2019 Autonomous Car Prototype

Driving us into the Future

Team 5

Revision	Description
1A	Adam: Contributed to the scope of the document. Creatred the abbreviations area, the brief discription on 3.4 Motor, and inserted the diagrams for all of part 4.
2A	Reece: Added all of 6. Test Process.
3A	Maria: Contributed to the scope of the document, added descriptions to 3.1, 3.3, 3.2, made cosmetic changes and updated table of contents.
4A	Michael: Contributed to the scope of the document, created all of the diagrams for Section 3.
1B	Adam: Updated coverpage, abbrevations, descriptions in part 4, and fixed revisions.
2B	Reece: Contributed to the scope of the document. Added figure names/table of figures, edited 3.2. Power Supply. Added 6.4. Finger Test, adjusted formatting of the document.
3B	Maria: Added descriptions for 4.3, 4.4, and brief description of Section 6.
4B	Michael: Revised the diagrams in Section 3. Contributed to the descriptions of 4.2.
1C	Adam: Fixed issues with table of contents, table of figures, inconsistent car names, figure 3.1, 3.4, 4.1, 4.2, formatting of text on 6.1, figure 6.3, figure 6.2. Added code and description for section 7.2.

Team Members

Michael Barilla

Adam Casper

Reece Neff

Maria Samia

Originator: Team 5

Checked:

Released:

Filename: ECE 306 Write Up

Title: **2019 Autonomous Car Prototype Driving us into the Future**

Date: **05/01/2019**

Document Number: **0-0000-000-0000-01**

Rev: **4E**

2C	Maria: Added description for sections 6.5, 7.1, 8.1, added necessary code for 9.1 and updated table of contents.
3C	Reece: Added content to 7.3, and 8.3, added necessary code for 9.3
4C	Michael: Added content to 7.4, 8.4 and 9.4 pertaining to the ADC on the car. Created flowchart for Ports.c and updated figure numbers to reflect the new images.
1D	Adam: Fixed multiple instances of ADC and Timers in the table and headings. Updated the sections 7.3, 8.3, 8.4, 8.5, 9.4, 9.5. Added additional photo of MSP. Updated the part 4 figures.
2D	Maria: Added content to 6.6, 7.5, flowcharts for 8.5 and code for 9.5.
3D	Reece: Added serial interrupts details to 7.5, flowcharts for 8.6, and code for 8.6
4D	Michael: Updated code and flowchart for Main to accommodate for Serial Communication. Adjusted Figure Numbers to accurately reflect the images.
1E	Adam: Fixed the front throughout the document. Remade the diagram for figure 8.2 and 8.3. Added code in section 9 for the interrupt switch 1, intercepting white, intercepting black, and the black line follow. Added descriptions and flowcharts for each of my functions.
2E	Michael: reformatted the code section to match the required formatting. Added personal code and completed Section 10, the conclusion.
3E	Maria: Added section 6.7, updated abbreviations. Added personal code and flowcharts for sections 8 and 9. Updated the final table of contents and figures.
4E	Reece: Added the power analysis and calculations in section 5. Added personal code and flowcharts for sections 8 and 9.

Table of Contents

Table of Figures

8. Scope

The 2019 Autonomous Car Prototype is an innovative take on the modern car. In this revolution, the car can be programmed through a straight forward user interface to follow a specified path. Another feature that is built into the car is the black line detection to follow the road as the car drives itself along a route. The Autonomous Car Prototype has an intuitive design to detect nearby cars and appropriate the correct instructions for various situations which will bring the passengers safely to their destination. This revolutionary vehicle will change the way society travels from one place to another in the years to come.

9. Abbreviations

Abbreviation	Definition
LCD	Liquid Crystal Display
LED	Light Emitting Diode
RC	Remote Control
FRAM	Ferroelectric Random-Access Memory
USB	Universal Serial Bus
I/O	Input / Output
ADC	Analog – to – Digital Converter
UI	User Interface
IOT	Internet of Things
BCD	Binary-coded Decimal
TXD	Transmitting Data
RXD	Receiving Data
PC	Personal Computer
TCP	Transmission Control Protocol
SSID	Service Set Identifier
IP	Internet Protocol

10. Overview

The 2019 Autonomous Car Prototype consists of a power supply, control board, MSP430FR2355 FRAM board, motor bridge, LCD display, and a user interface. See Figure 3.1 below.

