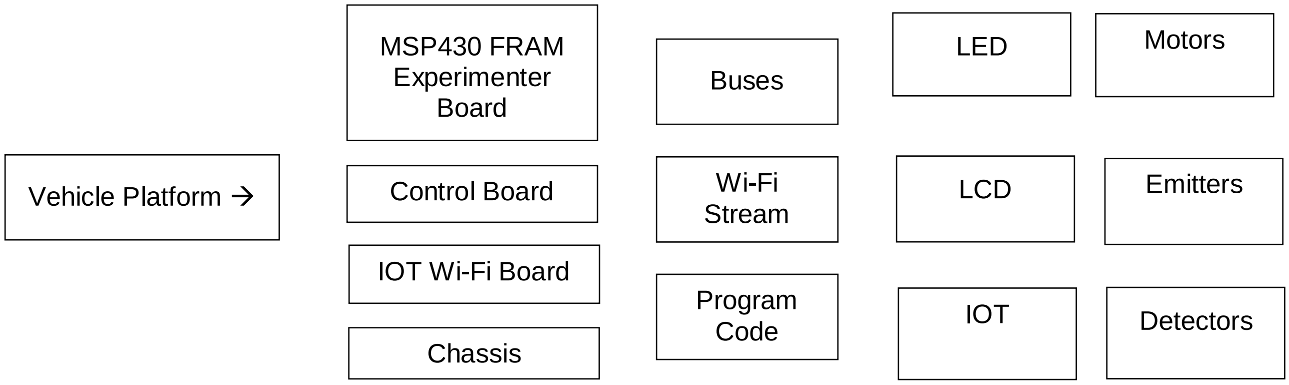


Figure 1 Overview Block diagram

3.1. Power Supply Unit Block Diagram

The AA battery pack is used to supply unregulated voltage to the Buck-Boost converter. The Buck-Boost converter then regulates and filters the voltage to a usable level for the MSP430 and Control boards to distribute to the various on-board systems; i.e. LEDs, the LCD, the IOT device, motors, emitters, and detectors.

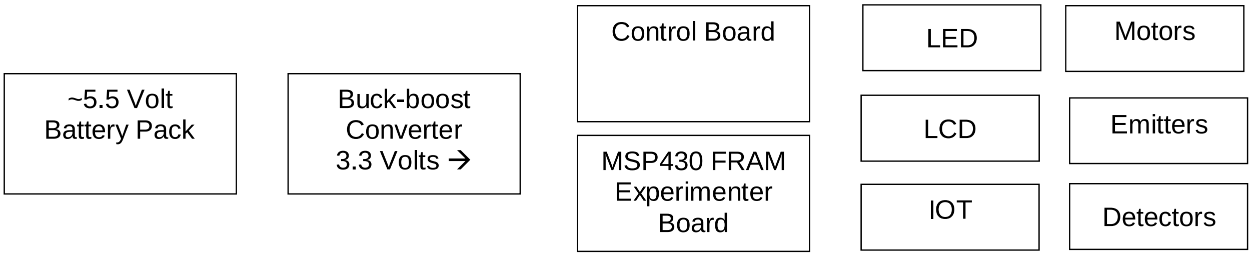


Figure 2 PSU Block Diagram

3.2. LED and LCD Screen Block

Information is sent from the MSP430 board to the Control board, and then to the LCD screen via the SPI bus. This data is interpreted and displayed by the LCD Screen. The LCD Screen is used to indicate instructions or useful information to the user. The LEDs share port connection with the drive signals as well as IOT communication signals. These LED outputs allowed for easier problem debugging in a runtime environment. The LEDs are driven by Port J and Port 3 displayed respectively below:

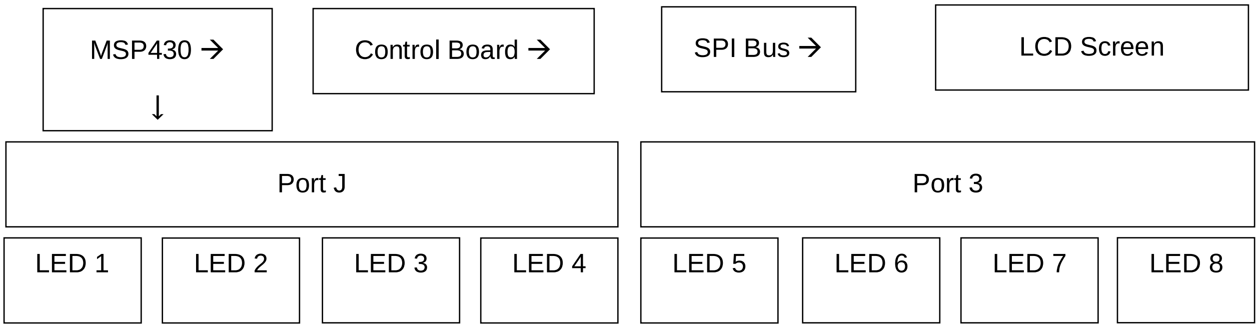


Figure 3 LCD Block Diagram

3.3. Forward Motor Control

Left and right motors are powered by the battery pack supply which requires the battery system switch to be enabled and a physical connection to the battery pack. Power travels through the partial H-Bridge and then to the connectors between the Control Board and MSP430 board. The MSP430 Experimenter board enables the motors via on-board pins. These pins are shared by LEDs that correspond to left motor forward and the right motor forward. Left and right operations are represented by LED 7 and LED 5 respectively. These LEDs and motors are driven by Port 3, as seen in the diagram below:

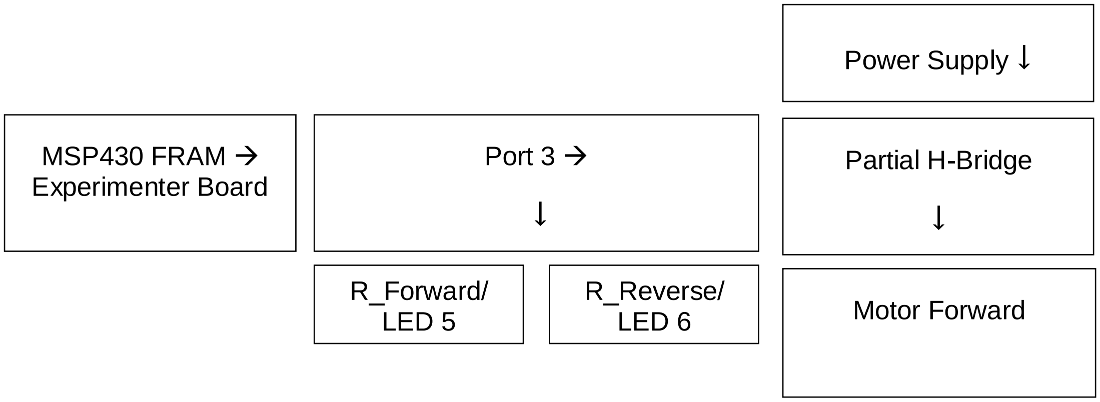


Figure 4 Forward Motor Control Block Diagram

3.4. Complete Motor Control (Forward/Reverse)

For complete motor control of the left and right motors, the system requires a physical connection to the battery pack and for the battery system switch to be enabled. Power travels through the full H-Bridge and then to the connectors between the Control Board and MSP430 board. The MSP430 Experimenter board enables the motors via on-board pins. These pins are shared by LEDs that correspond to the left motor and the right motor. Left motor forward, left motor reverse, right motor forward, and right motor reverse are represented by LED 7, LED 8, LED 5, and LED 6 respectively. These LEDs are driven by Port 3, as seen in the diagram below:

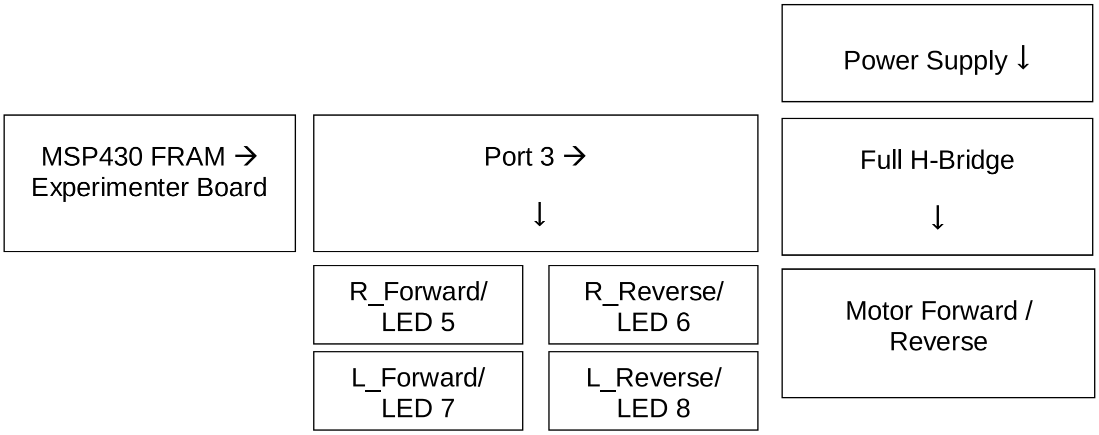


Figure 5 Forward/Reverse Motor Control Block Diagram

3.5. Detection Unit Block Diagram

Complete Detection Unit requires an emitter and two detectors working with the ADC. The emitter outputs an infrared signal that is partially reflected and partially absorbed by the surface below. The left and right detector components react to the amount of IR radiation reflected back from the emitter. The ADC processes this information to a system register making it a usable value by the implemented system. This allows detection of different surface types.

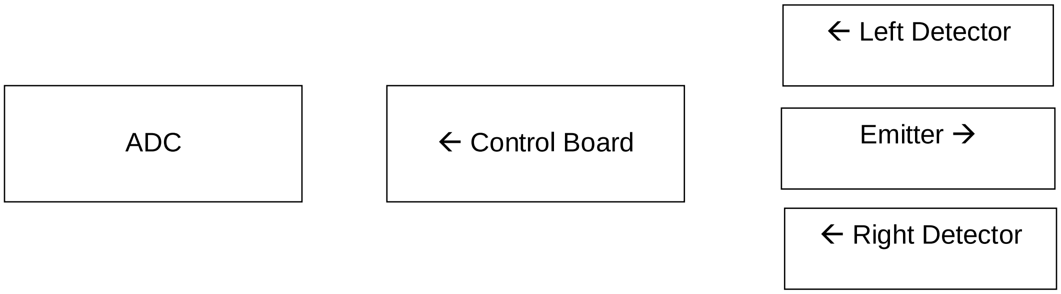


Figure 6 ADC Block Diagram

3.6. Serial Communication Unit Block Diagram

The serial communication unit utilizes UART standards to establish a link between other UART capable devices. This allows the vehicle to communicate between different vehicles as well as other serial devices. The MSP430 FRAM Experimenter board's TX and RX pins were fed to the control board through Port 2. The TX and RX pins were then made accessible through header pins on the control board. Serial Communication also allows the vehicle to interface with the wireless module as well as through USB serial streams.

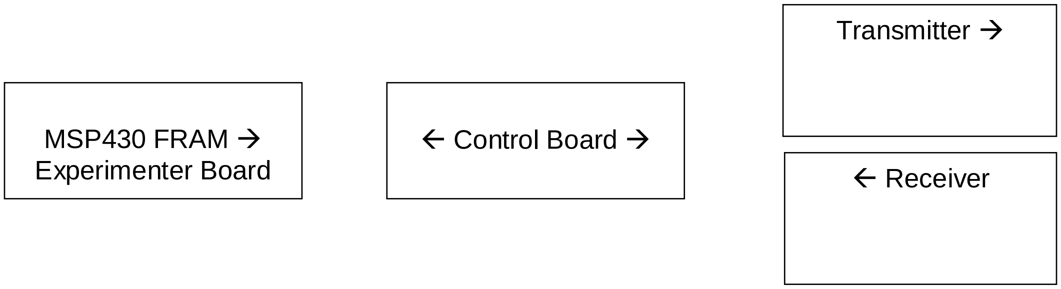


Figure 7 Serial Communication Unit Block Diagram

3.7. Wireless Module Unit Block Diagram

The wireless module utilized the MSP430 board's integrated process serial communication pins to communicate data with the vehicle. This bi-directional line allowed the vehicle to control the wireless module and the wireless module to send data back to the vehicle for processing. This module was essential in allowing wireless interface and control of the vehicle. Connection to an external access point allowed for various and different input methods to the vehicle, i.e. different computing devices or controller implementations.

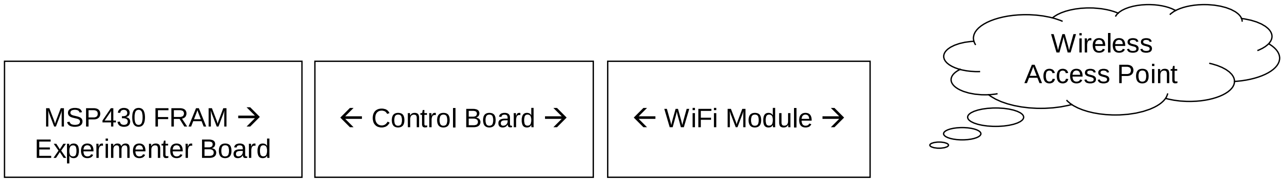


Figure 8 Serial Communication Unit Block Diagram

4. Hardware

This section covers a brief technical description of each hardware component and analyzes how the hardware component is utilized in the overall design of the vehicle. Schematics and diagrams are included for each section.

4.1. Power Supply Unit

The PSU utilizes one inductor, several resistors, capacitors, other electronic components, and a LT3532 Buck-Boost converter in order to supply constant DC voltage value of 3.3 V to the FRAM Experimenter Board and the Control Board. This PSU circuit allows the board to be powered by a battery pack, allowing for mobility and sufficient power draw to run the system's motors. A switch component has been mounted and connected to the board that opens the battery circuit. This switch prevents power drain while the device is not in use. In the Power Supply Unit's schematic shown in the following figure, the Buck-Boost converter chip is represented by the component U1. Input voltage is supplied from the battery pack via connector J1 and enabled by SW1. The diode, D2, is used to prevent back currents into the battery and provides a sufficient voltage drop to meet the voltage tolerance of the Buck-Boost converter. The converted voltage is driven at VOUT. Both input and output voltage signals are cleaned up by a network of bulk, distributor, and coupling capacitors. The output voltage is then supplied to the rest of the system.

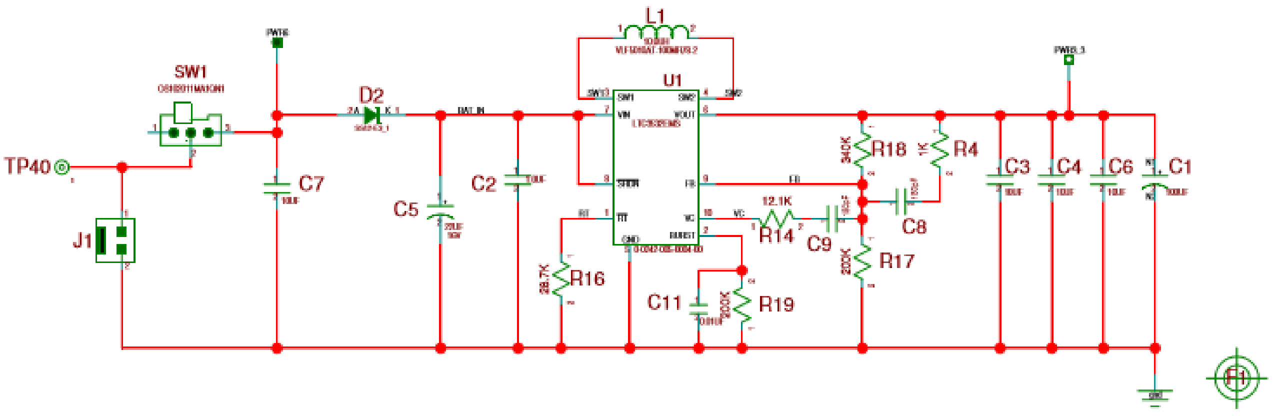


Figure 9 PSU Schematic

4.2. LCD Screen

The LCD Screen display used for this device is an EADOGS104-A. This component allows for output from the board to provide character feedback to the user. The EADOGS104-A features high-contrast functionality and 4x10 character output where any 2 rows can be merged to output larger character strings. This device is controlled by a SSD1803A CMOS LCD driver on the SPI bus and receives data from the MSP430 on the Texas Instrument FRAM Experimenter Board. The LCD Screen is illuminated by an EA LED36X28-A amber LED backlight that is mounted behind the LCD display. The backlight is enabled by the Q51 NFET transistor driven by the LCD_BACKLITE signal. Resistors are used to prevent excess current flow that may damage sensitive components within the device. Capacitors are used to clean up the input and output signals. The LCD Screen is connected to the SPI bus in order to receive instructions and configuration data from the Experimenter Board. The bus can be located on pins 15 through 18 of the device as seen in the diagrams below:

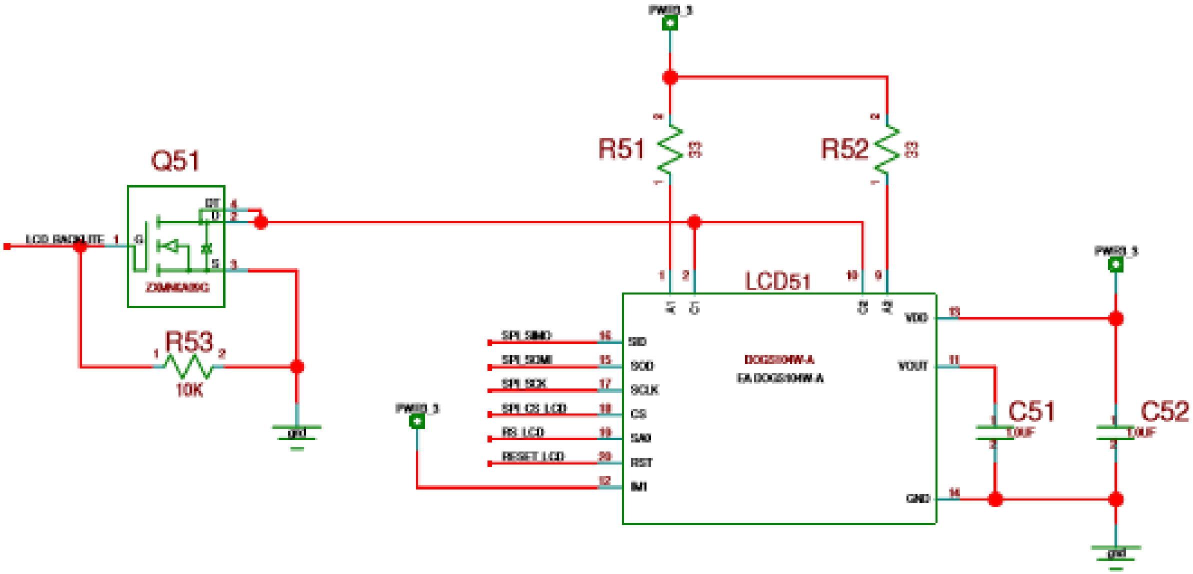


Figure 10 LCD Screen Circuit Schematic

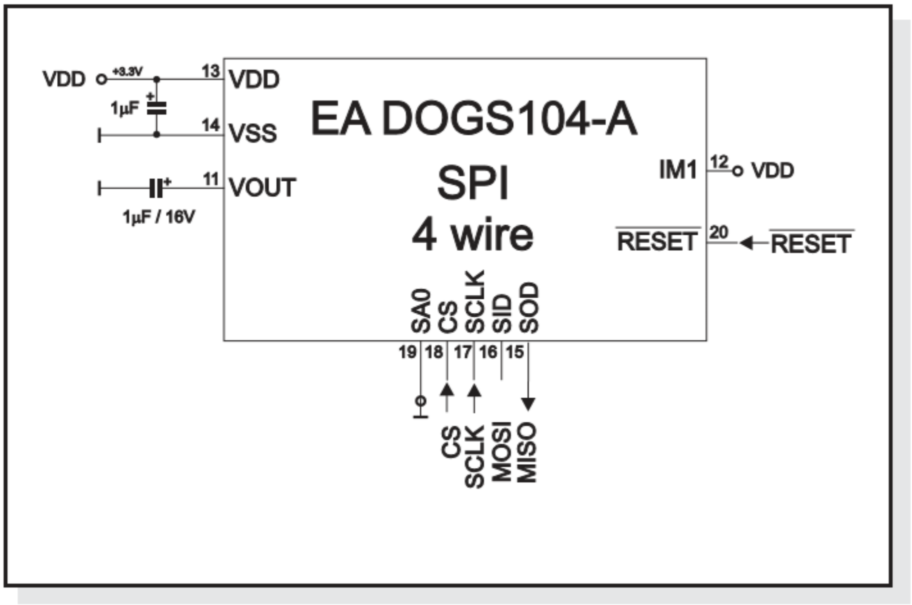


Figure 11 LCD Screen Specification Schematic

4.3. Forward Motor Control (Partial H-Bridge)

Forward motor control was implemented by partial installation of the H-Bridge circuit. The H-bridge is designed to allow the board to trigger forward and reverse operation of the motors safely without damaging the motors or board. Prior to proceeding with the entire H-Bridge, it was essential to first install, test, and verify the partial H-Bridge and the vehicle's forward drive functionality to ensure appropriate mobility and ultimately prevent damage to the board or motors. The successful implementation of the Partial H-Bridge was made possible by applying a short circuit to the jumper pins titled JMP-L and JMP-R. Without these jumpers the forward functionality would not have been available until full installation. Two N-FETs were installed to allow the FRAM Experimenter board to trigger battery supply voltage to the motors safely. The signal provided from the board creates a closed circuit path to ground across the motor's terminals. One terminal provided full battery voltage and the other a path to ground. A schematic of the left-side partial H-Bridge can be seen in the figure below:

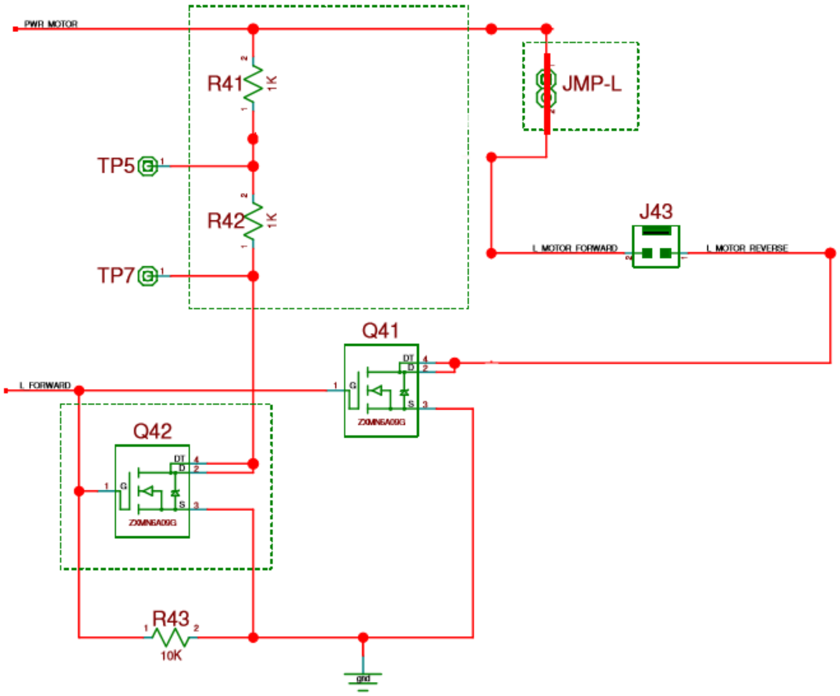


Figure 12 Partial H-Bridge Schematic

4.4. Forward/Reverse Motor Control (Full H-Bridge)

Forward and reverse motor control were implemented by full installation of the H-Bridge circuit. This installation included installing several N-FET and P-FET transistors to the circuit. Before hardware was tested, both JMP-L and JMP-R had to be open-circuited. The full H-Bridge simply expanded on the functionality of the partial H-Bridge to allow battery voltage to power both forward drive as well as reverse drive operations of the motors. N-FETs and P-FETs allowed the FRAM Experimenter board to safely supply battery voltage to the motors. The signal provided from the Experimenter board creates a closed circuit path to ground across the motors terminals. One terminal provided full battery voltage and the other a path to ground. These polarities flipped when triggering the opposite drive-direction signal. It is important to note that triggering both sides of the H-Bridge at once will cause an overload at one of the P-FETs, which may result in damage to the board or vehicle.

DO NOT TRIGGER BOTH FORWARD AND REVERSE CIRCUITS AT THE SAME TIME

The left motor and right motor H-Bridge circuit schematics may be seen below in Figures 10 and 11, respectively:

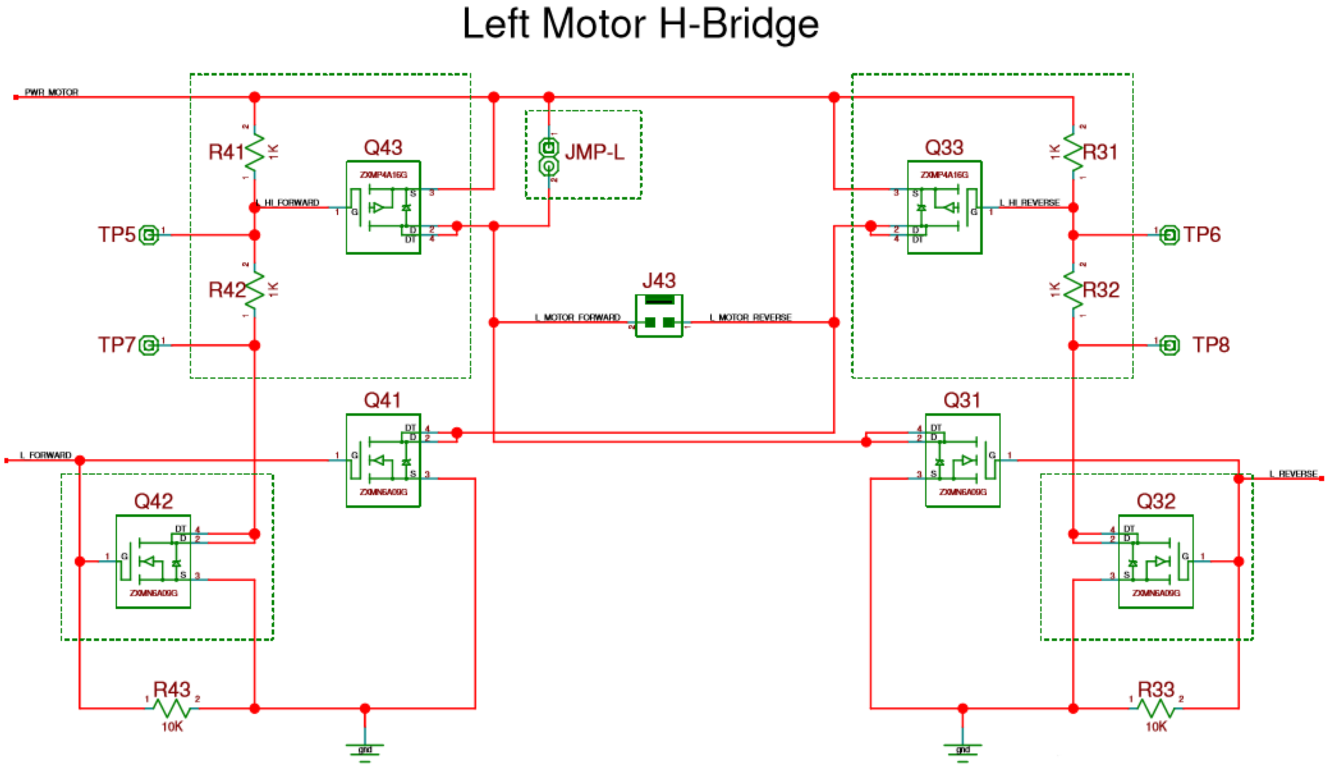


Figure 13 Left Motor H-Bridge Schematic

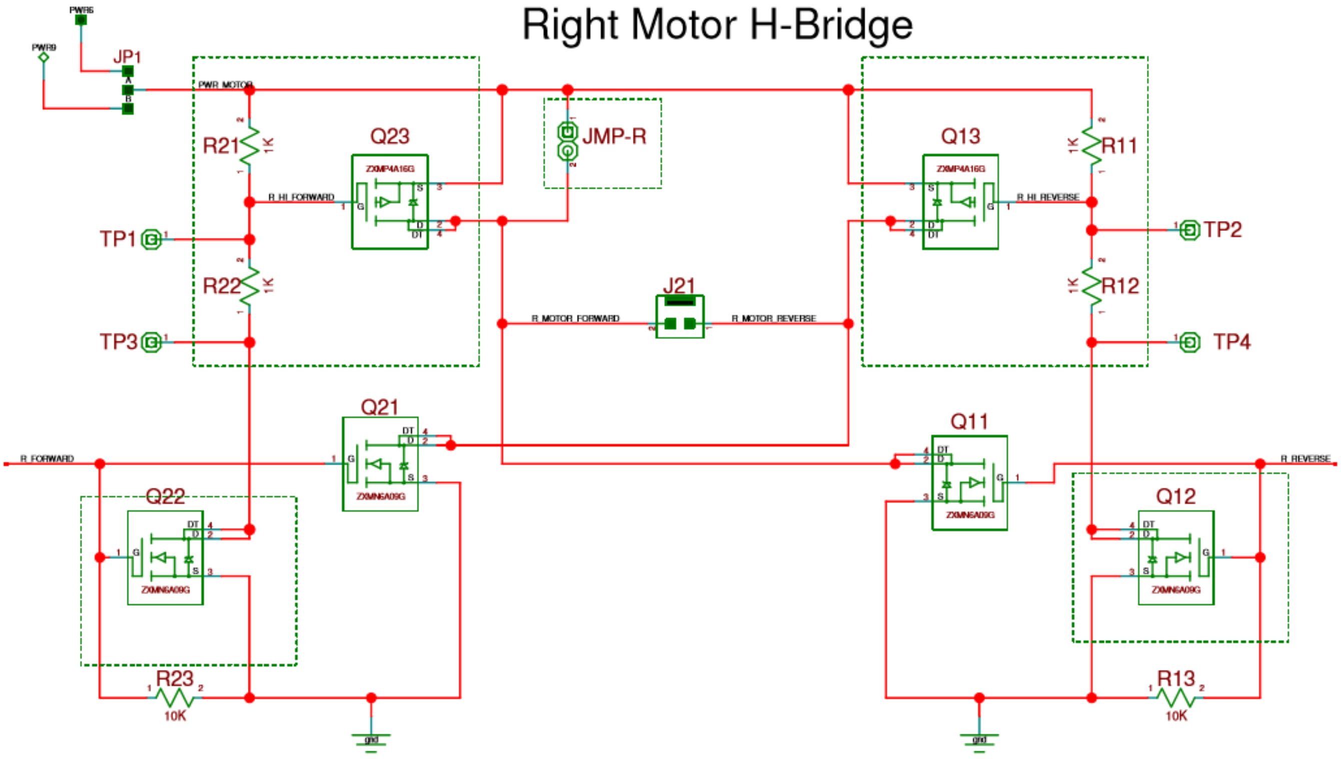


Figure 14 Right Motor H-Bridge Schematic

4.5. Detection Unit

The Detection Unit of the vehicle uses an infrared light emitting diode as an emitter component. This IR emitter outputs a constant IR light signal, which reacts accordingly to the surface below the vehicle. Depending on the reaction of the surface below the vehicle, the IR light is absorbed, refracted, or reflected. If the IR light is reflected then it may be detected by either/both of the two detector nodes. If the IR light is absorbed or refracted, then the detectors will detect a lower IR signal or none at all. These three components make the vehicle's ability to follow a black line possible because white surfaces reflect a lot of IR radiation while black surfaces absorb them. These differences in physical properties allow different readings to be detected between black and white. Below is the schematic for the emitter unit. The emitter draws power from the PWR3_3 pin allowing for a higher current sink and brighter IR light source. Drawing power from this pin improves detection capabilities. The circuit is enabled by an N-FET allowing the FRAM Experimenter Board to trigger the circuit with a GPI/O pin. Resistors are used to prevent excess current flow and prevent damage to electrical components.

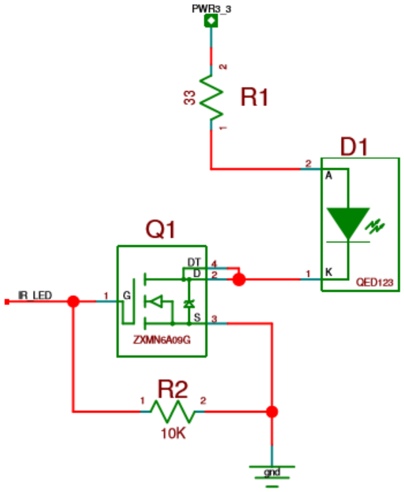


Figure 15 Emitter Schematic

Below are the schematics for the detector units. The detectors also draw power from the PWR3_3 pin allowing for an amplified detection signal. Drawing power from this pin improves detection capabilities. The circuit utilizes a resistor and capacitor to form a low pass filter. The low pass filter is used to help reduce noise in the analog signal before passing it to the on-board ADC. The ADC then converts the signal into a 10-bit resolution value that is used for detection logic in the vehicles programming. Resistors are used to prevent excess current flow and prevent damage to electrical components.

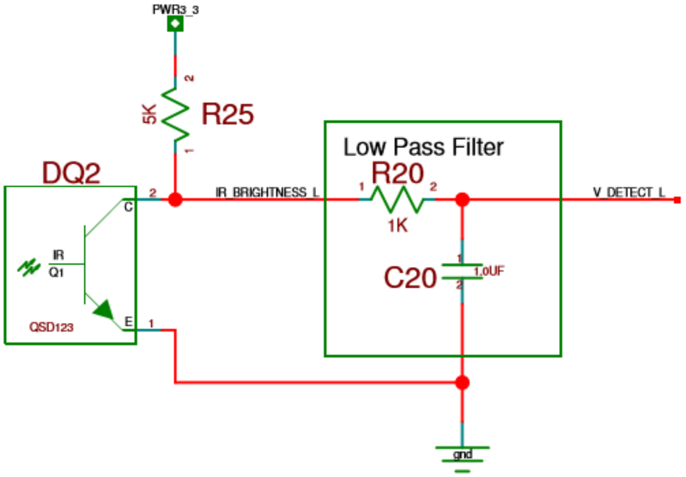


Figure 16 Left Detector Schematic

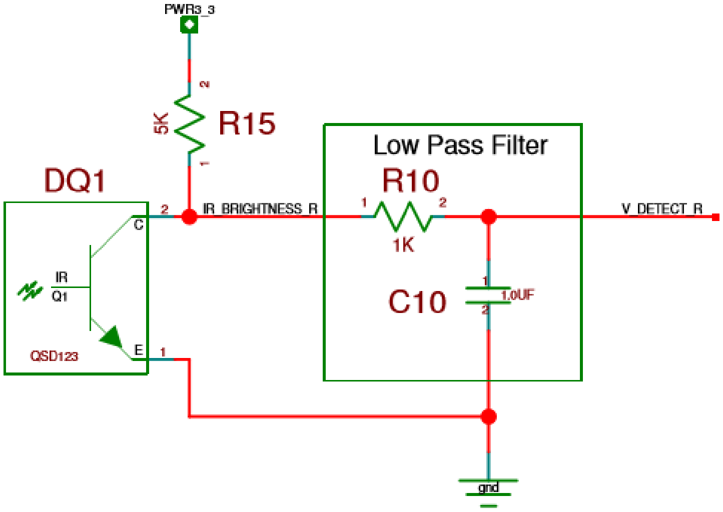


Figure 17 Right Detector Schematic

4.6. Serial Communication Unit

The serial communication unit was installed by soldering a 6-pin header to the available port pins on J8. This allowed for transmission and reception functionality on the control board, which is shown in the figure below. The serial communication unit utilizes the MSP430 FRAM Experimenter board's integrated UART protocol and modules. Utilizing this hardware allowed for quick and effective use of serial communication locally on the vehicle. Using a shorting plug to short pins 1 and 2, as well as pins 5 and 6 enabled an internal loop to send and receive data to and from the FRAM Experimenter Board. The shorting plug placement for pins 1 and 2 is shown by the black box in Figure 16. Open pins were used to during testing to mount the oscilloscope for baud rate verification and data packet verification. Verification of these aspects was essential before communicating, to make sure everything was operating as it was expected to.

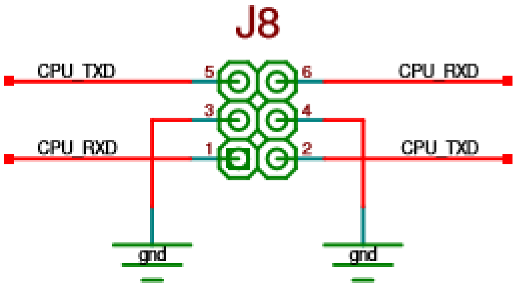


Figure 18 Serial Communication Pin Header

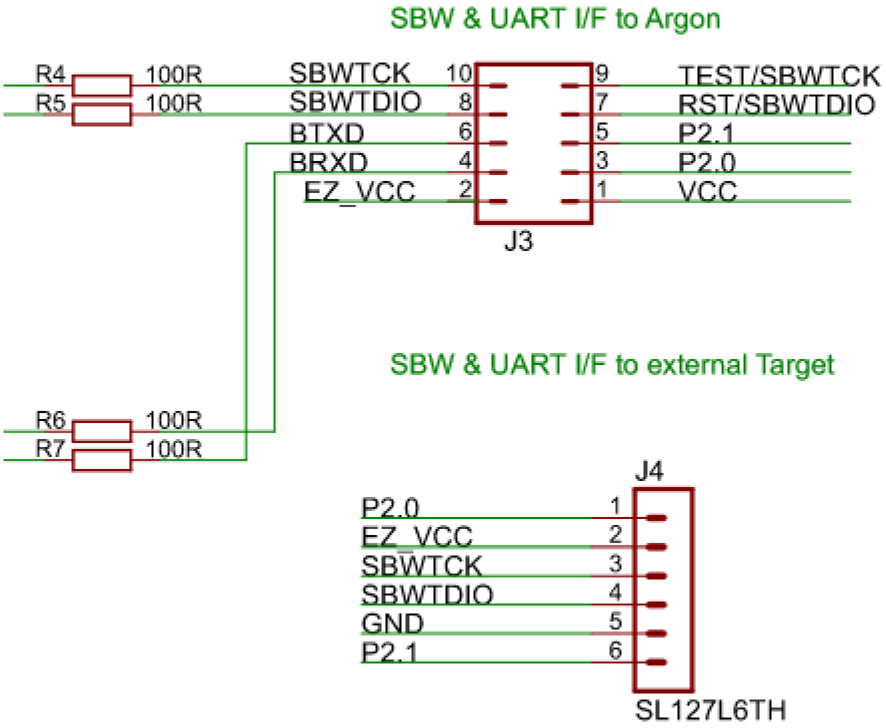


Figure 19 MSP430 Serial Schematic

4.7. Wireless Module Unit

The vehicle’s wireless module utilizes a SPWF01SA intelligent Wi-Fi module. This module is a plug and play, standalone device that allows for wireless internet connection utilizing 2.4GHZ band 802.11 b/g/n standards. The module is driven internally by an STM32 ARM Cortex-M3 microcontroller and houses its own voltage regulators and clock systems. The device supports integrated TCP/ IP protocols, up to eight simultaneous TCP or UDP client socket connection, one server socket connection, WEP/WPA/WPA2 security, miniAP mode, UART integration, and sixteen configurable GPIO pins. The SPWF01SA is mounted on a custom ASIC that allows connectivity to the MSP420 FRAM Experimenter board through appropriate control board port pins. This also allows the hardware configuration to match the design needed for utilization with the vehicle’s systems. LEDs were included on the custom ASIC to view signal states at runtime. Below is a schematic of the custom wireless ASIC board as well as the port layout on the pin header.

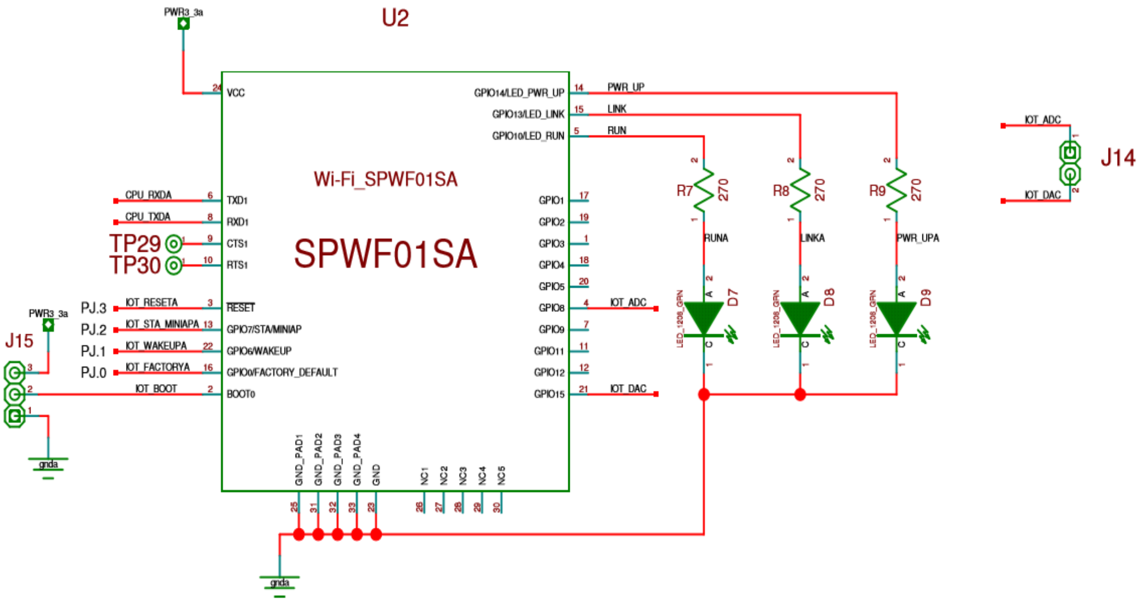


Figure 20 SPWF01SA Wireless Module Schematic

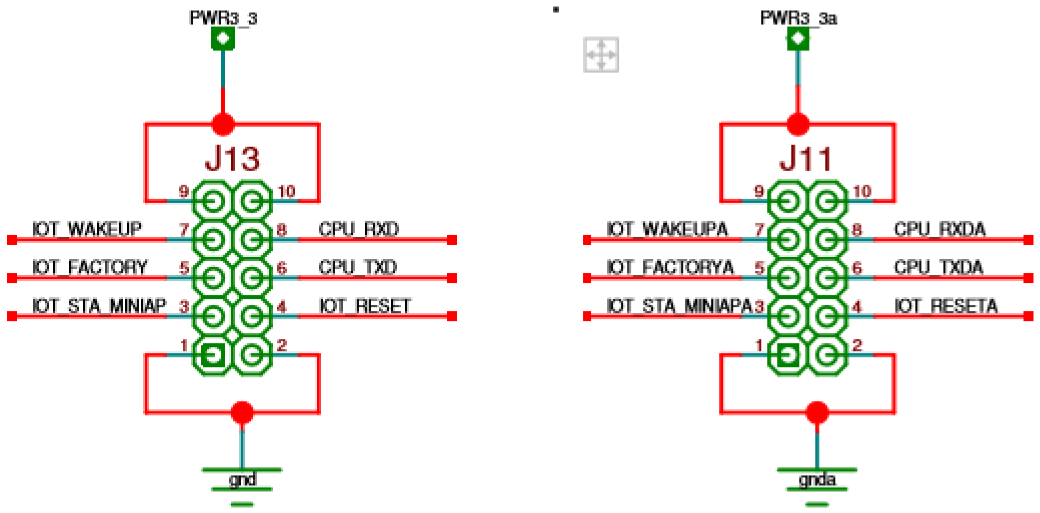


Figure 21 Wireless Module Port Pin Breakdowns

5. Power Analysis

This section gives an in-depth analysis of the vehicle's power consumption statistics and battery lifetime calculations. The battery-life calculations started with electrical current measurements from the vehicle's power supply while running different runtime operations. Each measurement listed below is the average of thirty samples taken from three different vehicles with ten samples per vehicle. The basic formulas listed below were then used in parallel with the sample data and the included battery specification details below.

Power = Voltage * Current

CURRENT DRAW

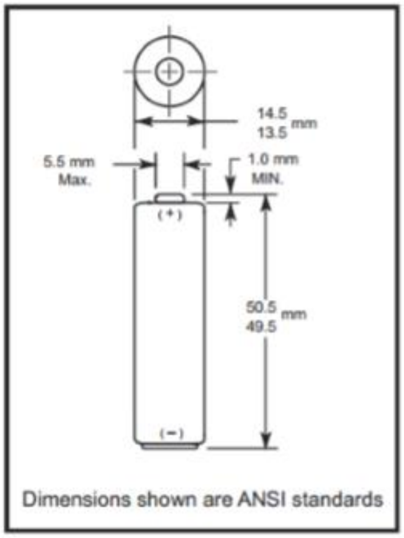
WITHOUT IR EMITTER	mA
IDLE	141.9
IDLE WI-FI	139.8
FORWARD	307.5
BACKWARDS	308.0
CLOCKWISE	307.7
COUNTER CLOCKWISE	307.0

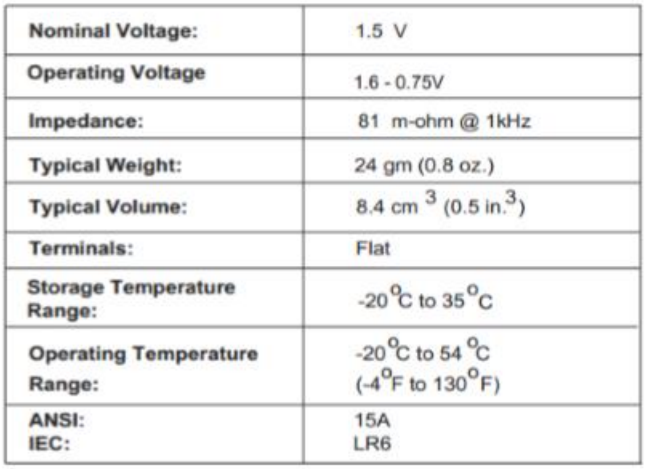
WITH IR EMITTER	mA
IDLE	201.3
IDLE WI-FI	197.1
FORWARD	326.1
BACKWARDS	329.4
CLOCKWISE	324.8
COUNTER CLOCKWISE	325.1

AVERAGES	
IDLE AVERAGE	170.0
DRIVE AVERAGE	317.0
RUNTIME AVERAGE	247.2

BOUNDS	
RUNTIME MIN	97.8
RUNTIME PEAK	404.2

Figure 22 Vehicle Current Draw in Milliamps during Different Operations





Nominal Voltage:	1.5 V
Operating Voltage	1.6 - 0.75V
Impedance:	81 m-ohm @ 1kHz
Typical Weight:	24 gm (0.8 oz.)
Typical Volume:	8.4 cm^{3} (0.5 in.^{3})
Terminals:	Flat
Storage Temperature Range:	-20°C to 35°C
Operating Temperature Range:	-20°C to 54 °C (-4°F to 130°F)
ANSI: **IEC:**	15A LR6

Figure 23 Common AA Battery Specifications

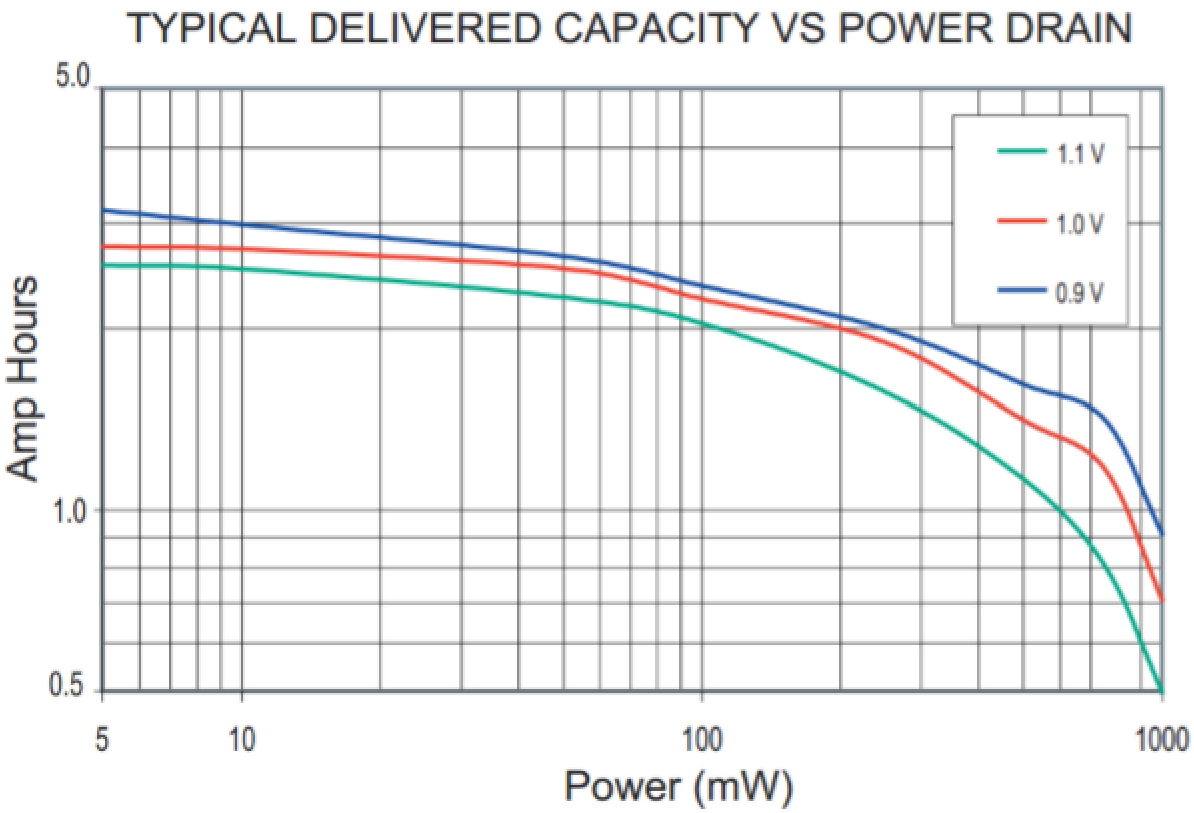


Figure 24 Common AA Battery Power (mW) vs. Amp-Hours (Ah)

Substituting the data and specifications listed above, the formula results in the following:

Power = (3.3 v) * (Current)

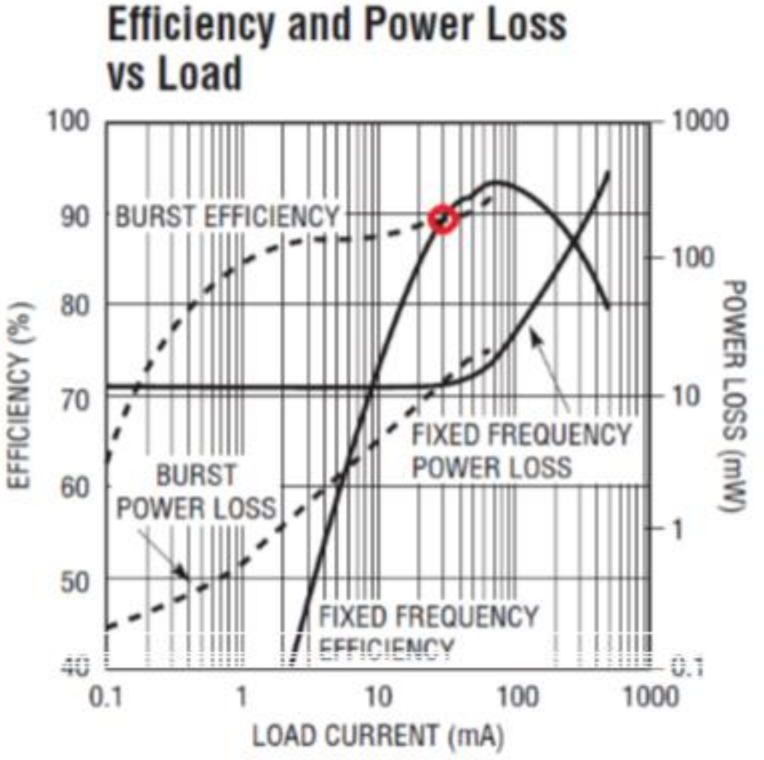


Figure 25 Efficiency of Buck-Boost Converter

Based on the figure above, the Buck-Boost converter delivers power from the batteries at approximately 90% efficiency. This calculation is also with regards to all four batteries. Thus we derive the following:

Power = (((3.3 v) * (Current)) / 90% efficiency) / 4 batteries
= ((3.3 v) * (Current)) / (0.9 * 4)
= 0.91666666667 * Current

Using the **Common AA Battery Power (mW) vs. Amp-Hours (Ah)** chart, we can substitute the above power calculation into the chart and find the total value for the system's Amp-Hours. Once we have the value for Amp-Hours, simply substitute the calculated amp-hour and current draw values into the following formula below to find time in hours:

Time = (Amp-Hours) / (Current)

Below are the results of substituting the sampled current draw data into the provided procedure above.