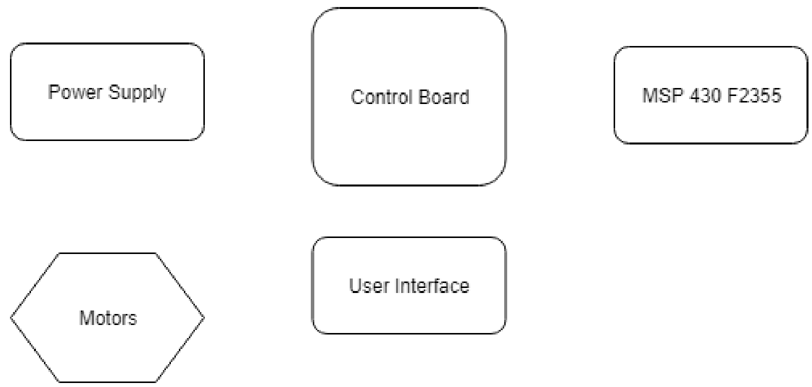

Figure 10.1 Overview Block Diagram

10.1 MSP430FR2355 FRAM Board

The MSP430FR2355 FRAM board features 2 switch buttons and 8 LEDs for user interface, an on-board ambient light sensor and connector for additional external analog sources, and a USB port for software input. Refer to Figure 3.1 below.

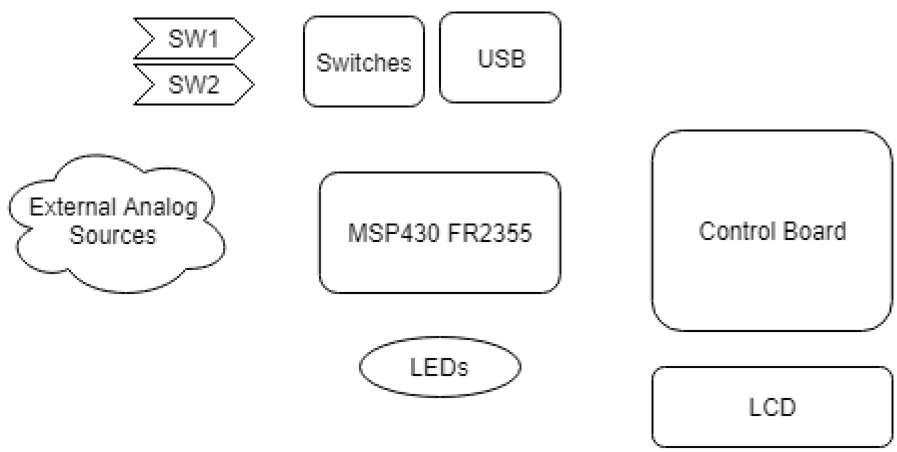

Figure 10.2 Block MSP430FR2355

10.2 Power Supply

The power supply is comprised of a set of 4 AA batteries which is connected to the power supply board through a buck boost converter providing the energy for all the 2019 Autonomous Car Prototype's necessary functions and equipment. The power system utilizes a switch to control the flow of energy in the system. Refer to Figure 3.3 below.