CURRENT MEASUREMENT	AMP-HOURS	BATTERY LIFETIME
IDLE CURRENT DRAW AVERAGE	~1.89 Ah	**11.12 Hours**
DRIVE CURRENT DRAW AVERAGE	~1.77 Ah	**5.59 Hours**
RUNTIME CURRENT DRAW AVERAGE	~1.85 Ah	**7.49 Hours**

Figure 26 Battery Life Calculations

6. Test Process

This section gives an overview of the methods used to test the design and functionality of the device extensively. These qualifications ensure quality design and stability of the product. Brief descriptions of each test method may be found in the appropriate section below.

6.1. Power Supply Unit

Before installing the components for the Buck-Boost converter circuit, several resistors were added to the board. After these components were soldered, they were tested using an ohm-meter to ensure that they were installed properly and had the expected resistance values. After this step was completed, the resistance between the pins of the LT3532 Buck-Boost converter was checked using an ohm-meter, to ensure the Buck-Boost converter was installed properly. Pads that were checked are numbered sequentially and may be viewed below.

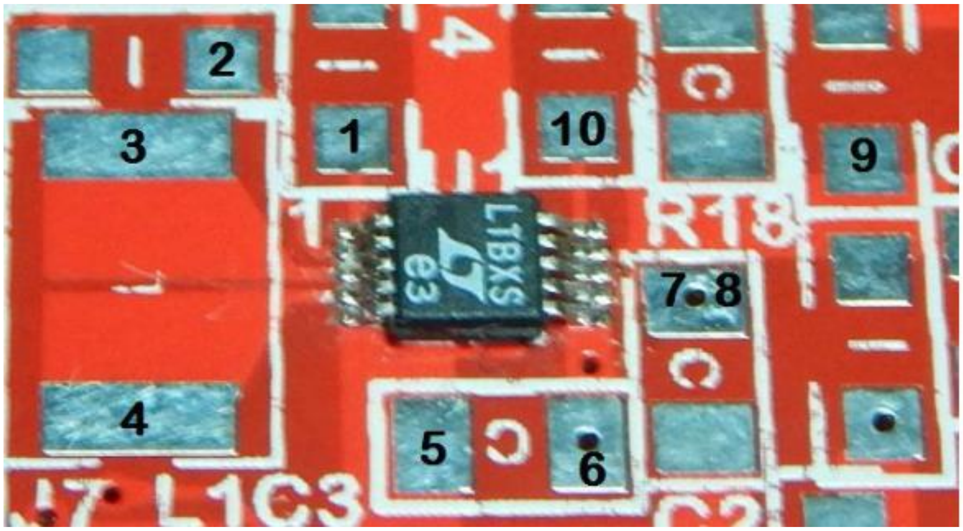


Figure 27 Buck-Boost Converter Pins

Once the pins were tested, each component of the Buck-Boost converter was installed. After each component was soldered, the resistance was tested to ensure that it was installed properly. This step was followed by the switch for the battery pack and then the power header was installed onto the control board. The soldering points of these components were tested with an ohm-meter to ensure that they were not shorted. After installation of all the necessary components, the output DC voltage was measured at the “3.3 V” test point using a volt-meter to verify that the Buck-Boost converter provided approximately 3.3 V.

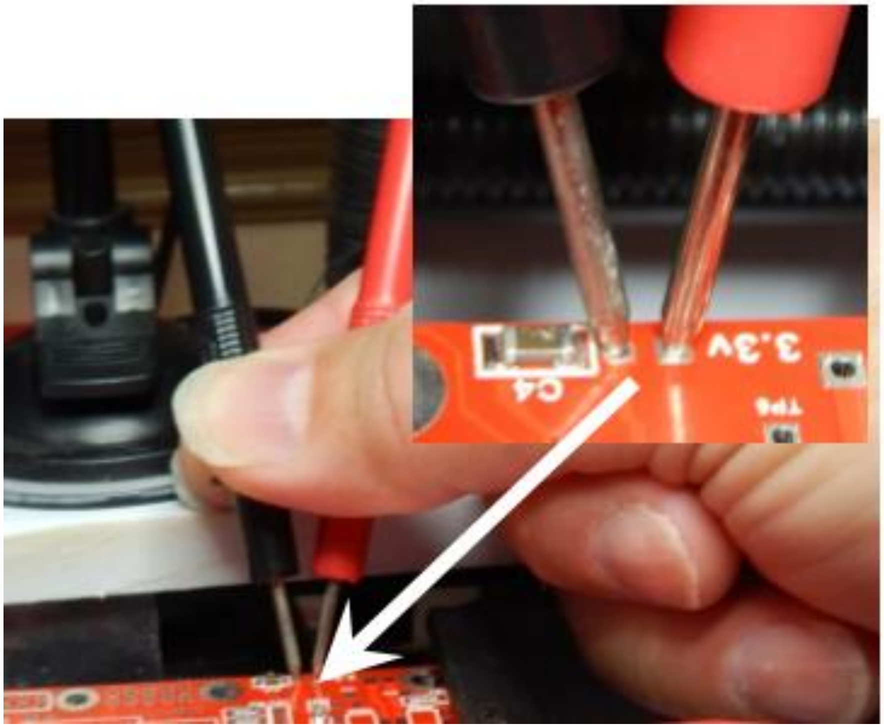


Figure 28 PSU Test Procedures

6.2. FRAM Experimenter Board

For the installation of the FRAM Experimenter Board, additional components were added to the control board to allow proper mating of the devices. These female docking connectors were also tested using similar procedures as the components in the PSU. After these components were added, pins used to connect the Control Board were mounted to the FRAM Experimenter Board. Each pin's resistance was checked with neighboring pins to ensure that no pins were short circuited during installation.

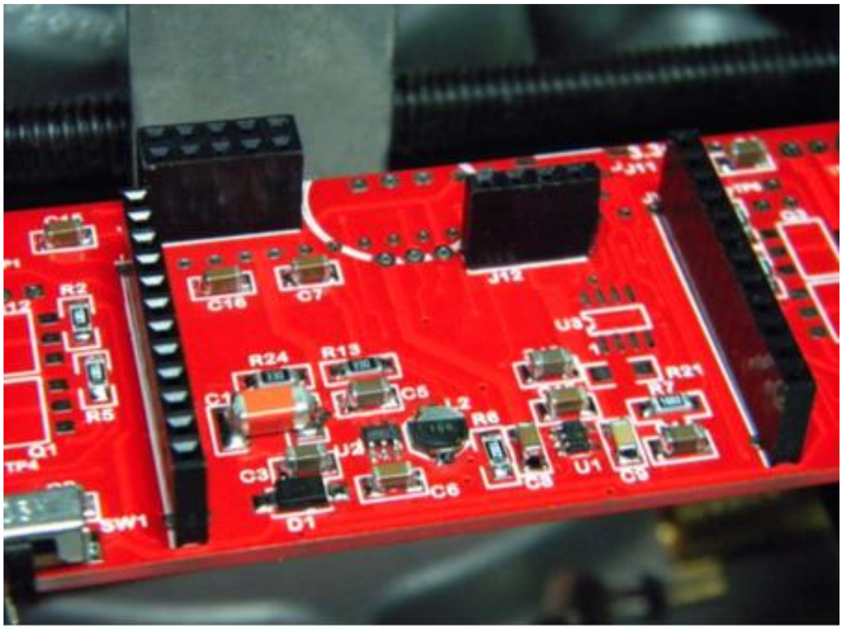
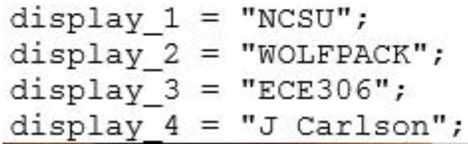


Figure 29 Connector Pins Installed in FRAM Board

6.3. LCD Screen

After the LCD Screen was raise-mounted to the Control Board, solder connections were verified to be flush to the board while still providing sufficient connectivity to the device. Software was then loaded to the board to verify the LCD would display text properly from the board. The default display test method may be seen in the figure below:



```
display_1 = "NCSU";
display_2 = "WOLFPACK";
display_3 = "ECE306";
display_4 = "J Carlson";
```

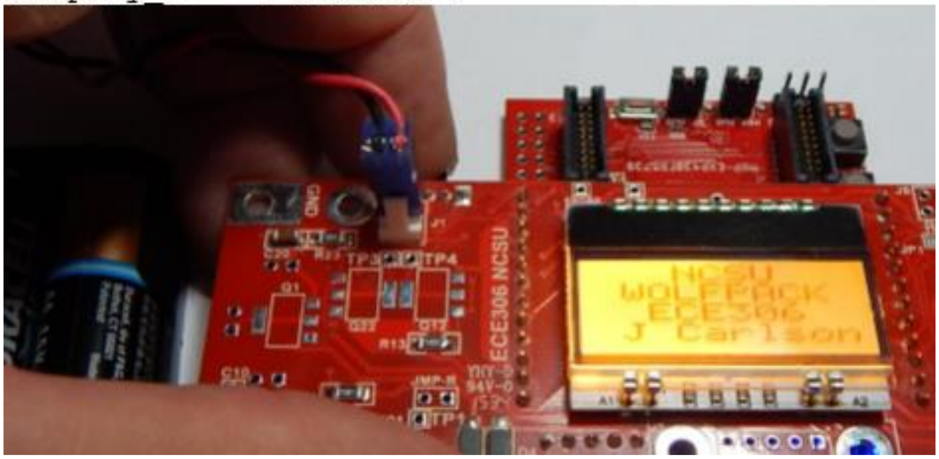


Figure 30 Default Display of LCD Screen

6.4. N-FET Testing of Left and Right Forward Motor Control

Testing the N-FETs for forward motor control involved first creating a jumper cable that could plug into the pins of the JV2 connector on the Control Board. The jumper cable was connected to pin 12 and pin 6 of JV2 to test forward motion of the right motor. The jumper cable allowed power to be supplied to the circuit without powering up the whole system. This method was utilized for testing purposes and to prevent damage to the board. After the battery pack was connected, the power switch was turned on and the voltage was measured across the connection at J21 using a multi-meter. A depiction of this procedure can be seen in the figure below:

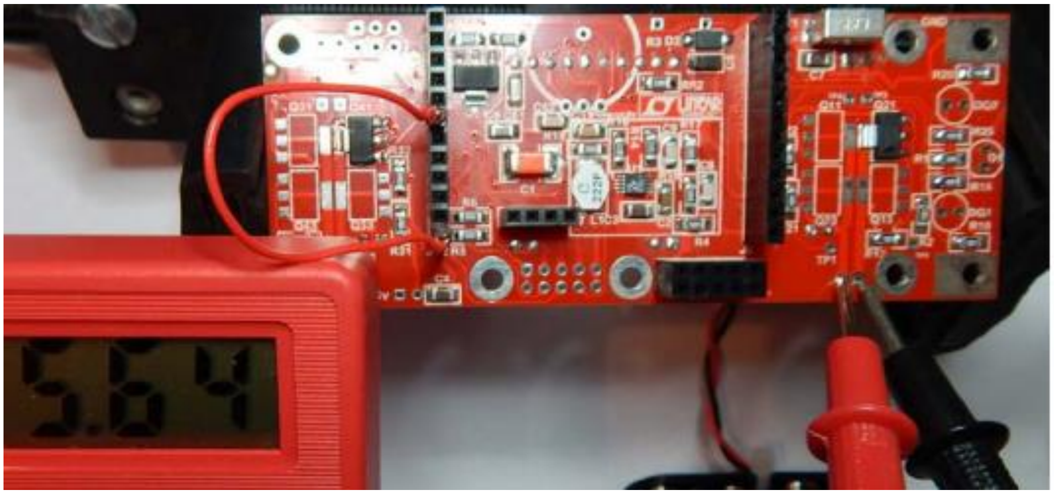


Figure 31 Set Up for Testing Forward Motor Control for Right Motor

Testing for forward motion of the left motor was done in a similar fashion as previously outlined. The difference is that the jumper cable was connected to pin 12 and pin 4, while the multi-meter was connected across the connection at J43. This simply powered a different portion of the circuit, and allowed to board to be partially powered on. This method was utilized for testing purposes and to prevent damage to the board. A depiction of this procedure can be seen in the figure below:

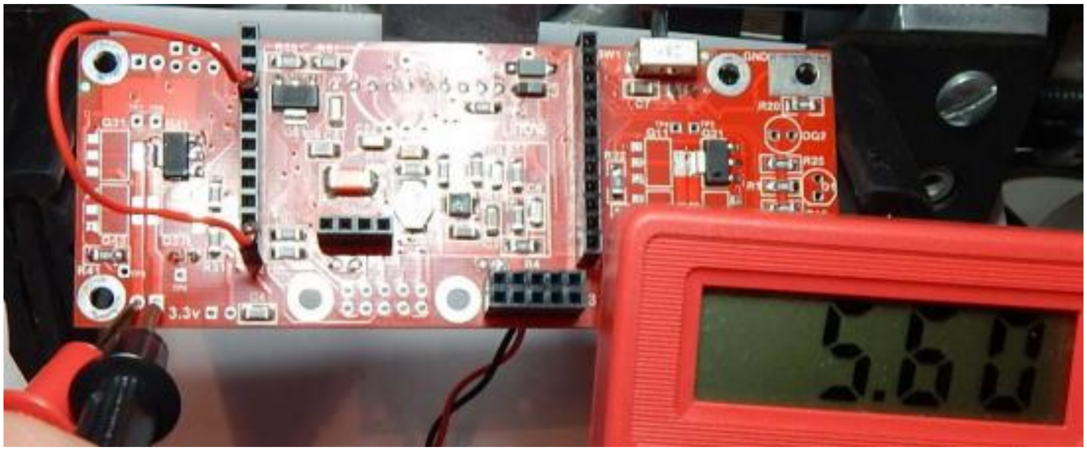


Figure 32 Set Up for Testing Forward Motor Control for Left Motor

6.5. Testing Forward and Reverse Motor Functionality

Testing for both forward and reverse motion of the motors involved first installing the remaining N-FETs to both the top and backside of the Control Board as seen in the following figures:

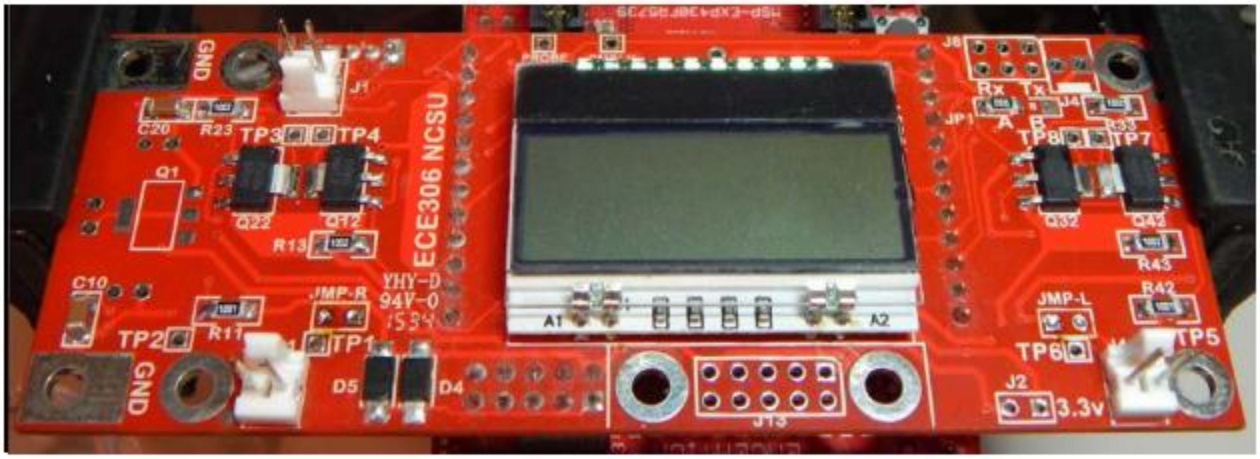


Figure 33 Top View of Control Board for N-FET Installation

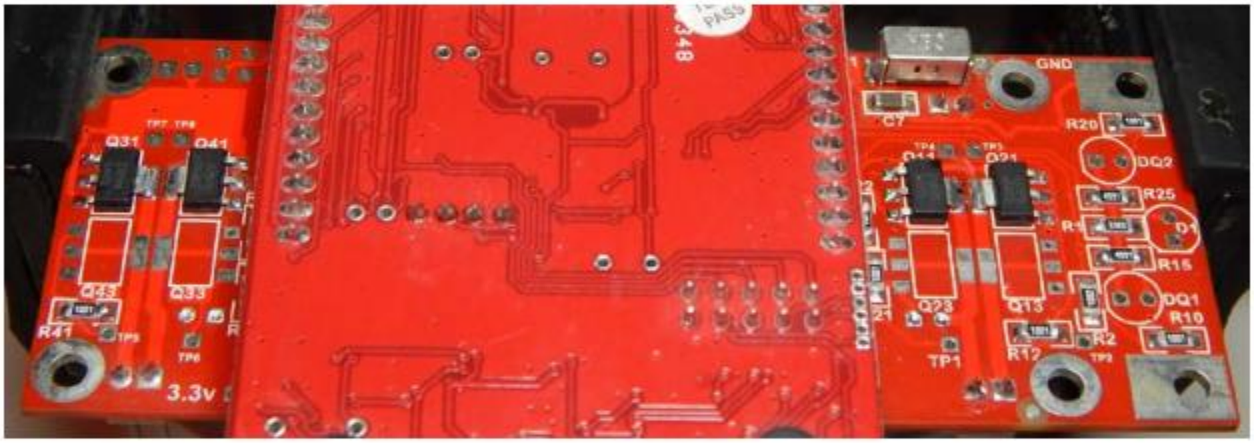


Figure 34 Backside of Control Board for N-FET Installation

Once the N-FETs were installed, the motors were disconnected from the board and test code was written for forward and reverse functionality of the motors. Prior to testing, a multi-meter was obtained, a battery pack was connected to the board, and the power source was switched on.

The forward motion of the right motor was tested first. With the battery on and the forward motion of the right motor off, the negative lead of the multi-meter was connected to ground and the positive lead was connected to point TP1 and then later TP3. The voltages at these two points were equal to the battery voltage. Then, the forward motion of the right motor was turned on and points TP1 and TP3 were measured once more. TP1 was roughly half the voltage of the batteries and TP3 was roughly at ground potential. Major discrepancies meant that the hardware was installed incorrectly. Once these points were tested, the motor was turned completely off.

Next the reverse motion of the right motor was tested. With the battery on and the reverse motion for the right motor off, the negative lead of the multi-meter was once again connected to ground while the positive lead was connected to point TP2 and then later TP4. Given the conditions of the battery pack and the motor, TP2 and TP4 were equal to the battery voltage. Then, the reverse motion of the right motor was turned on and TP2 and TP4 were measured once more. TP2 was seen to carry half the battery voltage and TP4 was roughly at ground potential. Again major discrepancies meant that the hardware was installed incorrectly. Once these points were tested, the motor was once again turned completely off.

Next the forward motion of the left motor was tested. With the battery on and the forward motion of the left motor off, the negative lead of the multi-meter was connected to ground while the positive lead was connected to point TP5 and then later TP7. TP5 and TP7 were both approximately equal to the battery voltage. Then the forward motion of the left motor was turned on and TP5 and TP7 were measured again. TP5 had roughly half the battery voltage and TP7 was roughly at ground. Major discrepancies meant that the hardware was installed incorrectly. Once these points were tested, the left motor was turned completely off.

Finally, the reverse motion of the left motor was tested. With the battery on and the reverse motion of the left motor off, the black lead of the multi-meter was connected to ground and the red lead was connected to point TP6 and later TP8. TP6 and TP8 were both approximately equal to the battery voltage. Then the reverse motion of the left motor was turned on and TP6 and TP8 were measured again. TP6 was roughly half the battery voltage and TP8 was roughly at ground potential. Major discrepancies meant that the hardware was installed incorrectly. Once these points were tested, the left motor was turned completely off.

Once testing was complete, the hardware had been verified to be functioning as expected. This confirmed that the remaining P-FETs could be installed safely to the Control Board. Prior to testing each state, forward and reverse of each motor right and motor left, it was important to always turn the motor completely off. Having both states enabled at the same time on one motor would overload the P-FET potentially resulting in damage to the board, motors, or vehicle. It is imperative to test each state separately and turn the motor completely off between each state to avoid severe damage to the system.

6.6. Detection Unit

Before installing the Detection Unit, it was important to ensure the LCD Screen's pins and solder were flush with the board. Once the pins were ensured to have no overhead, an insulator was placed over the pins to prevent shorts while installing the thumb wheel.

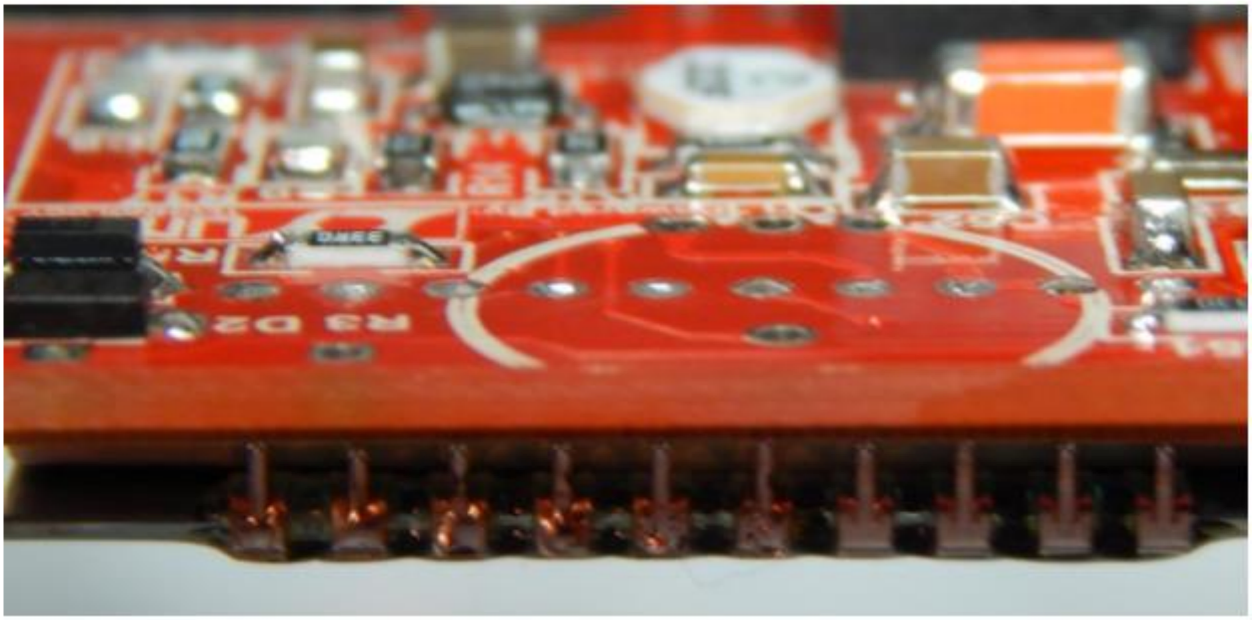


Figure 35 Flush LCD Screen Pins

An additional insulator was also attached to the thumb wheel as a preventative measure for shorts between the LCD and thumb wheel.

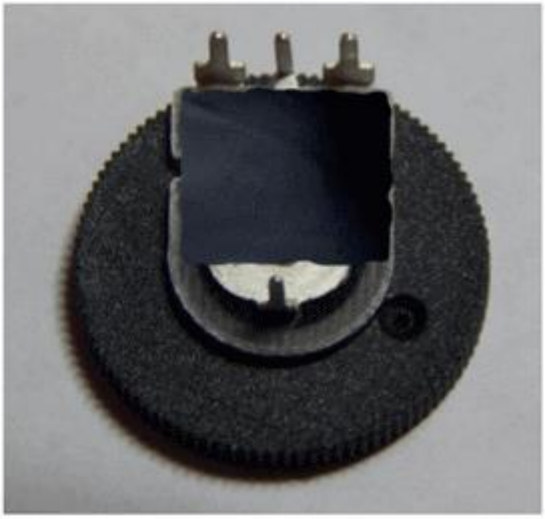


Figure 36 Insulated Thumb Wheel

Once the thumb wheel was safely installed on the control board, it was used to test the programming for the ADC. The thumb wheel was set to be the detected analog signal on the ADC's input channel and thus used to verify that the code was correct before switching over to the variability of the detector circuits. Code was drafted and implemented to convert up to 12-bits of binary data into a hexadecimal value that was displayable on the vehicle's LCD Screen. The outputted hexadecimal value was used for debugging purposes in real-time while testing code.

Black line detection testing was a series of trial-and-error tests with adjustments to ambient light calibration code, detection unit positioning, and different testing environments. The vehicle would be calibrated and tested in different environments, and then data would be collected on the vehicles behavior in that environment. The data collected was used to update code appropriately to change the behavior of the vehicle in the different environments. Whenever behavioral code was added or changed, the vehicle was re-evaluated in the environments again to ensure it was functioning properly. The testing process included the verification of the calibration mode on the vehicle, that the detectors were outputting expected or appropriate data to the LCD Screen, and that the vehicle would detect a black line and operate accordingly. Trial-and-error testing was essential for this portion to make sure that the vehicle met the conditions of variable environments.

6.7. Serial Communication Unit

Before linking the vehicles together to begin communications, it was essential to verify that the expected baud rate was generated and information was being transmitted from the vehicles board. In order to check this, an oscilloscope probe was mounted on the open TX and RX pins on the control board with reference to ground. This allowed the manual verification of both the baud rate and expected data, including parity, stop bits, and packet sizing. It was also important to test implemented code locally on one vehicle’s system. This was possible by utilizes a loopback method, or shorting both receive and transmit pins on the J8 header. These pins on the J8 header are shown below in Figure 15. This internal communication loop allowed testing the transmission and reception of valid data. Once these tests were completed, the code was adapted to function across multiple FRAM Experimenter boards at one time. These communication lines were later used to interface between the MSP430 FRAM Experimenter board and the wireless module.

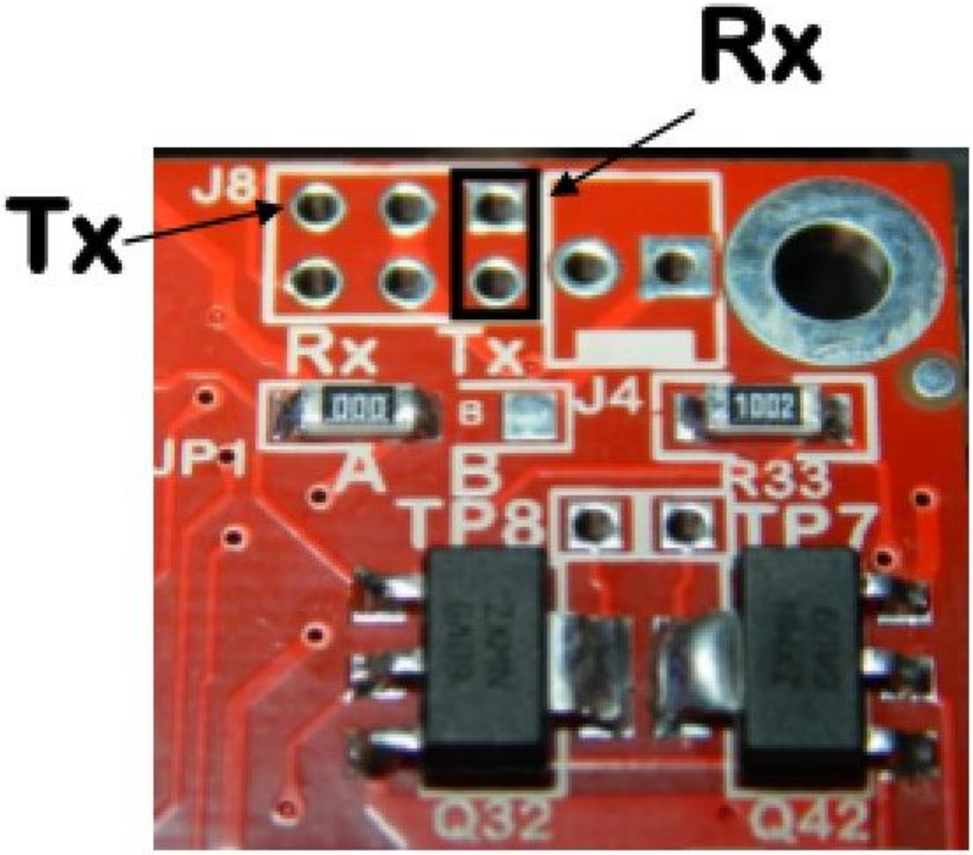


Figure 37 Control Board CPU TX & RX Pins

6.8. Internet of Things (IOT) Device

In order to establish bidirectional communication with the IOT device, both serial ports needed to first be configured to run at a baud rate of 9600. Using a terminal emulator set at 9600 baud, 8 data bits, 1 stop bit, no hardware flow control, and no local echo, a connection could be established to test and check the communication between the device and the PC. The Control Board was first removed from the FRAM Board and jumpers were then replaced onto J3 as seen in Figure 30. The TXD and RXD pins were linked to each other, providing a hardware loop back with the PC, using only the programming interface instead of the FRAM board and software code.

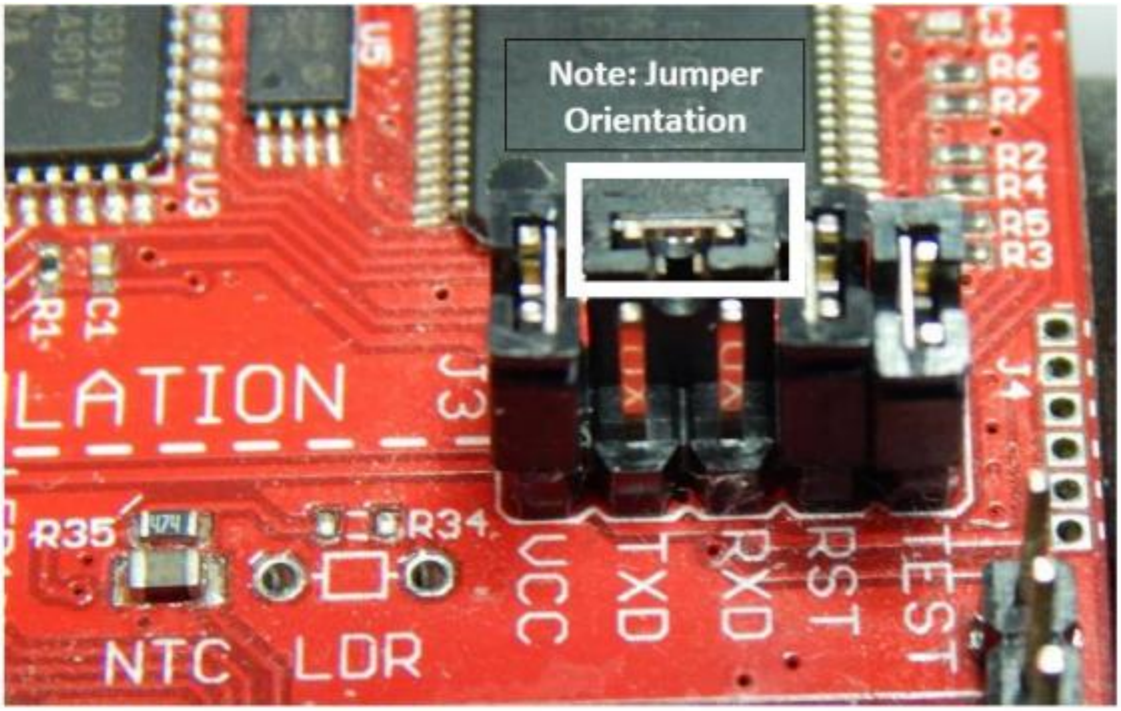

Note: Jumper Orientation

Figure 38 Jumper on TXD and RXD pins

Once this set up was completed, typing a character in the emulator resulted in the same character showing up in the emulator window as well. Typing in the terminal emulator and echoing the character back to the PC helped to insure that the information typed on the PC was received. For further testing, the interrupt service routines for both the PC and the IOT device were modified with the creation of a global variable set to “OFF” in the PC port configuration function. When only the first character was received from the PC receive interrupt service routine, the global variable was set to be “ON”. Because the variable is now on and both interfaces were set to run at 9600 baud, a character could be written to the PC back door transmit buffer from the receive interrupt service routine of the IOT device.

7. Software

In this section we discuss the general overview of how the software functions and is implemented on the vehicle. The primary module is in the “main.c” source file. This code is what runs when the FRAM Experimenter board is powered on. The main code starts by referencing the header files to utilize during runtime. These files are “msp430.h,” “functions.h,” and “macros.h.” The header files include variable declarations and parameters that the system references frequently at runtime. Next, main initializes the global variables from “macros.h” so that the system knows to reference them in other source files and functions. Next certain variables are initialized to their appropriate values and several functions are called to initialize system components. Main initializes ports, clocks, conditions, timers, LEDs, the LCD, the ADC, and the CPU UART communication in that order. After the vehicle’s systems are initialized, the code instructs the FRAM board to set the “splash” screen displaying our team information and begins standard operation. During standard operation the code loops continuously from a timer-based variable. This loop checks for various user inputs and updates the LCD display with the latest information. If any of the timers go over their maximum allotted time, they are reset to zero to ensure appropriate operation without any overflowing counter variables.

During the initialization, main calls a function that configures the system’s integrated ports. This is done by manipulating each ports control register to indicate particular settings for the specified port. For each port pin, the port initializes the select bits, the output bits, and then the direction bits. Port initialization almost always follows this order of bit manipulation. The ports that are configured by this function are Ports 1, 2, 3, 4, and J during the start of main. This section is important for it allows the pins to be utilized properly as inputs, outputs, or various other system purposes such as clock wires. Without this many subsystems would not be accessible or entirely inoperable.

An important part of the system design is the utilization of timers and timing. Timer initializations functions are called in the “timers.c” code. The configuration of timer A0 runs in the “timersA0.c” source file. Timers are configured to run at a specified tick rate by the implemented code. In this case, A0 is configured to run at a one millisecond tick. Therefore, every one millisecond the A0 timer ISR is triggered and handled. These timers are used for system delays and various synchronous functions within the system. If a timer goes over a certain threshold, the ISR resets the timer variable to avoid overflow and keep volatile value ranges consistent.

For the implementation of blackline following functionality, the vehicle must first configure and set the emitter and detectors to work with the ADC to pass the system usable, real-time variables. This starts with the initialization of the ADC module. During the initialization, the ADC is enabled, set to single channel, single sequence, at 10-bit resolution, with unsigned binary, at 200 kilo-samples per second, on channel A0. Once initialized and triggered, the ADC will read the values of that the detectors pick up from the emitter and convert it from analog input to usable digital data. Each time the ADC finishes converting an input, an ADC interrupt occurs telling the system that it can now utilize this data. Each time the ADC interrupt is triggered the next channel is set to be processed. Thus the ADC interrupt handler can address five different ADC channels. To trigger the next ADC process and channel, the “ADC10_Process” function is called in the desired section of code.

The vehicle has the ability to interface with other vehicles control systems via UART serial connection between CPU TX and RX pins. The vehicle starts by initializing variables and setting the USCIA1 to utilize SMCLK. There are several optional modes that may be set through the control registers. This can allow for parity bits, MSB transmission versus LSB, mode selection, number of stop bits, etc. with transmissions. Before locking the communication’s configuration process for operations, the code reconfigures the CPU_TXD and CPU_RXD pins that were reset by initializing UCA1. Once initialized, the vehicle is ready to receive and transmit information to another vehicle. It is important that the other vehicle on the communication line has the same UART networking configuration, i.e. parity, stop bits, etc. Similar to the ADC interrupt, the serial interrupt is triggered when it has usable data from the receive buffer. This information is then stored in an array that can be accessed throughout the code. To work with larger data transmissions, our vehicle breaks down the data from multiple received packets, performs appropriate operations, reworks the new data back into packets and sends them over to the receiving vehicle. This allows for more data to be transmitted and received outside the buffer length restrictions.

In order to implement the wireless functionality, the Wi-Fi module needed to be initialized, configured, and programmed to accept commands and send these commands to the FRAM Experimenter board to be processed appropriately. First of all, the wireless module is configured using a series of appropriately timed commands to get it in an initialized state. In this initialized state, the wireless module will attempt to connect to the specified wireless access point. Once connected, the wireless module opens a server socket that allows external clients to connect to the device. The vehicle also goes into a runtime, standby state and is ready to receive command from this external control source that is also connected to an internet access point. When commands are received, the vehicle performs the appropriate runtime actions.

To control the vehicle, there needed to be control software that allowed connection to the server socket over the established network connection. This control software came in the form of a java-based graphical user interface. The graphical user interface allows a user to connect over the network to the vehicle from its IP address and server port. Once connected the java client interprets keyboard input and translates them into correct runtime commands. These commands are sent to the vehicle over the wireless network where they are interpreted, processed, and run by the vehicle’s operating system.

8. Flow Chart

This section helps visualize each block of code by breaking down the process used with a flow chart. Each flow chart displays the logic process through the function using shapes and arrows. These graphical aids break the code down into easily followed blocks that point to each major action in the code.

8.1. Main Blocks

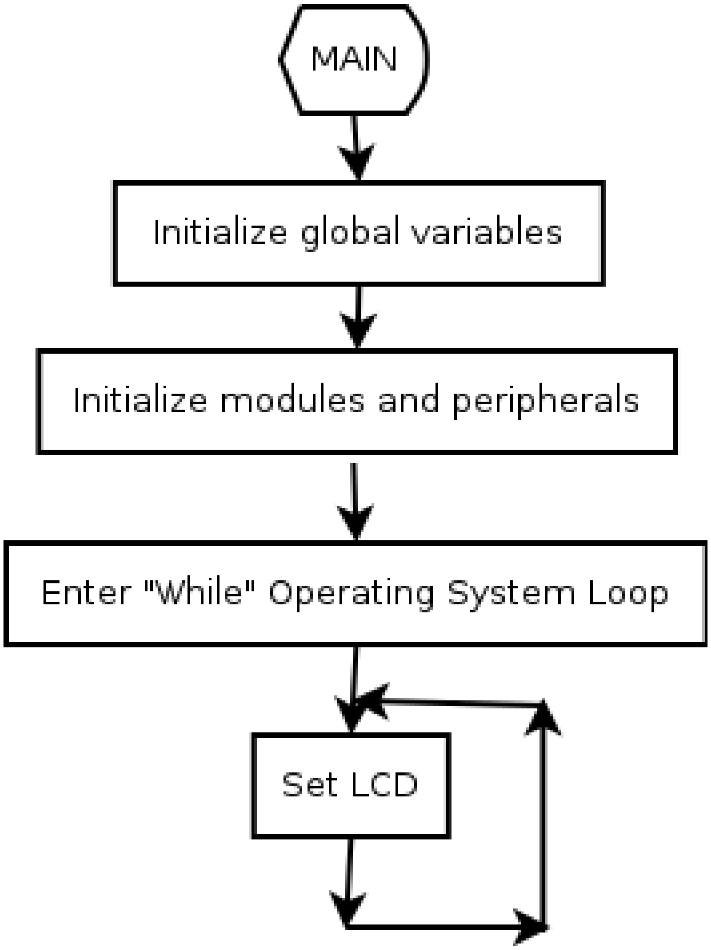

MAIN
Initialize global variables
Initialize modules and peripherals
Enter "While" Operating System Loop
Set LCD

Figure 39 Flow chart for main function in main.c

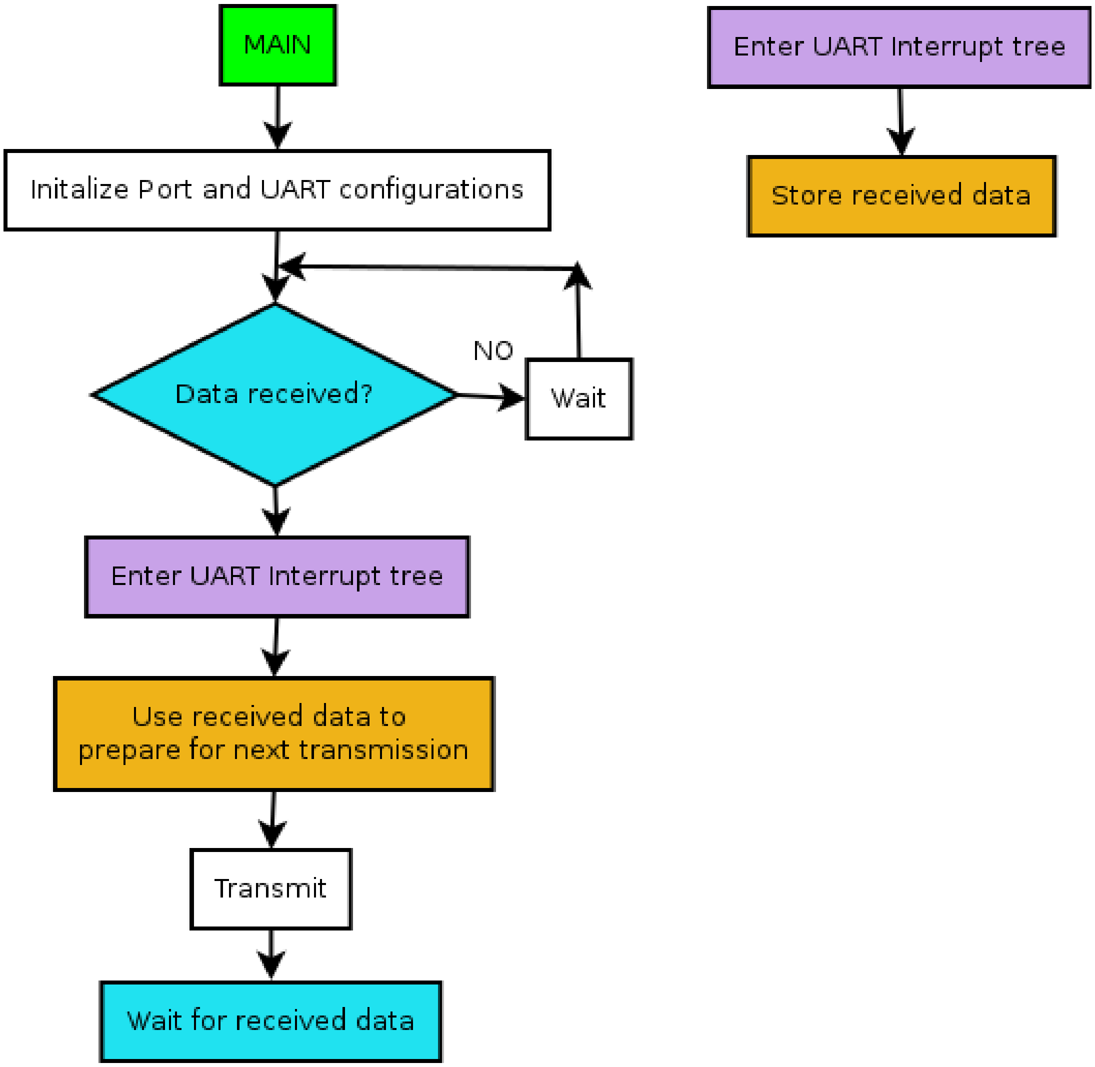

MAIN
Initalize Port and UART configurations
Data received?
NO
Wait
Enter UART Interrupt tree
Use received data to prepare for next transmission
Transmit
Wait for received data
Enter UART Interrupt tree
Store received data

Figure 40 Receive and Transmit flowchart

8.2. Initialization Blocks

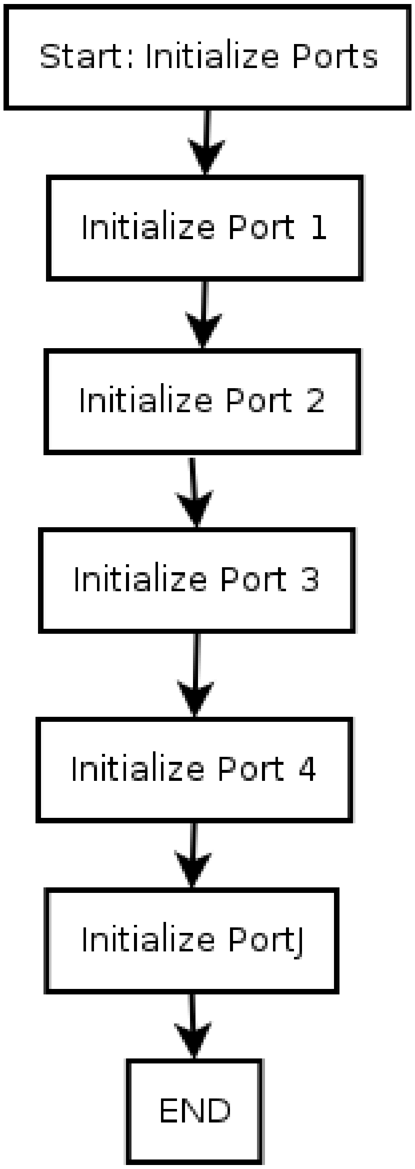


Figure 41 Flow chart for Init_Ports function in ports.c

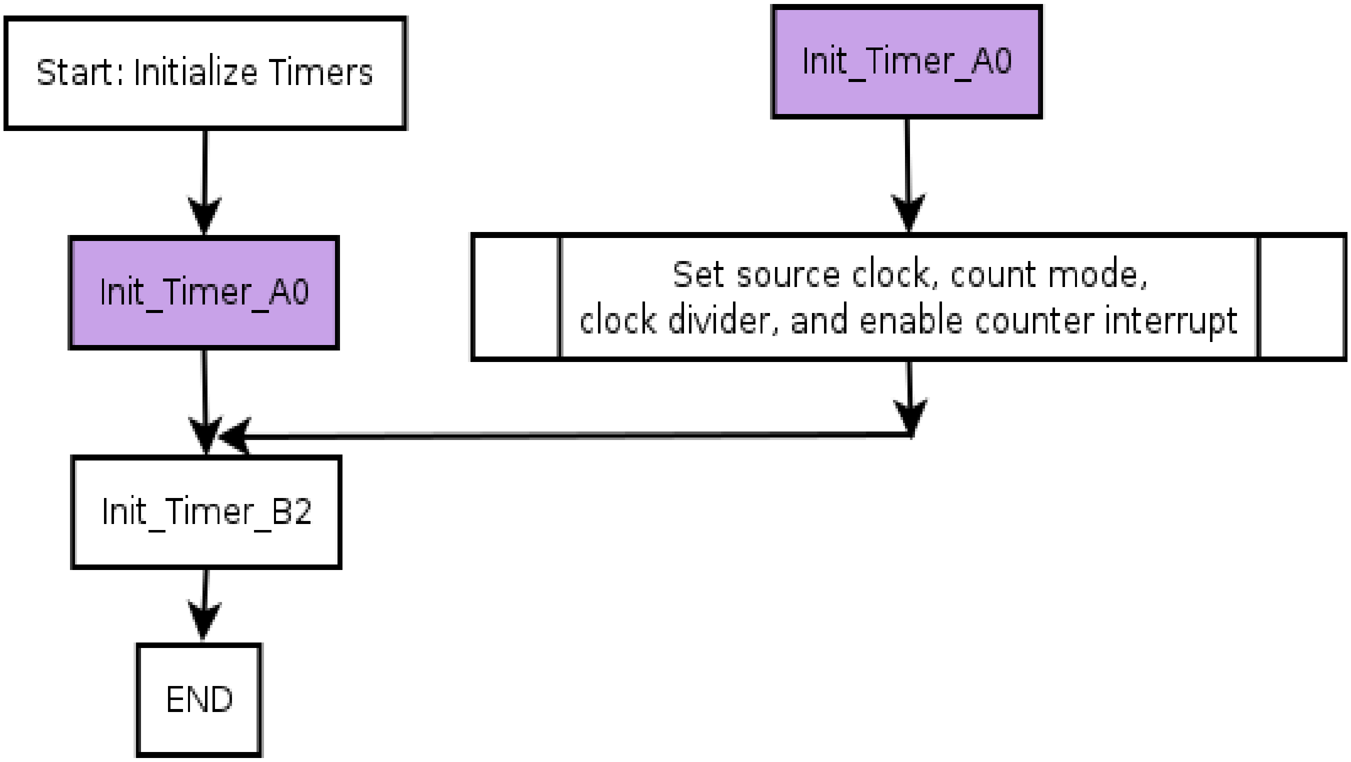

Start: Initialize Timers
Init_Timer_A0
Init_Timer_B2
END
Init_Timer_A0
Set source clock, count mode,
clock divider, and enable counter interrupt