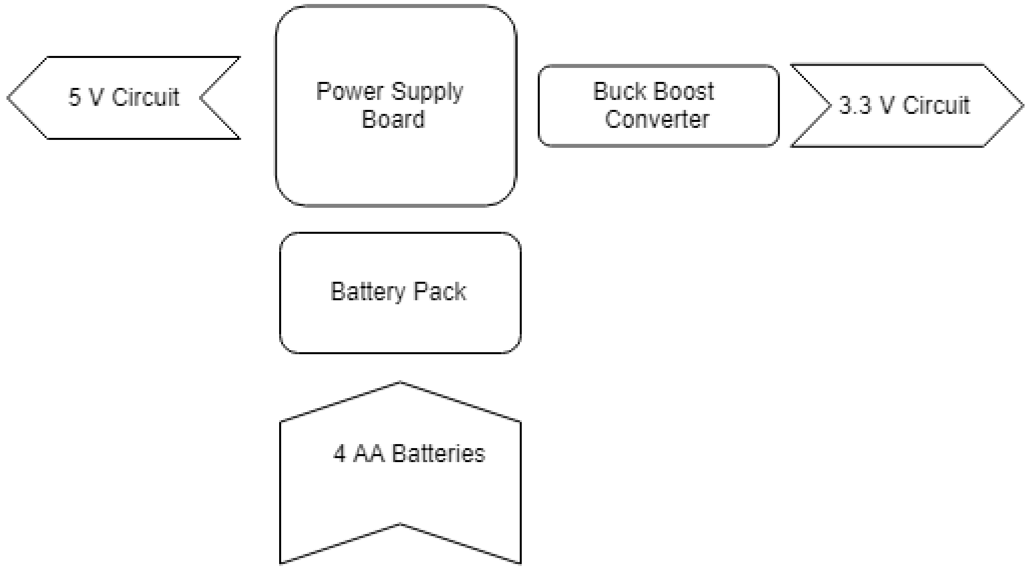

Figure 10.3 Power Supply Blocks

10.3 User Interface

The user interface block contains a LCD, 8 LEDs, Switch 1, Switch2, and a Thumb Wheel. Once powered, A working 2019 Autonomous Car Prototype should have LEDs blinking, LCD displaying messages to the user, and switches 1 and 2 that changes the display messages on the LCD. See Figure 3.3 below.

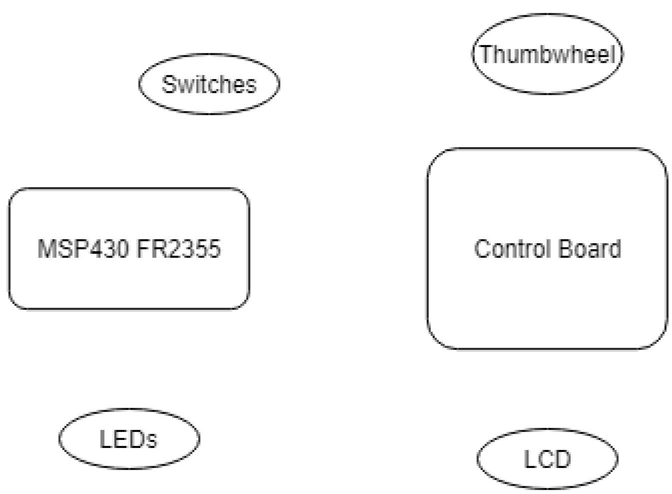

Figure 10.4 User Interface Blocks

10.4 Motor

The 2019 Autonomous Car Prototype contains two DC motors configured with an H–bridge control for forward command of the wheels. The H-bridge is connected through pins J21 and J43, for the right and left respectively. See Figure 3.4 below.

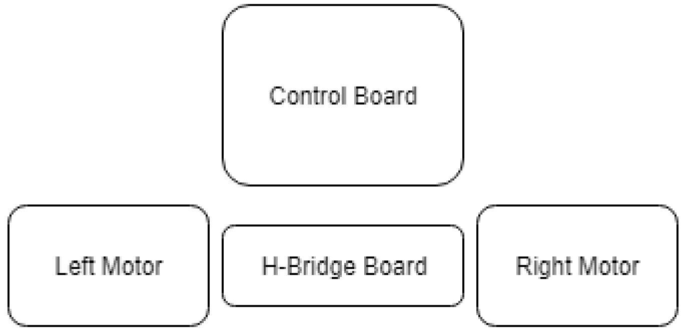

Figure 10.5 Motor Blocks

11. Hardware

Hardware for the project is made up of four major components - the MSP430FR2355 FRAM Board, Power System Board, User Interface devices, and two Motors. Below describes how each component should work according to the most recent project made based on the block diagrams shown above.

11.1 MSP430FR2355 FRAM Board

The MSP430FR2355 FRAM board is an embedded microcontroller that is focused on sensing and measurement. This FRAM board is ultra-low-power and includes 2 switch buttons and 8 LEDs for user interface, an on-board ambient light sensor and connector for additional external analog sources, and a USB port for software input. The MSP430FR2355 takes advantage of a 16-bit RISC CPU, a constant generator, and 16-bit registers.

	MSP430FR2355
Non-volatile memory (kB)	32
RAM (KB)	4
ADC	12-bit SAR
ADC: channels (#)	12
GPIO pins (#)	44
I2C	2
SPI	4
UART	2
Comparator channels (#)	2
Package Group	LQFP | 48 TSSOP | 38 VQFN | 40
Approx. price (US$)	2.40 | 1ku
Timers - 16-bit	4
Bootloader (BSL)	UART
Operating temperature range (C)	-40 to 105
Package size: mm2:W x L (PKG)	48LQFP: 81 mm2: 9 x 9 (LQFP | 48) 38TSSOP: 62 mm2: 6.4 x 9.7 (TSSOP | 38) 40VQFN: 36 mm2: 6 x 6 (VQFN | 40)
Features	DAC OpAmp PGA Real-Time Clock