Figure 42 Flow chart for Init_Timers function in timers.c

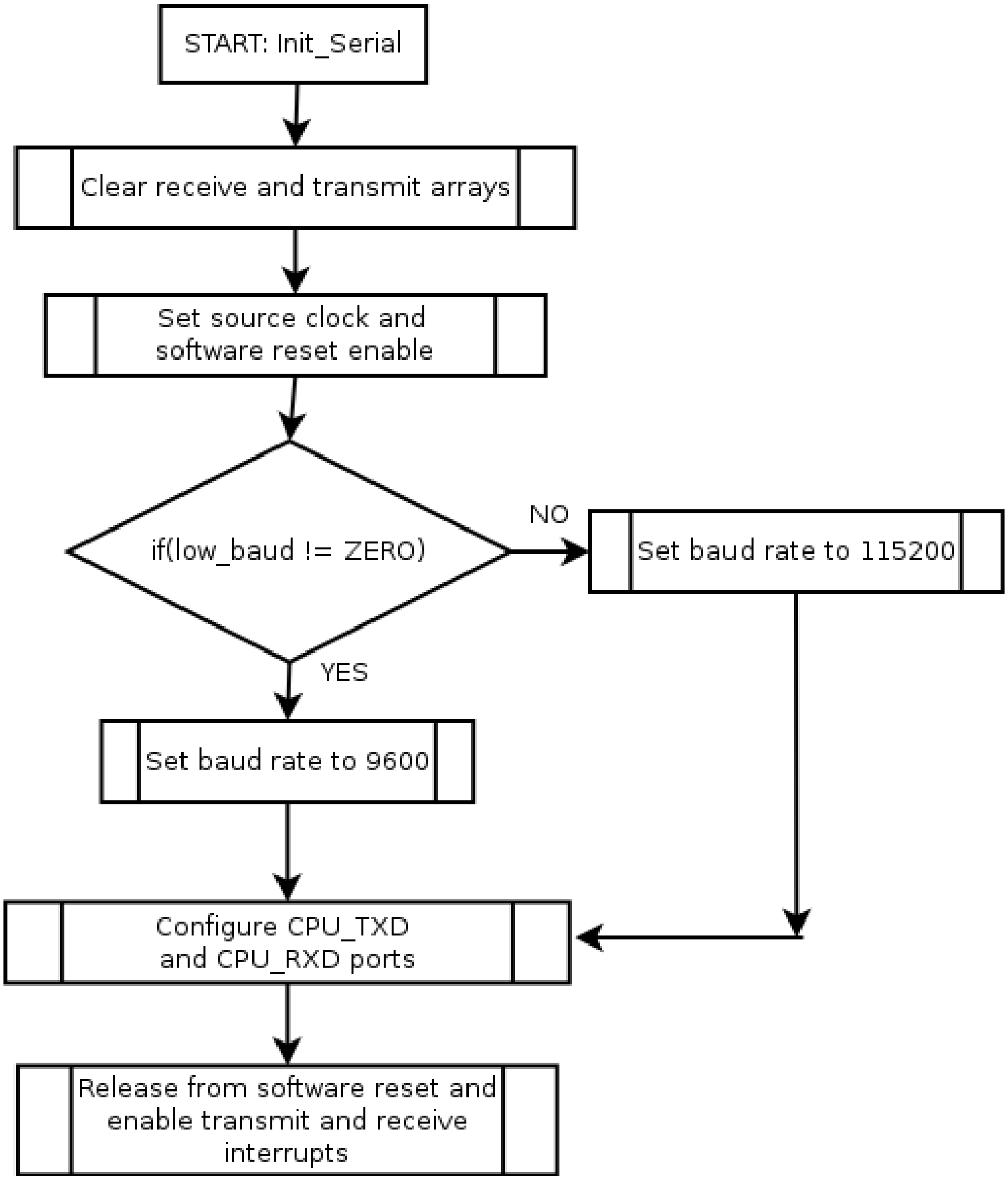

START: Init_Serial
Clear receive and transmit arrays
Set source clock and software reset enable
if(low_baud != ZERO)
NO
Set baud rate to 115200
YES
Set baud rate to 9600
Configure CPU_TXD and CPU_RXD ports
Release from software reset and enable transmit and receive interrupts

Figure 43 Function Init_Serial_UCA1 in init_serial.c

8.3. Interrupt Blocks

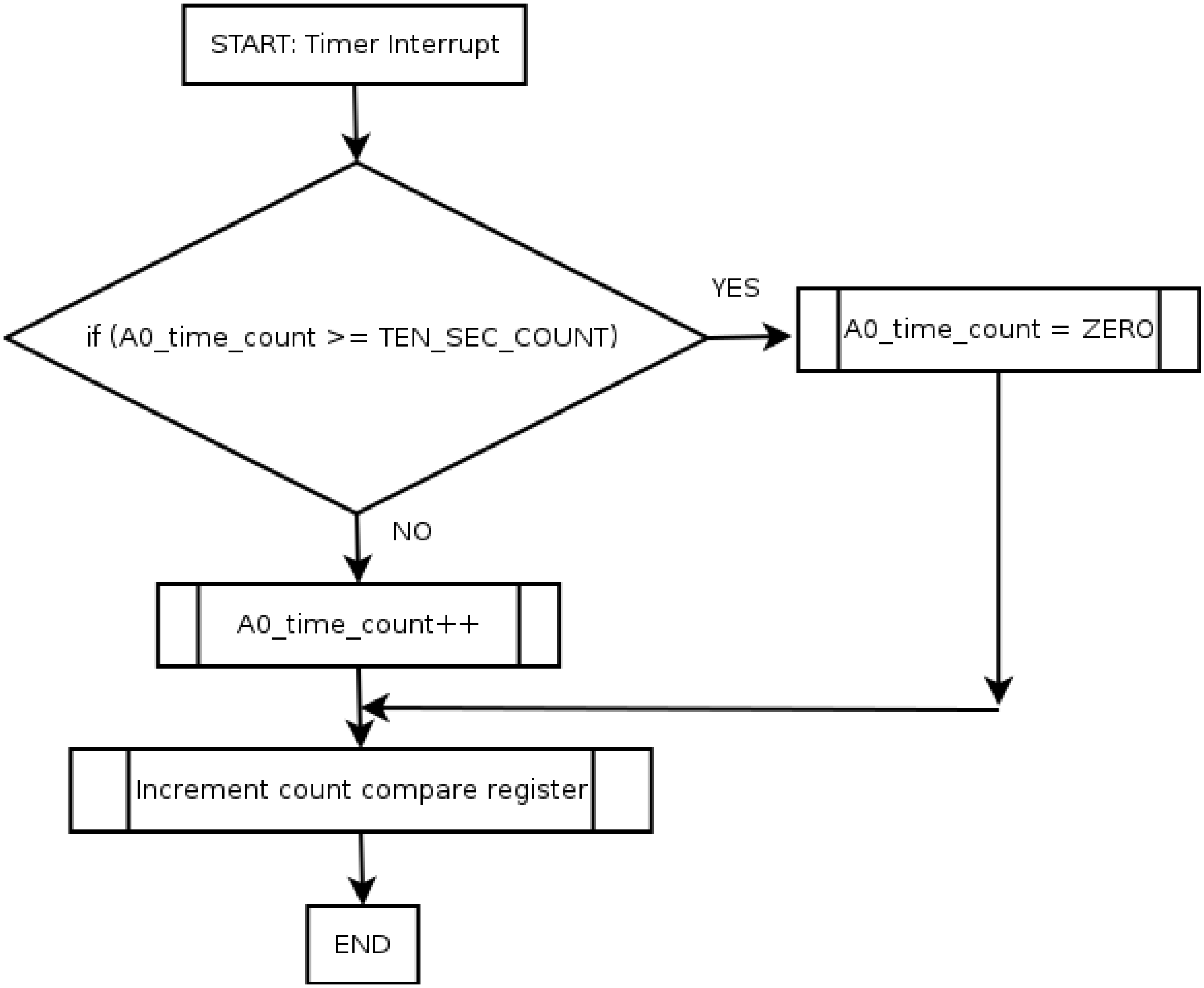

START: Timer Interrupt
if (A0_time_count >= TEN_SEC_COUNT)
YES
A0_time_count = ZERO
NO
A0_time_count++
Increment count compare register
END

Figure 44 Flow chart for TimerA0_Interrupt function in timer_interrupts.c

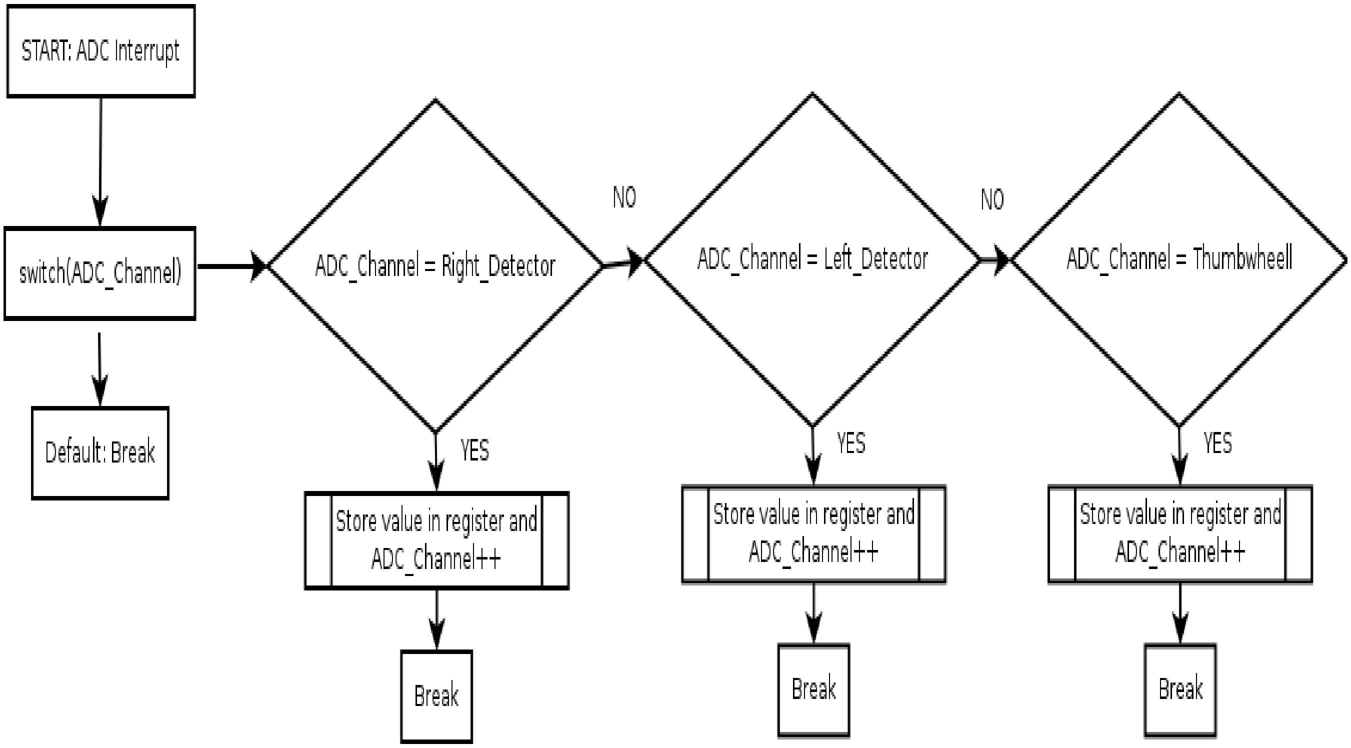

START: ADC Interrupt
switch(ADC_Channel)
Default: Break
ADC_Channel = Right_Detector
NO
ADC_Channel = Left_Detector
NO
ADC_Channel = Thumbwheell
YES
Store value in register and ADC_Channel++
Break
YES
Store value in register and ADC_Channel++
Break
YES
Store value in register and ADC_Channel++
Break

Figure 45 Flow chart for ADC10_ISR function in adc.c

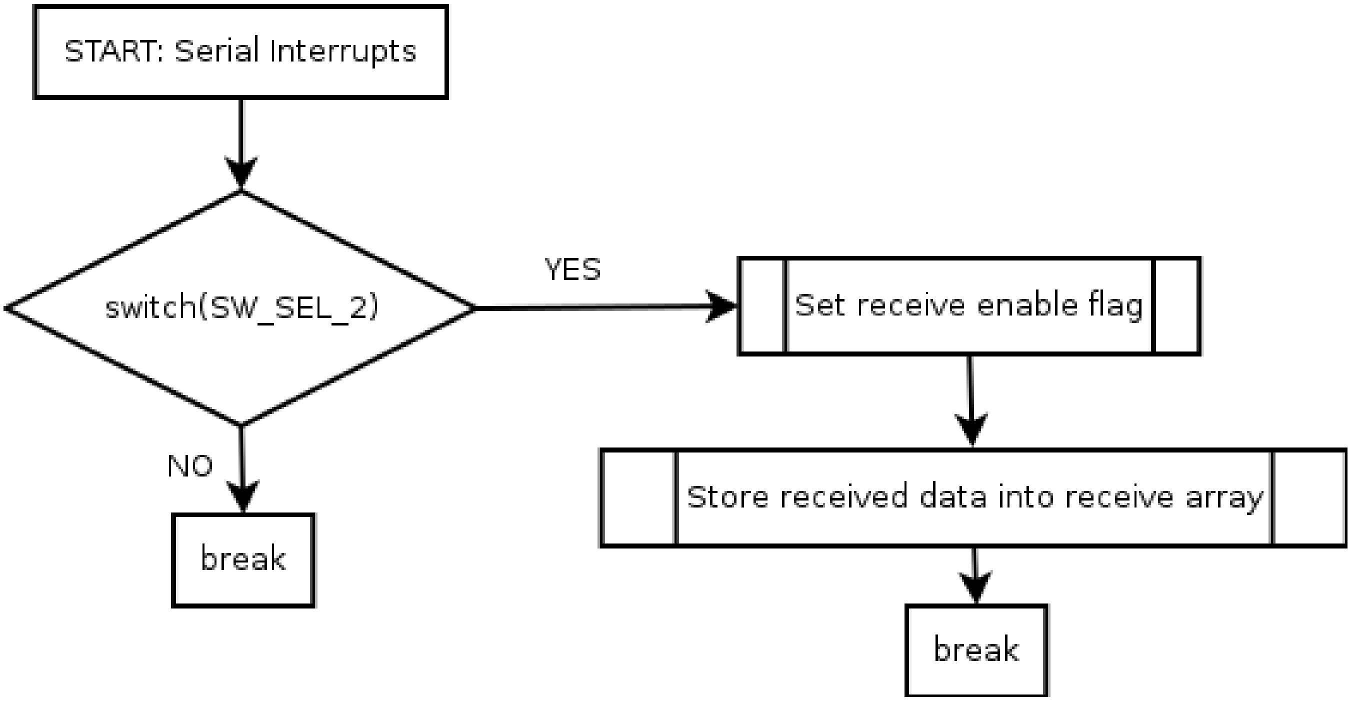


Figure 46 Flow chart for function USCI_A1_ISR in serial_interrupts.c

8.1. IOT Blocks

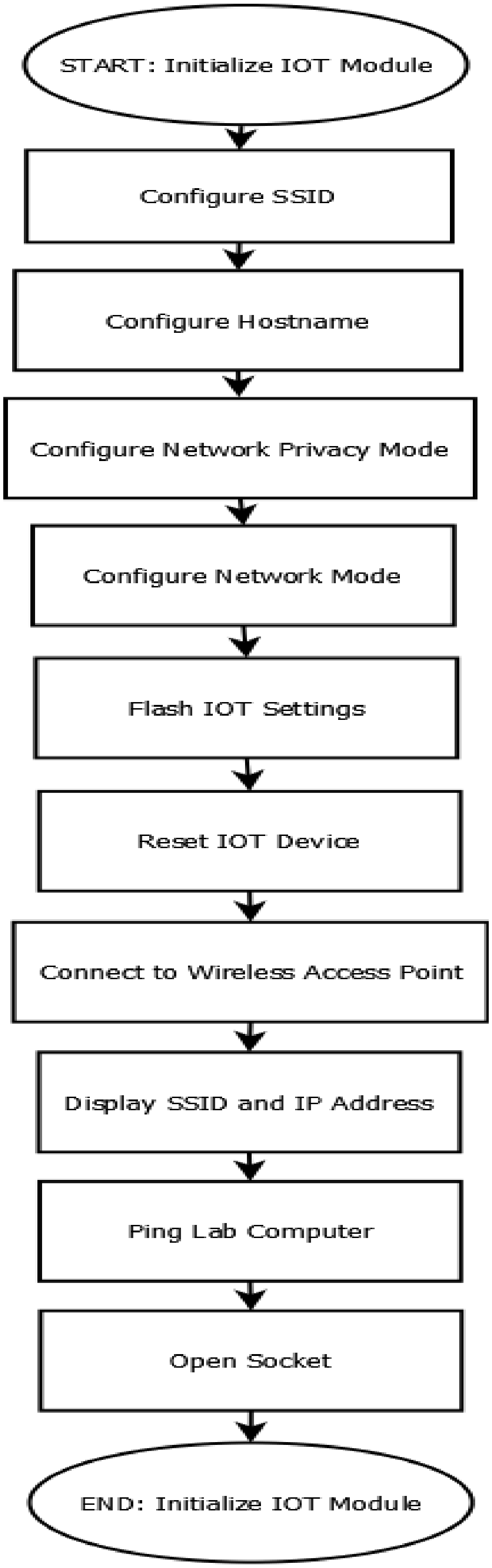

START: Initialize IOT Module
Configure SSID
Configure Hostname
Configure Network Privacy Mode
Configure Network Mode
Flash IOT Settings
Reset IOT Device
Connect to Wireless Access Point
Display SSID and IP Address
Ping Lab Computer
Open Socket
END: Initialize IOT Module

Figure 47 Flow chart for function NCSU_Config in init_IOT.c

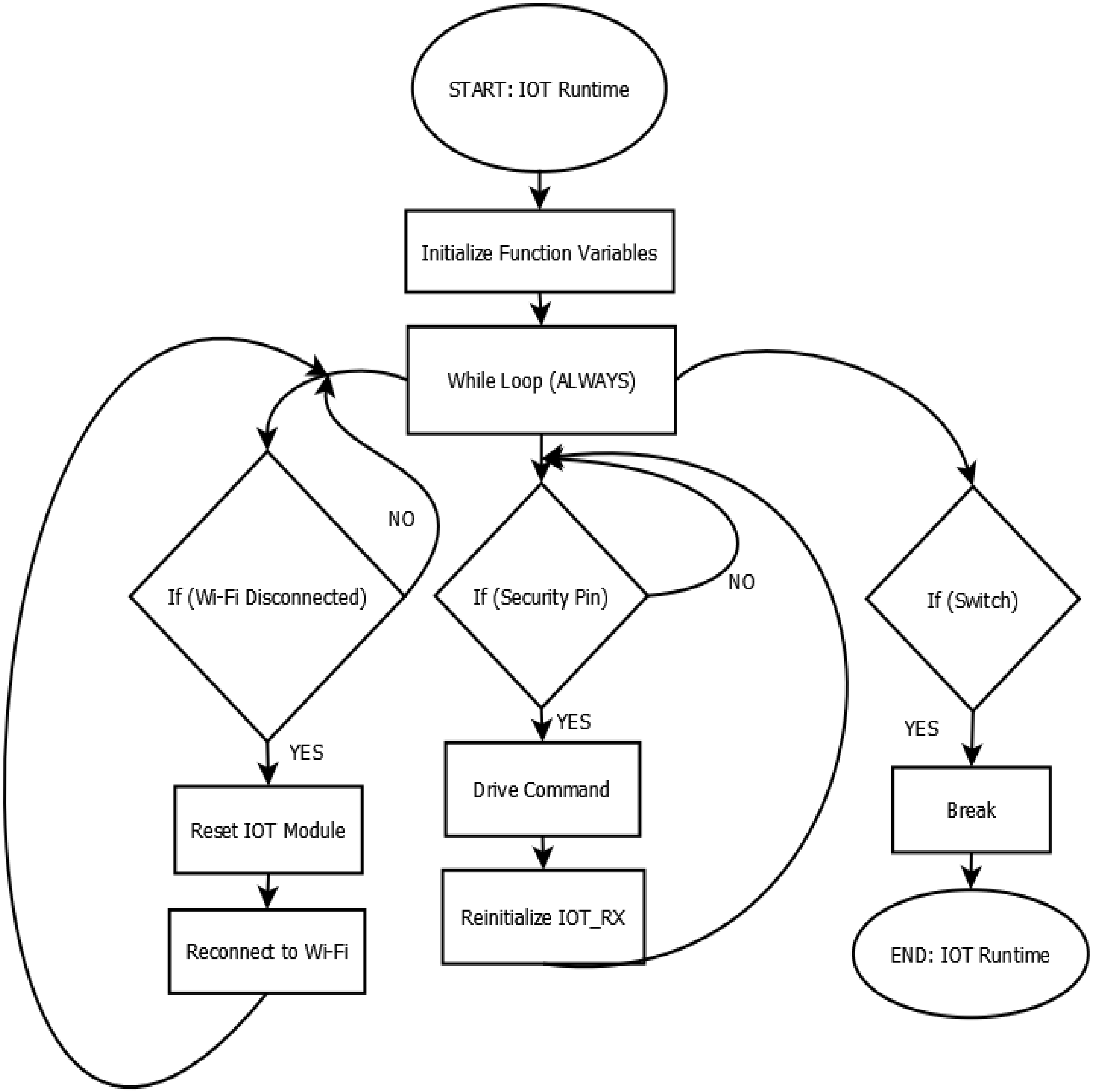


Figure 48 Flow chart for function Runtime_IOT in runtime_IOT.c

9. Software Listing

This section includes the source files used on the FRAM Experimenter board. This code is utilized to implement the function of the vehicle and its several different modes of operations. See different source code sections with details below.

9.1. main.c

```
//******************************************************************************
//
//  Description: This file contains the Main Routine - "While" Operating System
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

// Initialize Global Variables from macros.h
volatile unsigned char control_state[CNTL_STATE_INDEX];
volatile unsigned int Time_Sequence;
char led_smclk;
volatile char one_time;
extern char display_line_1[display_line_array];
extern char display_line_2[display_line_array];
extern char display_line_3[display_line_array];
extern char display_line_4[display_line_array];
extern char *display_1;
extern char *display_2;
extern char *display_3;
extern char *display_4;
char posL1;
char posL2;
char posL3;
char posL4;
char size_count;
char big;
char switch_select;
volatile unsigned int A0_time_count;
volatile unsigned int ADC_Right_Detector;
volatile unsigned int ADC_Left_Detector;
volatile unsigned int ADC_Thumb;
volatile unsigned int ADC_Temp;
volatile unsigned int ADC_Bat;
volatile unsigned int ADC_Channel;
_Bool adc_RD_Enable;
_Bool adc_LD_Enable;
_Bool thumbEnable;
_Bool tempEnable;
_Bool batEnable;
unsigned int blackLineLeft;
unsigned int blackLineRight;
volatile unsigned int rxRead;
volatile unsigned int IOTRead;
volatile char RX_Char[SMALL_RING_SIZE];
volatile char IOT_RX[SMALL_RING_SIZE];
volatile _Bool receiveEnable;
volatile _Bool baudToggle;
volatile _Bool IOT_Runtime;

void main(void){

//*********************************************************************************
//
//  Description: This file contains the Main Routine - "While" Operating System
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Global: Time_Sequence, switch_select, display_1, display_2, display_3,
// display_4,  posL1, posL2, posL3, posL4, big, switch_select, A0_time_count,
// ADC_Out, rxRead, rxWrite, txRead, txWrite, RX_Char[SMALL_RING_SIZE],
// TX_Char[SMALL_RING_SIZE], transmitEnable, receiveEnable
//
// Passed:
//
// Local: temp
//
```

```
// Return:  VOID
//
//****************************************************************************

  Init_Ports();                         // Initialize Ports
  Init_Clocks();                        // Initialize Clock System
  Init_Conditions();                    // Sets initial conditions for the LCD
  Time_Sequence = INITIAL;              // Sets Time_Sequence to 0
  Init_Timers();                        // Initialize Timers
  Five_msec_Delay(LIL_DELAY);           // Delay for the clock to settle
  Init_LCD();                           // Initialize LCD
  Init_ADC();                           // Initialize ADC
  Init_Serial_UCA0(TRUE);               // Initialize USCI-Baud Rate 9600
  Init_Serial_UCA1(TRUE);               // Initialize USCI-Baud Rate 9600
  // Initialize variables
  receiveEnable = FALSE;
  IOT_Runtime = FALSE;
  blackLineLeft = ADC_Left_Detector;
  blackLineRight = ADC_Right_Detector;
  switch_select = INITIAL;
  A0_time_count = INITIAL;
  int i = INITIAL;
  // Initial Display Set
  display_1 = "  Team    ";
  posL1 = LINE_POS_L0;
  display_2 = "Project 08";
  posL2 = LINE_POS_L0;
  display_3 = "  Two     ";
  posL3 = LINE_POS_L0;
  display_4 = "          ";
  posL4 = LINE_POS_L0;
  big = TRUE;
  lcd_BIG_mid();
  Display_Process();
  // Wait for SW1 or SW2
  while (ALWAYS) {
   if (!(P4IN & SW1)) {
    Five_msec_Delay(LIL_SW_DELAY);
```

```
      break;
    }
    if (!(P4IN & SW2)) {
      Five_msec_Delay(LIL_SW_DELAY);
      break;
    }
  }
  // Initial Menu
  display_1 = "          ";
  posL1 = LINE_POS_L0;
  display_2 = "          ";
  posL2 = LINE_POS_L0;
  display_3 = "          ";
  posL3 = LINE_POS_L0;
  display_4 = "          ";
  posL4 = LINE_POS_L0;
  big = TRUE;
  lcd_BIG_mid();
  Display_Process();

//------------------------------------------------------------------------------
// Beginning of the "While" Operating System
//------------------------------------------------------------------------------
  while(ALWAYS) {                         // Can the Operating system run
    switch(Time_Sequence){
    case CASE250:
      if(one_time){
        one_time = INITIAL;
      }
      Time_Sequence = INITIAL;            // Resets clock
    case CASE200:                         // 1000 msec
      if(one_time){
        one_time = INITIAL;
      }
    case CASE150:                         // 750 msec
      if(one_time){
        one_time = INITIAL;
      }
```

```
case CASE100:                                   // 500 msec
  if(one_time){
    one_time = INITIAL;
  }
case  CASE50:                                   // 250 msec
  if(one_time){
    one_time = INITIAL;
  }
  break;
default: break;
}
// Update all ADC channels
for(i = SW_SEL_5; i > INITIAL; i--) {
  ADC10_Process();
}
// START MAIN MENU//
// ITEM ONE
if ((ADC_Thumb > ITEM_ONE) && (ADC_Thumb <= ITEM_TWO)) {
  Display_Config();
}
// ITEM TWO
else if ((ADC_Thumb > ITEM_TWO) && (ADC_Thumb <= ITEM_THREE)) {
  Display_IOTCMD();
}
// ITEM THREE
else if ((ADC_Thumb > ITEM_THREE) && (ADC_Thumb <= ITEM_FOUR)) {
  Display_Backdoor();
}
// ITEM FOUR
else if ((ADC_Thumb > ITEM_FOUR) && (ADC_Thumb <= ITEM_FIVE)) {
  Display_Detect();
}
// ITEM FIVE
else if ((ADC_Thumb > ITEM_FIVE) && (ADC_Thumb <= ITEM_SIX)) {
  Display_Calibrate();
}
// ITEM SIX
else if ((ADC_Thumb > ITEM_SIX) && (ADC_Thumb <= ITEM_SEVEN)) {
      Display_Config();
    }
    // ITEM SEVEN
    else if ((ADC_Thumb > ITEM_SEVEN) && (ADC_Thumb <= ITEM_EIGHT)) {
      Display_IOTCMD();
    }
    // ITEM EIGHT
    else if ((ADC_Thumb > ITEM_EIGHT) && (ADC_Thumb <= ITEM_NINE)) {
      Display_Backdoor();
    }
    // ITEM NINE
    else if ((ADC_Thumb > ITEM_NINE) && (ADC_Thumb <= ITEM_TEN)) {
      Display_Detect();
    }
    // ITEM TEN
    else if ((ADC_Thumb > ITEM_TEN) && (ADC_Thumb <= ITEM_END)) {
      Display_Calibrate();
    }
    else {
      Display_Config();
    }
    // Update LCD Display
    Display_Process();
    Five_msec_Delay(LIL_DELAY);
    // Process switches
    Switches_Process();
    // Fail-safe for clock
    if(Time_Sequence > CASE250){
      Time_Sequence = INITIAL;
    }
    // Fail-safe for A0_time_count
    if (A0_time_count > FIVE_SEC_COUNT) {
      A0_time_count = INITIAL;
    }
  }
}
```

9.2. ports.c

```
//******************************************************************************
//
//  Description: This file contains the Port Routines
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

// Call port initialization for ports 1, 2, 3, 4, and J
void Init_Ports(void){
//******************************************************************************
//
//  Description: Initializes Ports
//
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed: USE_R_FORWARD
// Local:
// Return: VOID
//******************************************************************************

  Init_Port1();
  Init_Port2();
  Init_Port3(USE_R_FORWARD);
  Init_Port4();
  Init_PortJ();
}
```

```
void Init_Port1(void) {
//******************************************************************************
//
//  Description: Initializes Port 1
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed:
// Local:
// Return: VOID
//******************************************************************************

  // Configure PORT 1
  // V_DETECT_R          (0x01)
  // V_DETECT_L          (0x02)
  // IR_LED              (0x04)
  // V_THUMB             (0x08)
  // SPI_CS_LCD          (0x10)
  // SAO_LCD             (0x20)
  // SPI_SIMO            (0x40)
  // SPI_SOMI            (0x80)

  // Set Port 1 Registers
  P1SEL0 = CLEAR_REGISTER;
  P1SEL1 = CLEAR_REGISTER;
  P1DIR = CLEAR_REGISTER;
  // V_DETECT_R Initialization
  P1SEL0 |= V_DETECT_R;          // P1SEL0 ->  1
  P1SEL1 |= V_DETECT_R;          // P1SEL1 ->  1
  P1OUT &= ~V_DETECT_R;          // P1OUT ->   0
  P1DIR &= ~V_DETECT_R;          // P1DIR ->   0
  // V_DETECT_L Initialization
  P1SEL0 |= V_DETECT_L;          // P1SEL0 ->  1
  P1SEL1 |= V_DETECT_L;          // P1SEL1 ->  1
  P1OUT &= ~V_DETECT_L;              // P1OUT ->    0
  P1DIR &= ~V_DETECT_L;              // P1DIR ->    0
  // IR_LED Initialization
  P1SEL0 &= ~IR_LED;                 // P1SEL0 ->   0
  P1SEL1 &= ~IR_LED;                 // P1SEL1 ->   0
  P1OUT &= ~IR_LED;                  // P1OUT ->   0
  P1DIR |= IR_LED;                   // P1DIR ->    1
  // V_THUMB Initialization
  P1SEL0 |= V_THUMB;                 // P1SEL0 ->   1
  P1SEL1 |= V_THUMB;                 // P1SEL1 ->   1
  P1OUT &= ~V_THUMB;                 // P1OUT ->    0
  P1DIR &= ~V_THUMB;                 // P1DIR ->    0
  // SPI_CS_LCD Initialization
  P1SEL0 &= ~SPI_CS_LCD;             // P1SEL0 ->   0
  P1SEL1 &= ~SPI_CS_LCD;             // P1SEL1 ->   0
  P1OUT |= SPI_CS_LCD;               // P1OUT ->    1
  P1DIR |= SPI_CS_LCD;               // P1DIR ->    1
  // SA0_LCD Initialization
  P1SEL0 &= ~SA0_LCD;                // P1SEL0 ->   0
  P1SEL1 &= ~SA0_LCD;                // P1SEL1 ->   0
  P1OUT &=  ~SA0_LCD;                // P1OUT ->    0
  P1DIR |= SA0_LCD;                  // P1DIR ->    1
  // SPI_SIMO Initialization
  P1SEL0 &= ~SPI_SIMO;               // P1SEL0 ->   0
  P1SEL1 |= SPI_SIMO;                // P1SEL1 ->   1
  P1OUT &= ~SPI_SIMO;                // P1OUT ->    0
  P1DIR |= SPI_SIMO;                 // P1DIR ->    1
  // SPI_SOMI Initialization
  P1SEL0 &= ~SPI_SOMI;               // P1SEL0 ->   0
  P1SEL1 |= SPI_SOMI;                // P1SEL1 ->   1
  P1OUT &= ~SPI_SOMI;                // P1OUT ->    0
  P1DIR &= ~SPI_SOMI;                // P1DIR ->    0
  P1REN |= SPI_SOMI;                 // P1REN ->    1
}

void Init_Port2(void) {
//******************************************************************************
//
```

```
//  Description: Initializes Port 2
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed:
// Local:
// Return: VOID
//******************************************************************************

  // Configure PORT2
  // USB_TXD
  // USB_RXD
  // SPI_SCK
  // P2PIN3
  // P2PIN4
  // CPU_TXD
  // CPU_RXD
  // P2PIN7

  // Set Port 2 Registers
  P2SEL0 = CLEAR_REGISTER;
  P2SEL1 = CLEAR_REGISTER;
  P2OUT = CLEAR_REGISTER;
  P2DIR = CLEAR_REGISTER;
  // USB_TXD Initialization
  P2SEL0 &= ~USB_TXD;               // P2SEL0 ->   0
  P2SEL1 |= USB_TXD;                // P2SEL1 ->   1
  P2OUT &= ~USB_TXD;                // P2OUT ->    0
  P2DIR |= USB_TXD;                 // P2DIR ->    1
  // USB_RXD Initialization
  P2SEL0 &= ~USB_RXD;               // P2SEL0 ->   0
  P2SEL1 |= USB_RXD;                // P2SEL1 ->   1
  P2OUT &= ~USB_RXD;                // P2OUT ->    0
  P2DIR  &= ~USB_RXD;               // P2DIR ->    0
  // SPI_SCK Initialization
  P2SEL0 &= ~SPI_SCK;                 // P2SEL0 ->   0
  P2SEL1 |= SPI_SCK;                  // P2SEL1 ->   1
  P2OUT |= SPI_SCK;                   // P2OUT ->    1
  P2DIR |= SPI_SCK;                   // P2DIR ->    1
  // P2PIN3 Initialization
  P2SEL0 &= ~P2PIN3;                  // P2SEL0 ->   0
  P2SEL1 &= ~P2PIN3;                  // P2SEL1 ->   0
  P2OUT &= ~P2PIN3;                   // P2OUT ->    0
  P2DIR |= P2PIN3;                    // P2DIR ->    1
  P2REN &= ~P2PIN3;                   // P2REN ->    0
  // P2PIN4 Initialization
  P2SEL0 &= ~P2PIN4;                  // P2SEL0 ->   0
  P2SEL1 &= ~P2PIN4;                  // P2SEL1 ->   0
  P2OUT &= ~P2PIN4;                   // P2OUT ->    0
  P2DIR |= P2PIN4;                    // P2DIR ->    1
  P2REN &= ~P2PIN4;                   // P2REN ->    0
  // CPU_TXD Initialization
  P2SEL0 &= ~CPU_TXD;                 // P2SEL0 ->   0
  P2SEL1 |= CPU_TXD;                  // P2SEL1 ->   1
  P2OUT &= ~CPU_TXD;                  // P2OUT ->    0
  P2DIR |= CPU_TXD;                   // P2DIR ->    1
  // CPU_RXD Initialization
  P2SEL0 &= ~CPU_RXD;                 // P2SEL0 ->   0
  P2SEL1 |= CPU_RXD;                  // P2SEL1 ->   1
  P2OUT &= ~CPU_RXD;                  // P2OUT ->    0
  P2DIR  &= ~CPU_RXD;                 // P2DIR ->    0
  // P2PIN7 Initialization
  P2SEL0 &= ~P2PIN7;                  // P2SEL0 ->   0
  P2SEL1 &= ~P2PIN7;                  // P2SEL1 ->   0
  P2OUT &= ~P2PIN7;                   // P2OUT ->    0
  P2DIR |= P2PIN7;                    // P2DIR ->    1
  P2REN &= ~P2PIN7;                   // P2REN ->    0
}

void Init_Port3(char clock_state) {
//******************************************************************************
//
//  Description: Initializes Port 3
```

```
//
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed: clock_state
// Local:
// Return: VOID
//******************************************************************************

  // Configure PORT 3
  // X
  // Y
  // Z
  // LCD_BACKLITE
  // R_FORWARD
  // R_REVERSE
  // L_FORWARD
  // L_REVERSE

  // Set Port 3 Registers
  P3SEL0 = CLEAR_REGISTER;
  P3SEL1 = CLEAR_REGISTER;
  P3OUT = CLEAR_REGISTER;
  P3DIR = CLEAR_REGISTER;
  // P3X Initialization
  P3SEL0 &= ~P3X;                    // P3SEL0 ->   0
  P3SEL1 &= ~P3X;                    // P3SEL1 ->   0
  P3OUT &= ~P3X;                     // P3OUT ->    0
  P3DIR &= ~P3X;                     // P3DIR ->    0
  P3REN &= ~P3X;                     // P3REN ->    0
  // P3Y Initialization
  P3SEL0 &= ~P3Y;                    // P3SEL0 ->   0
  P3SEL1 &= ~P3Y;                    // P3SEL1 ->   0
  P3OUT &= ~P3Y;                     // P3OUT ->    0
  P3DIR &= ~P3Y;                     // P3DIR ->    0
```

```
P3REN &= ~P3Y;                          // P3REN ->     0
// Z Initialization
P3SEL0 &= ~P3Z;                         // P3SEL0 ->    0
P3SEL1 &= ~P3Z;                         // P3SEL1 ->    0
P3OUT &= ~P3Z;                          // P3OUT ->     0
P3DIR &= ~P3Z;                          // P3DIR ->     0
P3REN &= ~P3Z;                          // P3REN ->     0
// LCD_BACKLITE Initialization
P3SEL0 &= ~LCD_BACKLITE;                // P3SEL0 ->    0
P3SEL1 &= ~LCD_BACKLITE;               // P3SEL1 ->    0
P3OUT |= LCD_BACKLITE;                  // P3OUT ->     1
P3DIR |= LCD_BACKLITE;                  // P3DIR ->     1
// Choose between R_FORWARD & SMCLK
if (clock_state == USE_R_FORWARD) {
  // R_FORWARD
  P3SEL0 &= ~R_FORWARD;
  P3SEL1 &= ~R_FORWARD;
  P3OUT &= ~R_FORWARD;
  P3DIR |= R_FORWARD;
}
else {
  P3SEL0 |= SET_SMCLK;
  P3SEL1 |= SET_SMCLK;
  P3OUT |= SET_SMCLK;
  P3DIR |= SET_SMCLK;
}
// R_REVERSE
P3SEL0 &= ~R_REVERSE;
P3SEL1 &= ~R_REVERSE;
P3OUT &= ~R_REVERSE;
P3DIR |= R_REVERSE;
// L_FORWARD
P3SEL0 &= ~L_FORWARD;
P3SEL1 &= ~L_FORWARD;
P3OUT &= ~L_FORWARD;
P3DIR |= L_FORWARD;
// L_REVERSE
P3SEL0 &= ~L_REVERSE;
```

```
  P3SEL1 &= ~L_REVERSE;
  P3OUT &= ~L_REVERSE;
  P3DIR |= L_REVERSE;
}

void Init_Port4(void) {
//******************************************************************************
//
//  Description: Initializes Port 4
//
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed:
// Local:
// Return: VOID
//******************************************************************************

  // Configure PORT 4
  // SW1            (0x01)
  // SW2            (0x02)

  // Set Port 4 Registers
  P4SEL0 = CLEAR_REGISTER;
  P4SEL1 = CLEAR_REGISTER;
  P4OUT = CLEAR_REGISTER;
  P4DIR = CLEAR_REGISTER;
  // SW1 Initialization
  P4SEL0 &= ~SW1;              // SW1 set as I/O
  P4SEL1 &= ~SW1;              // SW1 set as I/O
  P4OUT |= SW1;                // Configures SW1 for pullup resistor operation
  P4DIR &= ~SW1;               // Direction set as input
  P4REN |= SW1;                // Enables Pull-up Resistor
  // SW2 Initialization
  P4SEL0 &= ~SW2;              // SW2 set as I/O
  P4SEL1 &= ~SW2;              // SW2 set as I/O
  P4OUT |= SW2;                // Configures SW1 for pullup resistor operation
  P4DIR &= ~SW2;               // Direction set as input
  P4REN |= SW2;                // Enables Pull-up Resistor
}

void Init_PortJ(void) {
//******************************************************************************
//
//  Description: Initializes Port J
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
// Passed:
// Local:
// Return: VOID
//******************************************************************************

  // Configure PORT J
  // IOT_FACTORY
  // IOT_WAKEUP
  // IOT_STA_MINIAP
  // RESET

  // Set Port J Registers
  PJSEL0 = CLEAR_REGISTER;
  PJSEL1 = CLEAR_REGISTER;
  PJOUT = CLEAR_REGISTER;
  PJDIR = CLEAR_REGISTER;
  // Initialize IOT_FACTORY
  PJSEL0 &= ~IOT_FACTORY;
  PJSEL1 &= ~IOT_FACTORY;
  PJOUT &= ~IOT_FACTORY;
  PJDIR |= IOT_FACTORY;
  // Initialize IOT_WAKEUP
  PJSEL0 &= ~IOT_WAKEUP;
  PJSEL1 &= ~IOT_WAKEUP;
  PJOUT |= IOT_WAKEUP;
  PJDIR |= IOT_WAKEUP;
  // Initialize IOT_STA_MINIAP
  PJSEL0 &= ~IOT_STA_MINIAP;
  PJSEL1 &= ~IOT_STA_MINIAP;
  PJOUT |= IOT_STA_MINIAP;
  PJDIR |= IOT_STA_MINIAP;
  // Initialize RESET
  PJSEL0 &= ~RESET;
  PJSEL1 &= ~RESET;
  PJOUT &= ~RESET;
  PJDIR |= RESET;
  // Initialize XINR & XOUTR
  PJSEL0 |= XINR;
  PJSEL0 |= XOUTR;
}
```

9.3. timers.c

```
//******************************************************************************
//
//  Description: This file contains initialize calls for different system
//  timers
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Init_Timers(void){
//******************************************************************************
//
//  Description: This file contains the Timer Routine
```

This document contains information that is **PRIVILEGED** and **CONFIDENTIAL**; you are hereby notified that any dissemination of this information is strictly prohibited.

```
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
//
// Passed:
//
// Local:
//
// Return: VOID
//
//*****************************************************************************

  Init_Timer_A0();      // Enable Timer A0
  //  Init_Timer_A1();  //
  //  Init_Timer_B0();  //
  //  Init_Timer_B1();  //
  Init_Timer_B2();      //  Required for provided compiled code to work
}
```

9.4. timersA0.c

```
//*****************************************************************************
//
//  Description: This file is used to initialize the A0 Timer
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//*****************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Init_Timer_A0(void){
//*****************************************************************************
//
//  Description: Initializes the A0 Timer
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals:
//
// Passed:
//
// Local:
//
// Return: VOID
//
//******************************************************************************

  TA0CTL = TASSEL__SMCLK;                // SMCLK source
  TA0CTL |= TACLR;                       // Resets TA0R, clock divider, count direction
  TA0CTL |= MC__CONTINOUS;               // Sets timer to count up continuously and then reset to zero
  TA0CTL |= ID__8;                       // Divide the clock by 8
  TA0CTL &= ~TAIE;                       // Disable overflow interrupt
  TA0CTL &= ~TAIFG;                      // Clear overflow interrupt flag
  TA0EX0 = TAIDEX_7;                     // Divide the clock by an additional 8
  TA0CCR0 = TA0CCR0_INTERVAL;            // CCR0
  TA0CCTL0 |= CCIE;                      // CCR0 enable interrupt
  // TA0CCR1 = TA0CCR1_INTERVAL          // CCR1
  // TA0CCTL1 |= CCIE;                   // CCR1 enable interrupt
  // TA0CCR2 = TA0CCR2_INTERVAL          // CCR2
  // TA0CCTL2 |= CCIE;                   // CCR2 enable interrupt
}
```

9.5. timer_interrupt.c

```
//******************************************************************************
//
//  Description: Handles interrupts from the A0 Timer
//
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//*****************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void){
//*****************************************************************************
//
//  Description: Handles interrupts from the A0 Timer
//
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: A0_time_count
//
// Passed:
//
// Local:
//
// Return: VOID
//
//*****************************************************************************

  if (A0_time_count >= TEN_SEC_COUNT) {
    A0_time_count = INITIAL;                   // Reset A0 Time Count
  }
  // Handle A0 Logic
  A0_time_count++;                             // Increment A0 Time Counter
  TA0CCR0 += TA0CCR0_INTERVAL;                 // Increment CCR0 Register
}
```

9.6. adc.c

```
//******************************************************************************
//
//  Description: This file contains the ADC Routines
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Init_ADC(void){
  //******************************************************************************
  //
  //  Description: This function initializes the ADC system
  //
  //  Team 2
  //  Nov 2015
  //  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
  //
  // Globals:
  //
  // Passed:
  //
  // Local:
  //
  // Return: VOID
  //
  //******************************************************************************

  ADC10CTL0 = CLEAR_REGISTER;            // Clear ADC10CTL0
  ADC10CTL0 |= ADC10SHT_2;               // 16 ADC clocks
  ADC10CTL0 &= ~ADC10MSC;                // Single Sequence
  ADC10CTL0 |= ADC10ON;                  // ADC ON - Core Enabled
  ADC10CTL1 = CLEAR_REGISTER;             // Clear ADC10CTL1
  ADC10CTL1 |= ADC10SHS_0;                // ADC10SC bit
  ADC10CTL1 |= ADC10SHP;                  // SAMPCON signal sourced from sampling timer
  ADC10CTL1 &= ~ADC10ISSH;                // The sample-input signal is not inverted
  ADC10CTL1 |= ADC10DIV_0;                // ADC10_B clock divider - divide by 1
  ADC10CTL1 |= ADC10SSEL_0;               // MODCLK
  ADC10CTL1 |= ADC10CONSEQ_0;             // Single-channel, single-conversion

  ADC10CTL2 = CLEAR_REGISTER;             // Clear ADC10CTL2
  ADC10CTL2 |= ADC10DIV_0;                // Pre-divide by 1
  ADC10CTL2 |= ADC10RES;                  // 10-bit resolution
  ADC10CTL2 &= ~ADC10DF;                  // Binary unsigned
  ADC10CTL2 &= ~ADC10SR;                  // Supports up to approximately 200 ksps

  ADC10MCTL0 = CLEAR_REGISTER;            // Clear ADC10MCTL0
  ADC10MCTL0 |= ADC10SREF_0;              // V(R+) = AVCC and V(R-) = AVSS
  ADC10MCTL0 |= ADC10INCH_0;              // Channel A0
  ADC10IE |= ADC10IE0;                    // Enable ADC conversion complete interrupt
}

#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void){
  //******************************************************************************
  //
  //  Description: This function contains the ADC interrupt service routine
  //  handler code that addresses different cases of the interrupt.
  //
  //  Team 2
  //  Nov 2015
  //  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
  //
  // Globals: ADC_Thumb, ADC_Right_Detector, ADC_Left_Detector,
  // ADC_Temp, ADC_Bat, ADC_Channel
  //
  // Passed:
  //
  // Local:
```

```
//
// Return: VOID
//
//****************************************************************************

switch(__even_in_range(ADC10IV,SW_SEL_12)) {
case SW_SEL_0: break;
case SW_SEL_2: break;
case SW_SEL_4: break;
case SW_SEL_6: break;
case SW_SEL_8: break;
case SW_SEL_10: break;
case SW_SEL_12:
  // Need this to change the ADC10INCH_x value.
  ADC10CTL0 &= ~ADC10ENC;                    // Toggle ENC bit.

  switch (ADC_Channel++){
    // Right Detector
  case Right_Detector:
    ADC10MCTL0 = ADC10INCH_1;                // Next channel
    ADC_Left_Detector = ADC10MEM0;           // Read Channel
    break;
    // Left Detector
  case Left_Detector:
    ADC10MCTL0 = ADC10INCH_3;                // Next channel
    ADC_Right_Detector = ADC10MEM0;          // Read Channel
    break;
    // Thumbwheel
  case Thumbwheel:
    ADC10MCTL0 = ADC10INCH_11;               // Next channel
    ADC_Thumb = ADC10MEM0;                   // Read Channel
    break;
    // Temperature
  case CHANNEL_A10:
    ADC10MCTL0 = ADC10INCH_10;               // Next channel
    ADC_Temp = ADC10MEM0;                    // Read Channel
    break;
    // Internal
    case CHANNEL_A11:
      ADC10MCTL0 = ADC10INCH_0;              // Next channel
      ADC_Bat = ADC10MEM0;                   // Read Channel
      ADC_Channel=INITIAL;
      break;
    default:
      break;
    }
    break;
  default: break;
  }
}

void ADC10_Process(void){
  //*****************************************************************************
  //
  //  Description: This function contains the ADC process routine that triggers
  //  the update of the ADC values
  //
  //  Team 2
  //  Nov 2015
  //  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
  //
  // Globals:
  //
  // Passed:
  //
  // Local:
  //
  // Return: VOID
  //
  //*****************************************************************************

  while (ADC10CTL1 & BUSY);                  // Wait if ADC10 core is active
  ADC10CTL0 |= ADC10ENC + ADC10SC;           // Sampling and conversion start
}
```

9.7. init_serial.c

```
//*****************************************************************************
```

```
//
//  Description: Initializes serialized communications
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Init_Serial_UCA0(_Bool lowBaud) {
//******************************************************************************
//
//  Description: Initializes serialized communications
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: rxRead, rxWrite, txRead, txWrite, RX_Char[WORD_LENGTH],
// TX_Char[WORD_LENGTH],
//
// Passed: lowBaud
//
// Local: i
//
// Return: VOID
//
//******************************************************************************

  // Configure UART 0
  UCA0CTLW0 = INITIAL;                    // Use Word register
  UCA0CTLW0 |= UCSSEL__SMCLK;
  UCA0CTL1 |= UCSWRST;                    // Set Software reset enable
  // Set Baudrate
  if (lowBaud)
  {
    UCA0BRW = BRW9600;
    UCA0MCTLW = BAUD_9600;            // Set Baud Rate to 9600
  }
  else
  {
    UCA0BRW = BRW115200;
    UCA0MCTLW = BAUD_115200;          // Set Baud Rate to 115200
  }
  // USB_TXD Initialization
  P2SEL0 &= ~USB_TXD;                 // P2SEL0 ->   0
  P2SEL1 |= USB_TXD;                  // P2SEL1 ->   1
  P2OUT|= USB_TXD;                    // P2OUT ->    1
  P2DIR |= USB_TXD;                   // P2DIR ->    1
  // USB_RXD Initialization
  P2SEL0 &= ~USB_RXD;                 // P2SEL0 ->   0
  P2SEL1 |= USB_RXD;                  // P2SEL1 ->   1
  P2OUT &= ~USB_RXD;                  // P2OUT ->    0
  P2DIR  &= ~USB_RXD;                 // P2DIR ->    0
  // CPU_TXD Initialization
  P2SEL0 &= ~CPU_TXD;                 // P2SEL0 ->   0
  P2SEL1 |= CPU_TXD;                  // P2SEL1 ->   1
  P2OUT |= CPU_TXD;                   // P2OUT ->    1
  P2DIR |= CPU_TXD;                   // P2DIR ->    1
  // CPU_RXD Initialization
  P2SEL0 &= ~CPU_RXD;                 // P2SEL0 ->   0
  P2SEL1 |= CPU_RXD;                  // P2SEL1 ->   1
  P2OUT &= ~CPU_RXD;                  // P2OUT ->    0
  P2DIR  &= ~CPU_RXD;                 // P2DIR ->    0
  // Release from reset
  UCA0CTLW0 &= ~UCSWRST;              // Release from reset
  // Hande interrupts
  UCA0IE |= UCTXIE;                   // Enable TX interrupt
  UCA0IE |= UCRXIE;                   // Enable RX interrupt
}

void Init_Serial_UCA1(_Bool lowBaud) {
```

```
//****************************************************************************
//
//  Description: Initializes serialized communications
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: rxRead, rxWrite, txRead, txWrite, RX_Char[WORD_LENGTH],
// TX_Char[WORD_LENGTH],
//
// Passed: lowBaud
//
// Local: i
//
// Return: VOID
//
//****************************************************************************

  // Count
  int i;
  // Initial RX
  for (i=INITIAL; i<SMALL_RING_SIZE; i++) {
    RX_Char[i] = INITIALIZE_CHAR;
  }
  rxRead = INITIAL;
  // Configure UART 0
  UCA1CTLW0 = INITIAL;                        // Use Word register
  UCA1CTLW0 |= UCSSEL__SMCLK;
  UCA1CTL1 |= UCSWRST;                        // Set Software reset enable

  // Optional Initialization Controls
  //UCA1CTLW0 |= UCPEN;
  //UCA1CTLW0 &= ~UCPAR;
  //UCA1CTLW0 &= ~UCMSB;
  //UCA1CTLW0 &= ~UC7BIT;
  //UCA1CTLW0 &= ~UCSPB;
  //UCA1CTLW0 &= ~UCMODE0;
```

```
//UCA1CTLW0 &= ~UCMODE1;
//UCA1CTLW0 &= ~UCSYNC;
//UCA1CTLW0 |= UCSSEL0;
//UCA1CTLW0 |= UCSSEL1;
//UCA1CTLW0 &= ~UCRXEIE;
//UCA1CTLW0 &= ~UCBRKIE;
//UCA1CTLW0 &= ~UCDORM;
//UCA1CTLW0 &= ~UCTXADDR;
//UCA1CTLW0 &= ~UCTXBRK;
//UCA1CTLW1 &= ~UCGLIT0;
//UCA1CTLW1 &= ~UCGLIT1;

// Set Baudrate
if (lowBaud)
{
  UCA1BRW = BRW9600;
  UCA1MCTLW = BAUD_9600;           // Set Baud Rate to 9600
}
else
{
  UCA1BRW = BRW115200;
  UCA1MCTLW = BAUD_115200;         // Set Baud Rate to 115200
}
// USB_TXD Initialization
P2SEL0 &= ~USB_TXD;                // P2SEL0 ->   0
P2SEL1 |= USB_TXD;                 // P2SEL1 ->   1
P2OUT|= USB_TXD;                   // P2OUT ->    1
P2DIR |= USB_TXD;                  // P2DIR ->    1
// USB_RXD Initialization
P2SEL0 &= ~USB_RXD;                // P2SEL0 ->   0
P2SEL1 |= USB_RXD;                 // P2SEL1 ->   1
P2OUT &= ~USB_RXD;                 // P2OUT ->    0
P2DIR  &= ~USB_RXD;                // P2DIR ->    0
// CPU_TXD Initialization
P2SEL0 &= ~CPU_TXD;                // P2SEL0 ->   0
P2SEL1 |= CPU_TXD;                 // P2SEL1 ->   1
P2OUT |= CPU_TXD;                  // P2OUT ->     1
P2DIR |= CPU_TXD;                  // P2DIR ->    1
```

```
  // CPU_RXD Initialization
  P2SEL0 &= ~CPU_RXD;                  // P2SEL0 ->    0
  P2SEL1 |= CPU_RXD;                   // P2SEL1 ->    1
  P2OUT &= ~CPU_RXD;                   // P2OUT ->     0
  P2DIR  &= ~CPU_RXD;                  // P2DIR ->     0
  // Release from reset
  UCA1CTLW0 &= ~UCSWRST;               // Release from reset
  // Hande interrupts
  UCA1IE &= ~UCTXIE;                   // Disable TX interrupt
  UCA1IE &= ~UCRXIE;                   // Disable RX interrupt
}
```

9.8. serial.c

```
//******************************************************************************
//
//  Description: This file is used to implement the test code for serial
//  communications.
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Project_6(void){
//******************************************************************************
//
//  Description: This file is used to implement the test code for serial
//  communications.
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: RX_Char
```

```
//
// Passed:
//
// Local: packet1, packet2, RX_Char[LINE_POS_L0], RX_Char[LINE_POS_L1],
// tmpPacket, count
//
// Return: VOID
//
//****************************************************************************

// Configure UART with 9600 Baud
Init_Serial_UCA1(TRUE);
// Initialize display for counter
display_1 = "----------";
posL1 = LINE_POS_L0;
display_2 = " Counter: ";
posL2 = LINE_POS_L0;
display_3 = "          ";
posL3 = LINE_POS_L0;
display_4 =  "----------";
posL4 = LINE_POS_L0;
Display_Process();
// Initialize variables
char packet1 = INITIAL;
char packet2 = INITIAL;
RX_Char[LINE_POS_L0] = INITIAL;
RX_Char[LINE_POS_L1] = INITIAL;
unsigned int tmpPacket = INITIAL;
unsigned int count = INITIAL;
// Begin sending/receiving packets
while (ALWAYS) {
        UCA1TXBUF = packet1;                // Send packet 1
        Five_msec_Delay(SW_SEL_1);          // Wait
        UCA1TXBUF = packet2;                // Send packet 2
        Five_msec_Delay(SW_SEL_1);          // Wait
        // Read new transmissions
        packet1 = RX_Char[LINE_POS_L0];
        packet2 = RX_Char[LINE_POS_L1];
```

```
    // Decipher packet and increment counter
    count = packet2;            // Open packet1 2
    count <<= SHIFT_8;          // Shift packet 2 to MSB
    count |= packet1;           // Open packet 1
    count++;                    // Increment counter
    hexToInt(count);            // Display counter to display
    // Prepare packets
    packet1 = (char) count;
    tmpPacket = count >> SHIFT_8;
    packet2 = (char) tmpPacket;
    // Send Packets
    UCA1TXBUF = packet1;
    Five_msec_Delay(SW_SEL_1);
    UCA1TXBUF = packet2;
    Five_msec_Delay(SW_SEL_1);
    // Reinitialize counter
    if (!(P4IN & SW1)){
        packet1 = INITIAL;
        packet2 = INITIAL;
        count = INITIAL;
        tmpPacket = INITIAL;
        RX_Char[LINE_POS_L0] = INITIAL;
        RX_Char[LINE_POS_L1] = INITIAL;
    }
    // Break out of communcation loop
    if (!(P4IN & SW2)){
        break;
    }
    // Update the display
    Display_Process();
    // Wait before repeating
    Five_msec_Delay(BIGGER_DELAY);
}

display_1 = "P. Six";
posL1 = LINE_POS_L2;
display_2 = "Complete!";
posL2 = LINE_POS_L1;
```

```
 display_3 = "          ";
 posL3 = LINE_POS_L0;
 display_4 =  "          ";
 posL4 = LINE_POS_L0;
 Display_Process();
}
```

9.9. serial_interrupt.c

```
//*****************************************************************************
//
//  Description: Handles eUSCI_A interrupts
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//*****************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void){
//*****************************************************************************
//
//  Description: Handles eUSCI A0 interrupts
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: rxRead RX_Char[WORD_LENGTH], receiveEnable
//
// Passed:
//
// Local: temp
//
// Return: VOID
```

```
//
//****************************************************************************

  unsigned int temp = INITIAL;
  switch(__even_in_range(UCA0IV, USCI_val8)) {
  case SW_SEL_0:                              //Vector 0 - No Interrupt
    break;
  case SW_SEL_2:                              // Vector 2 - RXIFG
    // Code for Receive
    temp = rxRead;
    receiveEnable = TRUE;                     // Tranmission received
    RX_Char[temp] =  UCA0RXBUF;               // Store Transmission
    if (++rxRead >= (SMALL_RING_SIZE)) {
      rxRead = INITIAL;                       // Reset Index
    }
    break;
  case SW_SEL_4:                              // Vector 4 - TXIFG
    // Code for Transmit
    break;
  default: break;
  }
}

#pragma vector = USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void){
//****************************************************************************
//
//  Description: Handles eUSCI A1 interrupts
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: rxRead IOT_RX[WORD_LENGTH], receiveEnable
//
// Passed:
//
// Local: temp
```

```
//
// Return: VOID
//
//******************************************************************************

  unsigned int temp = INITIAL;
  switch(__even_in_range(UCA1IV, USCI_val8)) {
  case SW_SEL_0:                                  //Vector 0 - No Interrupt
    break;
  case SW_SEL_2:                                  // Vector 2 - RXIFG
    // Code for Receive
    temp = IOTRead;
    IOT_RX[temp] =  UCA1RXBUF;                    // Store Transmission
    // Helps with IOT command space
    if (IOT_RX[temp] == '*') {
      IOTRead = INITIAL;
      IOT_RX[Pos0] = IOT_RX[temp];
    }
    if (IOT_RX[temp] == '+') {
      IOTRead = INITIAL;
      IOT_RX[Pos0] = IOT_RX[temp];
    }
    if (++IOTRead >= (SMALL_RING_SIZE)) {
      IOTRead = INITIAL;                          // Reset Index
    }
    break;
  case SW_SEL_4:                                  // Vector 4 - TXIFG
    // Code for Transmit
    break;
  default: break;
  }
}
```

9.10. init_IOT.c

```
//******************************************************************************
//
//  Description: This file is used to implement the IOT functions
//
//  Team 2
//  Oct 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void NCSU_Config(void){
//******************************************************************************
//
//  Description: This function is used to configure the IOT device on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], display_1, display_2, display_3,
// display_4
//
// Passed:
//
// Local: i, firstDOT, secondDOT
//
// Return: VOID
//
//******************************************************************************

  // Initialize variables
  int i = INITIAL;
  _Bool firstDOT = FALSE;
  // Format LCD screen
  Display_Format();
  // Handle interrupts
  UCA1IE |= UCTXIE;                    // Enable TX interrupt
  UCA1IE |= UCRXIE;                    // Enable RX interrupt
```

```
// CONFIGURE SSID
SSID();
// CONFIGURE HOSTNAME
HOST();
// CONFIGURE NETWORK PRIVACY MODE
NPM();
// CONFIGURE NETWORK MODE
NET_MODE();
// FLASH SETTING
FLASH();
//// RESET ////
IOT_RESET();
// LCD screen update
Display_Format();
display_2 = " COMPLETE ";
Display_Process();
Five_msec_Delay(ONE_SEC_MULT);
// LCD screen update
Display_Format();
display_2 = "CONNECTING";
Display_Process();
Five_msec_Delay(FIVE_SEC_MULT);
// GET SSID to Display
char SSIDcmd[Pos12] = "AT+S.SSIDTXT";
// Initialize IOT_RX ring
INIT_IOT_RX();
// Send command
for(i=INITIAL; i < Pos12; i++) {
  UCA1TXBUF = UCA0TXBUF = SSIDcmd[i];
  One_msec_Delay();
}
UCA1TXBUF = UCA0TXBUF = TX_FINISH;
Five_msec_Delay(BIG_DELAY + ONE_MSEC);
// Display SSID
display_1[Pos0] = display_1[Pos1] = display_1[Pos2] = ' ';
display_1[Pos3] = IOT_RX[Pos8];
display_1[Pos4] = IOT_RX[Pos9];
display_1[Pos5] = IOT_RX[Pos10];
```

```
display_1[Pos6] = IOT_RX[Pos11];
display_1[Pos7] = display_1[Pos8] = display_1[Pos9] = ' ';
display_2 = "  ipaddr  ";
display_3 = display_4 = "          ";
big = INITIAL;
lcd_4line();
Display_Process();
Five_msec_Delay(ONE_SEC_MULT);
// GET IP Address to display
char IPADDRcmd[Pos18] = "AT+S.STS=ip_ipaddr";
// Initialize IOT_RX ring
INIT_IOT_RX();
// Send command
for(i=INITIAL; i < Pos18; i++) {
  UCA1TXBUF = UCA0TXBUF = IPADDRcmd[i];
  One_msec_Delay();
}
UCA1TXBUF = UCA0TXBUF = TX_FINISH;
Five_msec_Delay(BIG_DELAY + ONE_MSEC);
// Disable Interrupt
UCA1IE &= ~UCRXIE;
// Handle first IP chunk xxx.xxx.
firstDOT = FALSE;
int indexIP = INITIAL;
for(i=Pos1; i < Pos9; i++) {
  // Update display with first IP chunk
  display_3[i] = IOT_RX[i+Pos14];
  // Look for second dot
  if ((IOT_RX[i+Pos14] == '.') && (firstDOT == TRUE)) {
    indexIP = i+Pos13;
    break;
  }
  // Look for first dot
  if (IOT_RX[i+Pos14] == '.') firstDOT = TRUE;
}
display_3[Pos0] = display_3[Pos9] = ' ';
// Display first IP chunk
Display_Process();
```

```
  Five_msec_Delay(BIG_DELAY);
  // Display second IP chunk xxx.xxx
  display_4 = "          ";
  for(i=Pos2; i < Pos9; i++) {
    // Look for IP end
    if (IOT_RX[i+indexIP] == '\n') {
      display_4[i-Pos1] = ' ';
      break;
    }
    // Update display with second IP chunk
    display_4[i] = IOT_RX[i+indexIP];
  }
  // Display IP Address
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  // Reenable interrupt
  UCA1IE |= UCRXIE;
  // PING LAB COMPUTER
  PING();
  Five_msec_Delay(BIGGER_DELAY);
  // OPEN SOCKET CONNECTION
  OPEN_SOCKET();
  CHECK_SOCKET();
  while(ALWAYS) {
    if (!(P4IN & SW2) || (IOT_Runtime)) {
      // Debounce
      Five_msec_Delay(LIL_SW_DELAY);
      // Clean LCD
      display_1[Pos0] = display_1[Pos1] = display_1[Pos2] = '-';
      display_1[Pos3] = display_1[Pos4] = display_1[Pos5] = '-';
      display_1[Pos6] = display_1[Pos7] = display_1[Pos8] = '-';
      display_1[Pos9] = '-';
      break;
    }
  }
}
```

9.11. command_IOT.c

```
//******************************************************************************
//
//  Description: This file is used to implement the IOT functions
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//******************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void IOT_RESET(void){
//******************************************************************************
//
//  Description: This function is used to reset the IOT device
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], display_2
// display_4
//
// Passed:
//
// Local:
//
// Return: VOID
//
//******************************************************************************

  display_2 = "  RESET   ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  // Drive Reset
  PJOUT |= RESET;
  // Wait
  Five_msec_Delay(LIL_DELAY);
  // Release Reset
  PJOUT &= ~RESET;
  // Re-initialize LCD
  Init_LCD();
  Five_msec_Delay(ONE_SEC_MULT);
}

void IOT_ACK(void){
//*****************************************************************************
//
//  Description: This function is used to determine when the IOT device has
//  sent back an acknowledgement
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE]
//
// Passed:
//
// Local:
//
// Return: VOID
//
//*****************************************************************************

  // Wait for IOT confirmation
  while (IOT_RX[Pos2] != 'O') {
    // Allow break
    if (!(P4IN & SW2)) {
      // Debounce
      Five_msec_Delay(LIL_SW_DELAY);
      break;
    }
  }
}

void INIT_IOT_RX(void){
//*****************************************************************************
//
//  Description: This function is used to reinitialize the IOT Read Ring
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead
//
// Passed:
//
// Local: i
//
// Return: VOID
//
//*****************************************************************************

  for(int i=INITIAL; i < SMALL_RING_SIZE; i++) IOT_RX[i] = INITIAL;
  IOTRead = INITIAL;
}

void SSID(void){
//*****************************************************************************
//
//  Description: This function is used to configure the SSID on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead, display_2
```

```
//
// Passed:
//
// Local: i, SSIDcmd
//
// Return: VOID
//
//******************************************************************************

  int i = INITIAL;
  //// SET SSID ////
  display_2 = " SET SSID ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  char SSIDcmd[Pos17] = "AT+S.SSIDTXT=ncsu";
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos17; i++) {
    UCA1TXBUF = UCA0TXBUF = SSIDcmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
  // Wait for IOT confirmation
  IOT_ACK();

  ////// GET SSID ////
  //  display_2 = " GET SSID ";
  //  Display_Process();
  //  Five_msec_Delay(BIG_DELAY);
  //
  //  // Initialize IOT_RX ring
  //  for(i=INITIAL; i < SMALL_RING_SIZE; i++) IOT_RX[i] = INITIAL;
  //
  //  IOTRead = INITIAL;
  //  for(i=INITIAL; i < Pos12; i++) {
  //    UCA1TXBUF = UCA0TXBUF = SSIDcmd[i];
  //    One_msec_Delay();
  //  }
  //  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  //  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}

void HOST(void){
//******************************************************************************
//
//  Description: This function is used to configure the hostname on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead, display_2
//
// Passed:
//
// Local: i, HOSTcmd
//
// Return: VOID
//
//******************************************************************************

  int i = INITIAL;
  //// SET HOSTNAME ////
  display_2 = " SET HOST ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  char HOSTcmd[Pos34] = "AT+S.SCFG=ip_hostname,ECE-306_02_C";
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos34; i++) {
    UCA1TXBUF = UCA0TXBUF = HOSTcmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
  // Wait for IOT confirmation
  IOT_ACK();

  ////// GET HOSTNAME ////
  //  display_2 = " GET HOST ";
  //  Display_Process();
  //  Five_msec_Delay(BIG_DELAY);
  //
  // // Initialize IOT_RX ring
  //  for(i=INITIAL; i < SMALL_RING_SIZE; i++) IOT_RX[i] = INITIAL;
  //
  //  IOTRead = INITIAL;
  //  for(i=INITIAL; i < Pos21; i++) {
  //    UCA1TXBUF = UCA0TXBUF = HOSTcmd[i];
  //    One_msec_Delay();
  //  }
  //  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  //  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}

void NPM(void){
//******************************************************************************
//
//  Description: This function is used to configure the network privacy mode
//  on NCSU wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead, display_2
//
// Passed:
//
// Local: i, NPMcmd
```

```
//
// Return: VOID
//
//******************************************************************************

  int i = INITIAL;
  //// SET NETWORK PRIVACY MODE////
  display_2 = " SET NPM  ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  char NPMcmd[Pos26] = "AT+S.SCFG=wifi_priv_mode,0";
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos26; i++) {
    UCA1TXBUF = UCA0TXBUF = NPMcmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
  // Wait for IOT confirmation
  IOT_ACK();

  ////// GET NETWORK PRIVACY MODE ////
  //  display_2 = " GET NPM  ";
  //  Display_Process();
  //  Five_msec_Delay(BIG_DELAY);
  //
  // // Initialize IOT_RX ring
  //  for(i=INITIAL; i < SMALL_RING_SIZE; i++) IOT_RX[i] = INITIAL;
  //
  //  IOTRead = INITIAL;
  //  for(i=INITIAL; i < Pos24; i++) {
  //    UCA1TXBUF = UCA0TXBUF = NPMcmd[i];
  //    One_msec_Delay();
  //  }
  //  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  //  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
```

```
}

void NET_MODE(void){
//******************************************************************************
//
//  Description: This function is used to configure the network mode on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead, display_2
//
// Passed:
//
// Local: i, NETcmd
//
// Return: VOID
//
//******************************************************************************

  int i = INITIAL;
  //// SET NETWORK MODE ////
  display_2 = " SET MODE ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  char NETcmd[Pos21] = "AT+S.SCFG=wifi_mode,1";
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos21; i++) {
    UCA1TXBUF = UCA0TXBUF = NETcmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
  // Wait for IOT confirmation
  IOT_ACK();

  ////// GET NETWORK MODE ////
  //  display_2 = " GET MODE ";
  //  Display_Process();
  //  Five_msec_Delay(BIG_DELAY);
  //
  // // Initialize IOT_RX ring
  //  for(i=INITIAL; i < SMALL_RING_SIZE; i++) IOT_RX[i] = INITIAL;
  //
  //  IOTRead = INITIAL;
  //  for(i=INITIAL; i < Pos19; i++) {
  //    UCA1TXBUF = UCA0TXBUF = NETcmd[i];
  //    One_msec_Delay();
  //  }
  //  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  //  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}

void FLASH(void){
//******************************************************************************
//
//  Description: This function is used to flash the IOT device on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead, display_2
//
// Passed:
//
// Local: i, F1cmd, F2cmd
//
// Return: VOID
//
//******************************************************************************
```

```
  int i = INITIAL;
  //// Begin Flash ////
  display_2 = "  FLASH   ";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  char F1cmd[Pos4] = "AT&W";
  char F2cmd[Pos9] = "AT+CFUN=0";
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  for(i=INITIAL; i < Pos4; i++) {
    UCA1TXBUF = UCA0TXBUF = F1cmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
  // Wait for IOT confirmation
  IOT_ACK();
  //// Finalize Flash ////
  display_2 = "FLASH DONE";
  Display_Process();
  Five_msec_Delay(BIG_DELAY);
  // Initialize IOT_RX ring
  INIT_IOT_RX();
  for(i=INITIAL; i < Pos9; i++) {
    UCA1TXBUF = UCA0TXBUF = F2cmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}

void PING(void){
//*****************************************************************************
//
//  Description: This function is used to configure the IOT device on NCSU
//  wireless network
//
```

```
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead
//
// Passed:
//
// Local: i, PINGcmd
//
// Return: VOID
//
//****************************************************************************

  int i = INITIAL;
  // Initialize IOT_RX ring
  char PINGcmd[Pos23] = "AT+S.PING=152.14.99.126";
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos23; i++) {
    UCA1TXBUF = UCA0TXBUF = PINGcmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  One_msec_Delay();
}

void OPEN_SOCKET(void){
//****************************************************************************
//
//  Description: This function is used to configure the IOT device on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead
```

```
//
// Passed:
//
// Local: i, OScmd
//
// Return: VOID
//
//*****************************************************************************

  int i = INITIAL;
  // Initialize IOT_RX ring
  char OScmd[Pos15] = "AT+S.SOCKD=2552";
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos16; i++) {
    UCA1TXBUF = UCA0TXBUF = OScmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}

void CHECK_SOCKET(void){
//*****************************************************************************
//
//  Description: This function is used to configure the IOT device on NCSU
//  wireless network
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: IOT_RX[SMALL_RING_SIZE], IOTRead
//
// Passed:
//
// Local: i, CHKScmd
//
```

```
// Return: VOID
//
//*****************************************************************************

  int i = INITIAL;
  // Initialize IOT_RX ring
  char CHKScmd[Pos22] = "AT+S.STS=ip_sockd_port";
  INIT_IOT_RX();
  // Send command
  for(i=INITIAL; i < Pos22; i++) {
    UCA1TXBUF = UCA0TXBUF = CHKScmd[i];
    One_msec_Delay();
  }
  UCA1TXBUF = UCA0TXBUF = TX_FINISH;
  Five_msec_Delay(BIGGER_DELAY + ONE_MSEC);
}
```

9.12. runtime_IOT.c

```
//*****************************************************************************
//
//  Description: This file implements the IOT runtime 'operating system'
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//*****************************************************************************

//Points to Header files for Preprocessing
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"

void Runtime_IOT(void){
//*****************************************************************************
//
//  Description: This function implements the IOT operating system post
//  configuration
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: Time_Sequence, display_1, display_2, display_3, display_4,
// PosL1, PosL2, PosL3, PosL4, big, IOT_RX[],
//
// Passed: L, R, T, i
//
// Local: L, R, T, i
//
// Return: VOID
//
//*****************************************************************************

  // Format the display
  Display_Format();
  display_2 = "SYS. READY";
  Display_Process();
  Five_msec_Delay(BIGGER_DELAY);
  // Initialize variables
  IOT_Runtime = TRUE;
  int L,R,T = TRUE;
  // Initialize IOT read ring
  INIT_IOT_RX();
  // Initialize Time_Sequence
  Time_Sequence = INITIAL;

  while(ALWAYS) {
    // Command has been issued
    if (IOT_RX[Pos0] == '*') {
      // HARD CODED SECURITY PIN
      if ((IOT_RX[Pos1] == '7') && (IOT_RX[Pos2] == '7') && (IOT_RX[Pos3] == '7') && (IOT_RX[Pos4] == '7')) {
        Five_msec_Delay(TRUE);
        // Disable Interrupt
        UCA1IE &= ~UCRXIE;
        // FORWARD COMMAND
        if(IOT_RX[Pos5] == 'F') {
          // Hex conversion operation
          L = IOT_RX[Pos6] - NUMBER_OP;
          R = IOT_RX[Pos7] - NUMBER_OP;
          T = IOT_RX[Pos8] - NUMBER_OP;
          IOT_driveForward(L, R, T);
        }
        // REVERSE COMMAND
        else if(IOT_RX[Pos5] == 'R') {
          // Hex conversion operation
          L = IOT_RX[Pos6] - NUMBER_OP;
          R = IOT_RX[Pos7] - NUMBER_OP;
          T = IOT_RX[Pos8] - NUMBER_OP;
          IOT_driveReverse(L, R, T);
        }
        // CLOCKWISE COMMAND
        else if(IOT_RX[Pos5] == 'C') {
          // Hex conversion operation
          T = IOT_RX[Pos6] - NUMBER_OP;
          IOT_Clockwise(T);
        }
        // COUNTER-CLOCKWISE COMMAND
        else if(IOT_RX[Pos5] == 'Q') {
          // Hex conversion operation
          T = IOT_RX[Pos6] - NUMBER_OP;
          IOT_Counterclockwise(T);
        }
        else if (IOT_RX[Pos5] == 'O') {
          IOT_driveForward(Pos1, Pos5, Pos3);
        }
        // Reinitialize IOT read ring
        INIT_IOT_RX();
        // Reenable Interrupt
        UCA1IE |= UCRXIE;
      }
    }
    if ((IOT_RX[Pos18] == 'L') && (IOT_RX[Pos19] == 'o') && (IOT_RX[Pos20] == 's') && (IOT_RX[Pos21] == 't')){
        IOT_RESET();
        PING();
        INIT_IOT_RX();
      }
    if (IOT_RX[Pos0] == '+') {
      Five_msec_Delay(TRUE);
      if ((IOT_RX[Pos5] == ':') && (IOT_RX[Pos6] == '4') && (IOT_RX[Pos7] == '1')){
        IOT_RESET();
        PING();
        INIT_IOT_RX();
      }
    }
    // Manual WiFi Reconnect
    if (!(P4IN & SW1)) {
      Five_msec_Delay(LIL_SW_DELAY);
      IOT_RESET();
      PING();
      INIT_IOT_RX();
    }
    // Look to break
    if (!(P4IN & SW2)) {
      Five_msec_Delay(LIL_SW_DELAY);
      break;
    }
    // Runs PING at Time_Sequence max
    // Reinitializes Time_Sequence
    if(Time_Sequence > CASE250){
      //PING();
      Time_Sequence = INITIAL;
    }
  }
}

void IOT_driveForward(unsigned int driveLeft, unsigned int driveRight, unsigned int driveTime){
//******************************************************************************
//
//  Description: This function implements the IOT forward drive
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
```

```
//
// Globals: Time_Sequence, display_1, display_2, display_3, display_4,
// PosL1, PosL2, PosL3, PosL4, big,  A0_time_count
//
// Passed: driveLeft, driveRight
//
// Local: driveLeft, driveRight, driveTime
//
// Return: VOID
//
//****************************************************************************

  // Initialize Display
  Display_Format();
  display_2 = "Forward";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
  // Initialize drive states
  Protect_Motors();
  driveTime *= TIME_MULT;
  Time_Sequence = A0_time_count = INITIAL;
  // Drive based on passed variables
  while(A0_time_count <= driveTime) {
    Left_Forward_On(driveLeft);
    Left_Forward_Off();
    Right_Forward_On(driveRight);
    Right_Forward_Off();
  }
  // Reset drive state
  Protect_Motors();
  Time_Sequence = A0_time_count = INITIAL;
  // Back to IOT runtime configuration
  display_2 = "SYS. READY";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
}

void IOT_driveReverse(unsigned int driveLeft, unsigned int driveRight, unsigned int driveTime){
```

```
//******************************************************************************
//
//  Description: This function implements the IOT reverse drive
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: Time_Sequence, display_1, display_2, display_3, display_4,
// PosL1, PosL2, PosL3, PosL4, big,  A0_time_count
//
// Passed: driveLeft, driveRight
//
// Local: driveLeft, driveRight, driveTime
//
// Return: VOID
//
//******************************************************************************

  // Initialize Display
  Display_Format();
  display_2 = " Reverse  ";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
  // Initialize drive states
  Protect_Motors();
  driveTime *= TIME_MULT;
  Time_Sequence = A0_time_count = INITIAL;
  // Drive based on passed variables
  while(A0_time_count <= driveTime) {
    Left_Reverse_On(driveLeft);
    Left_Reverse_Off();
    Right_Reverse_On(driveRight);
    Right_Reverse_Off();
  }
  // Reset drive state
  Protect_Motors();
  Time_Sequence = A0_time_count = INITIAL;
  // Back to IOT runtime configuration
  display_2 = "SYS. READY";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
}

void IOT_Clockwise(unsigned int driveTime){
//*****************************************************************************
//
//  Description: This function implements the IOT clockwise drive
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: Time_Sequence, display_1, display_2, display_3, display_4,
// PosL1, PosL2, PosL3, PosL4, big,  A0_time_count
//
// Passed: driveTime
//
// Local: driveTime
//
// Return: VOID
//
//*****************************************************************************

  // Initialize Display
  Display_Format();
  display_2 = "Clockwise ";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
  // Initialize drive states
  Protect_Motors();
  driveTime *= TIME_MULT;
  Time_Sequence = A0_time_count = INITIAL;
  // Drive based on passed variables
  while(A0_time_count <= driveTime) {
    Left_Forward_On(TRUE);
    Left_Forward_Off();
    Right_Reverse_On(TRUE);
    Right_Reverse_Off();
  }
  // Reset drive state
  Protect_Motors();
  Time_Sequence = A0_time_count = INITIAL;
  // Back to IOT runtime configuration
  display_2 = "SYS. READY";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
}

void IOT_Counterclockwise(unsigned int driveTime){
//******************************************************************************
//
//  Description: This function implements the IOT counter-clockwise drive
//
//  Team 2
//  Nov 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (6.30.3)
//
// Globals: Time_Sequence, display_1, display_2, display_3, display_4,
// PosL1, PosL2, PosL3, PosL4, big,  A0_time_count
//
// Passed: driveLeft, driveRight, driveTime
//
// Local: driveTime
//
// Return: VOID
//
//******************************************************************************

  // Initialize Display
  Display_Format();
  display_1 = " Counter  ";
  display_2 = "Clockwise ";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
  // Initialize drive states
  Protect_Motors();
  driveTime *= TIME_MULT;
  Time_Sequence = A0_time_count = INITIAL;
  // Drive based on passed variables
  while(A0_time_count <= driveTime) {
    Left_Reverse_On(TRUE);
    Left_Reverse_Off();
    Right_Forward_On(TRUE);
    Right_Forward_Off();
  }
  // Reset drive state
  Protect_Motors();
  Time_Sequence = A0_time_count = INITIAL;
  // Back to IOT runtime configuration
  Display_Format();
  display_2 = "SYS. READY";
  Display_Process();
  //Five_msec_Delay(BIG_DELAY);
}
```

10. RACECARLSSON.java

This section includes the source file for the java program used to control the vehicle of the wireless network. This code is utilized to implement the wireless and IOT functions of the vehicle. See different source code sections with details below.

```
//******************************************************************************
//
//  Description: This file contains the remote control program for the
//  RACECARLSSON C52 device. Utilizes server socket connections to remotely
//  command the device.
//
//  Team 2
//  Nov 2015
//  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
//******************************************************************************
```

```
import java.awt.Color;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class RACECARLSSON implements ActionListener, KeyListener{
	// INSTANCE VARIABLES
	static
	// CMD-Line parameter enables debug log in console
	boolean debug = false;
	Socket s;
	int serverPort = 2552;
	DataOutputStream dos;
	String newLine  = System.lineSeparator();
	Boolean connectMode = true;
	// GUI Objects
	JPanel topPanel = new JPanel();
	JPanel centerPanel = new JPanel();
	JPanel bottomPanel = new JPanel();
	JButton connectButton = new JButton("Connect");
	JLabel serverAddressLabel = new JLabel("Server Address: ");
	JTextField serverAddressTF = new JTextField();
	JTextArea controlTF = new JTextArea();
	JTextArea logTA = new JTextArea();
```

```
JScrollPane logScrollPane    = new JScrollPane(logTA);
JFrame cmdWindow = new JFrame("RACECARLSSON C52");

public RACECARLSSON() {
//****************************************************************************
//
//  Description: RACECARLSSON Constructor Method
//
//  Team 2
//  Nov 2015
//  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
//
// Global: debug, newLine
//
// Passed:
//
// Local:
//
// Return: VOID
//
//****************************************************************************

        if (debug) System.out.println(newLine + "Christopher Woedy - RACECARLSSON" + newLine);
        // Set grid layouts
        topPanel.setLayout(new GridLayout(1,2));           // rows/cols
        centerPanel.setLayout(new GridLayout(1,2));     // rows/cols
        bottomPanel.setLayout(new GridLayout(1,1));     // rows/cols
        // Add Elements
        topPanel.add(serverAddressLabel);
        topPanel.add(serverAddressTF);
        centerPanel.add(controlTF);
        centerPanel.add(logScrollPane);
        bottomPanel.add(connectButton);
        // Add to Content Panes
        cmdWindow.getContentPane().add(topPanel,"North");
        cmdWindow.getContentPane().add(centerPanel,"Center");
        cmdWindow.getContentPane().add(bottomPanel,"South");
        // Modifiy Colors
        connectButton.setBackground(Color.black);
        connectButton.setForeground(Color.green);
        // Set Font
        connectButton.setFont (new Font("default",Font.PLAIN,20));
        serverAddressLabel.setFont (new Font("default",Font.PLAIN,20));
        serverAddressTF.setFont (new Font("default",Font.PLAIN,20));
        controlTF.setFont (new Font("default",Font.PLAIN,20));
        logTA.setFont (new Font("default",Font.PLAIN,20));
        // Set Log
        logTA.setEditable(false);
        logTA.setLineWrap(true);
        logTA.setWrapStyleWord(true);
        logTA.setBorder(BorderFactory.createLineBorder(Color.black));
        logTA.setText("-------------- LOG --------------");
        //Set Control
        controlTF.setBorder(BorderFactory.createLineBorder(Color.black));
        controlTF.setEditable(false);
        controlTF.setText("---------- CONTROLS ----------"
                + newLine + "Forward:" + newLine + "W - Forward"
                + newLine + "Q - Left & Forward" + newLine + "E - Right & Forward"
                + newLine + newLine + "Reverse:" + newLine + "S - Reverse"
                + newLine + "A - Left & Reverse" + newLine + "D - Right & Reverse"
                + newLine + newLine + "X - Clockwise" + newLine + "Z - CounterClockwise");
        // Add Listeners
        connectButton.addActionListener(this);
        connectButton.addKeyListener(this);
        controlTF.addKeyListener(this);
        logTA.addKeyListener(this);
        serverAddressLabel.addKeyListener(this);
        // Show the window
        cmdWindow.setSize(560,500);
        cmdWindow.setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
    //******************************************************************************
    //
    //  Description: RACECARLSSON Action Handler used to implement actions from GUI
```

```
//  GUI objects.
//
//  Team 2
//  Nov 2015
//  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
//
// Global:
//
// Passed: ae
//
// Local:
//
// Return: VOID
//
//********************************************************************************

        // Connect ActionListener
        if (ae.getSource() == connectButton) {
        // Connect
         if (connectMode) {
                 connectButton.setText("Disconnect");
                 connectButton.setForeground(Color.red);
                 connectMode = false;
                 connect();
         }
         // Disconnect
         else {
                serverAddressLabel.setForeground(Color.black);
                connectButton.setText("Connect");
                connectButton.setForeground(Color.green);
                connectMode = true;
                disconnect();
         }
        }
}

private void connect() {
//********************************************************************************
```

```
//
//  Description: Used to open a socket connection on the RACECARLSSON device
//
//  Team 2
//  Nov 2015
//  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
//
// Global: debug, newLine, s
//
// Passed:
//
// Local: serverAddress, ioe
//
// Return: VOID
//
//*************************************************************************
        if (debug) System.out.println("Entering connect().");
        logTA.append(newLine+"Connecting to device...");
        logTA.setCaretPosition(logTA.getDocument().getLength());
        // Get IP Address
        String serverAddress = (String) serverAddressTF.getText().trim();
        //Attempt to connect to server
         try {
                s = new Socket(serverAddress, serverPort);
         }
         catch (IOException ioe)
         {
                 serverAddressLabel.setForeground(Color.red);
                 if (debug) System.out.println(ioe.toString());
                 logTA.append(newLine+ioe.toString());
                 logTA.setCaretPosition(logTA.getDocument().getLength());
                 return; // Stop connection
         }
        //Connection established
  if (debug) System.out.println("Connected to RACECARLSSON C52");
        logTA.append(newLine+"Connected to RACECARLSSON C52");
        logTA.setCaretPosition(logTA.getDocument().getLength());
  // GUI Cleanup
```

```
            serverAddressLabel.setForeground(Color.black);
            serverAddressTF.setEditable(false);
            connectButton.setText("Disconnect");
        }

        private void disconnect() {
        //***************************************************************************
        //
        //  Description: Used to close a socket connection on the RACECARLSSON device.
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, newLine, s
        //
        // Passed:
        //
        // Local: ioe
        //
        // Return: VOID
        //
        //***************************************************************************

                try {
                        // Close Socket Connection
                        s.close();
                }
                catch (IOException ioe) {
                        if (debug) System.out.println(ioe.toString());
                        logTA.append(newLine+ioe.toString());
                        logTA.setCaretPosition(logTA.getDocument().getLength());
                }
                // GUI Cleanup
                 serverAddressLabel.setForeground(Color.black);
                 serverAddressTF.setEditable(true);
                logTA.append(newLine+"Disconnected from device");
                logTA.setCaretPosition(logTA.getDocument().getLength());
```

```
	}

	@Override
	public void keyPressed(KeyEvent ke) {
	//****************************************************************************
	//
	//  Description: Handles commands sent to the RACECARLSSON device by keystrokes
	//
	//  Team 2
	//  Nov 2015
	//  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
	//
	// Global:
	//
	// Passed:
	//
	// Local: ke
	//
	// Return: VOID
	//
	//****************************************************************************
		// Forward
		if (ke.getKeyChar() == 'w') {
			forward();
		}
		// Reverse
		if (ke.getKeyChar() == 's') {
			reverse();
		}
		// Forward Left
		if (ke.getKeyChar() == 'q') {
			forwardLeft();
		}
		// Forward Right
		if (ke.getKeyChar() == 'e') {
			forwardRight();
		}
		// Reverse Left
```

```
            if (ke.getKeyChar() == 'a') {
                    reverseLeft();
            }
            // Right
            if (ke.getKeyChar() == 'd') {
                    reverseRight();
            }
            // Clockwise
            if (ke.getKeyChar() == 'x') {
                    clockwise();
            }
            // Counter-Clockwise
            if (ke.getKeyChar() == 'z') {
                    counterclockwise();
            }
    }

    private void forward() {
    //*******************************************************************************
    //
    //  Description: Sends the FORWARD command to the device
    //
    //  Team 2
    //  Nov 2015
    //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
    //
    // Global: debug, s, dos
    //
    // Passed:
    //
    // Local: e
    //
    // Return: VOID
    //
    //*******************************************************************************

 try {
    // Send Command
```

```
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777F111");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}


        private void reverse() {
        //****************************************************************************
        //
        //  Description: Sends the REVERSE command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //****************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777R111");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}
```

```
        private void forwardLeft() {
        //*****************************************************************************
        //
        //  Description: Sends the FORWARD LEFT command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //*****************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777F121");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}

        private void forwardRight() {
        //*****************************************************************************
        //
        //  Description: Sends the FORWARD RIGHT command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //***************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777F211");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}


        private void reverseLeft() {
        //***************************************************************************
        //
        //  Description: Sends the REVERSE LEFT command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //******************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777R121");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}

        private void reverseRight() {
        //******************************************************************************
        //
        //  Description: Sends the REVERSE RIGHT command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //******************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777R211");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}
        private void clockwise() {
        //******************************************************************************
        //
        //  Description: Sends the Clockwise command to the device
        //
        //  Team 2
        //  Nov 2015
        //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
        //
        // Global: debug, s, dos
        //
        // Passed:
        //
        // Local: e
        //
        // Return: VOID
        //
        //******************************************************************************

    try {
        // Send Command
        dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
        dos.writeUTF("*7777C1");
        }
    catch (IOException e) {
        if (debug) System.out.println(e.toString());
                return;
        }
}


        private void counterclockwise() {
        //******************************************************************************
```

```
      //
      //  Description: Sends the CounterClockwise command to the device
      //
      //  Team 2
      //  Nov 2015
      //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
      //
      // Global: debug, s, dos
      //
      // Passed:
      //
      // Local: e
      //
      // Return: VOID
      //
      //****************************************************************************

    try {
      // Send Command
      dos = new DataOutputStream(new DataOutputStream(s.getOutputStream()));
      dos.writeUTF("*7777Q1");
      }
    catch (IOException e) {
      if (debug) System.out.println(e.toString());
            return;
      }
}

      public static void main(String[] args) {
      //****************************************************************************
      //
      //  Description: MAIN Routine
      //
      //  Team 2
      //  Nov 2015
      //  Built with Eclipse IDE for Java Developers Version: Mars.1 Release (4.5.1)
      //
      // Global: debug
      //
      // Passed: args[]
      //
      // Local: e
      //
      // Return: VOID
      //
      //*****************************************************************************************

            // Handle debugging case
            if (args.length != 0) {
                if (args[0].contains("debug")) debug = true;
            }
            if (debug) System.out.println("Debugging is enabled");
            // Start Program
            new RACECARLSSON();
      }
      @Override
      // Unimplemented - Required by Compiler
      public void keyReleased(KeyEvent ke) {}

      @Override
      // Unimplemented - Required by Compiler
      public void keyTyped(KeyEvent ke) {}
}
```

11. Conclusion

As a whole, the process of designing, assembling, and testing of the vehicle called for a significant amount of trial and error, careful inspection, and collaboration. Although problems were encountered with most every aspect of the vehicle's design process and implementation, some aspects were more challenging than others, giving rise to countless errors to debug before optimizing the final product.

During the analog-to-digital conversion (ADC) component of the process, some difficulty was encountered in understanding the steps to convert analog signals to digital signals when using the emitters, detectors, and thumb wheel. This was remedied by first, clearly outlining and understanding the steps needed to make the conversion and then transferring this plan into a software implementation.

Frequently during testing and debugging, issues surrounding the LCD were encountered. Because the underlying LCD code was inaccessible, it was essential to work around the limitations of the LCD in order to progress in testing. This involved exhaustive debugging and stepping through many lines of code-one line at a time-to see the desired output display on the LCD, ultimately working around any glitches that occurred at runtime.

At one point in time, the FRAM experimenter board ran into a manufacturing defect and proved to be ineffective for some time. This was resolved by ordering a new FRAM Experimenter board to replace the faulty component. However, this process may be avoided by testing each component and performing the battery test to ensure no factory defects.

The varying battery voltage also affected the motor speed performance of the vehicle as well. Lower voltages resulted in less power and slower speeds; to remedy this batteries were recharged often and used during testing sessions as well. Additionally with motor control, environmental factors, such as dust and dirt allocation on wheels, had a negative impact on the vehicle's performance. These factors would result in less traction on wheels, causing the vehicle to be more prone to spinning out or sliding more often than necessary. This problem was resolved by frequently cleaning the wheels prior to testing so the vehicle could run normally.

The serial communication component of the vehicle was the most challenging in that it involved a complete redesign in the thought process for its implementation. After toiling for hours on the initial approach, it was difficult to gravitate towards completely forgetting this approach and starting from scratch. Fortunately, through a significant amount of thought, another more effective approach was developed that allowed for the serial communication component of the process to work seamlessly.

Regarding the IOT device, the most cumbersome problems encountered involved Wi-Fi variability and network issues. Frequently, the Wi-Fi would time out or the module would be unable to connect to the network due to the presence of other devices. To work around these issues, a socket-sever connection with the vehicle was created, which also added an extra measure of security while on the network as well. Runtime resets also allowed the Wi-Fi module to re-associate with the wireless access point, often with the same IP address.

Ultimately, the overall problem that was resolved through every stage of development was adjusting software to operate appropriately in a real-world environment. This was accomplished through considering delays in code and optimizing software to run more efficiently and ultimately operate faster at runtime. Even through the challenges and short-comings, this process lead to the development of a successful product.