Figure 11.2 MSP430FR2355 Specs

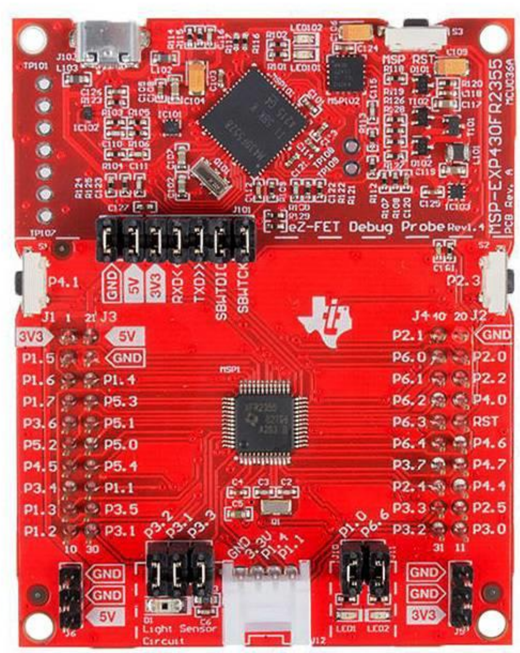

Figure 11.1 MSP430FR2355

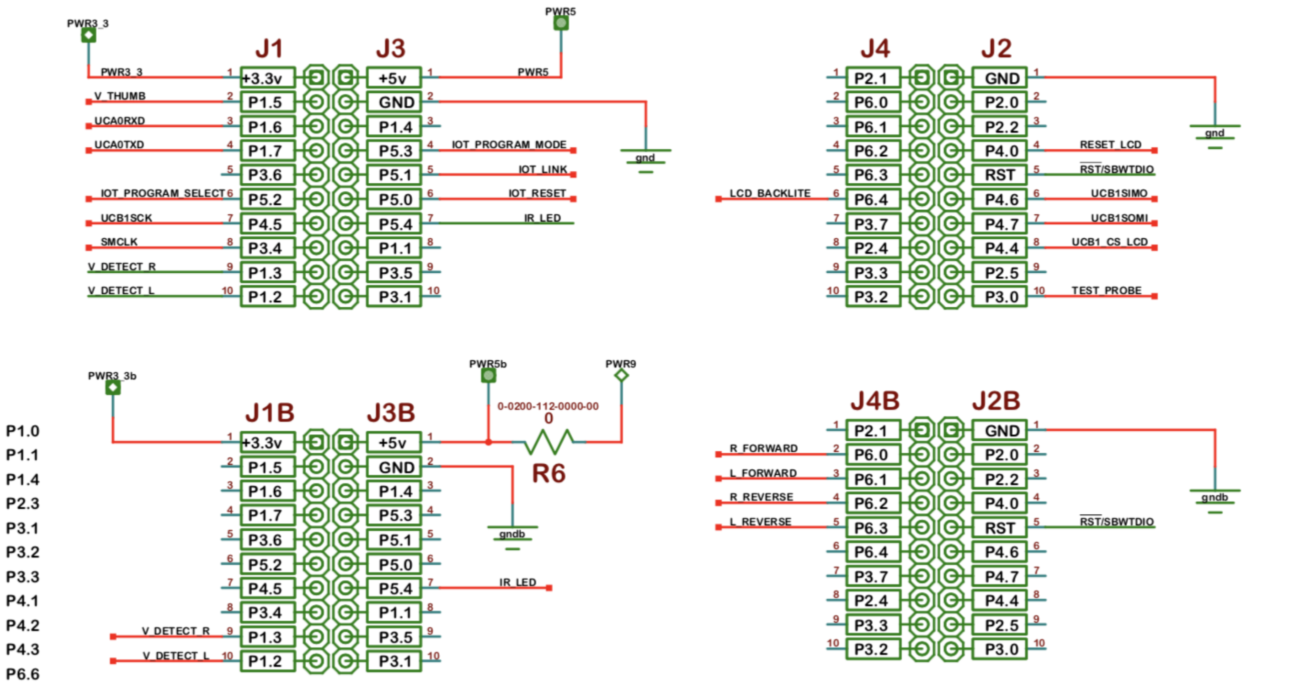

Figure 11.3 MSP430 FRAM Board

11.2 Power System

The power system block diagram takes in 4 AA batteries (aprox.6V). Using a Buck Boost Converter, the 6V is diminished to a 3.3V circuit. The result is a 5V circuit and a 3.3V circuit which are outputted through the respective pins and can be used by the control board for the rest of the system.

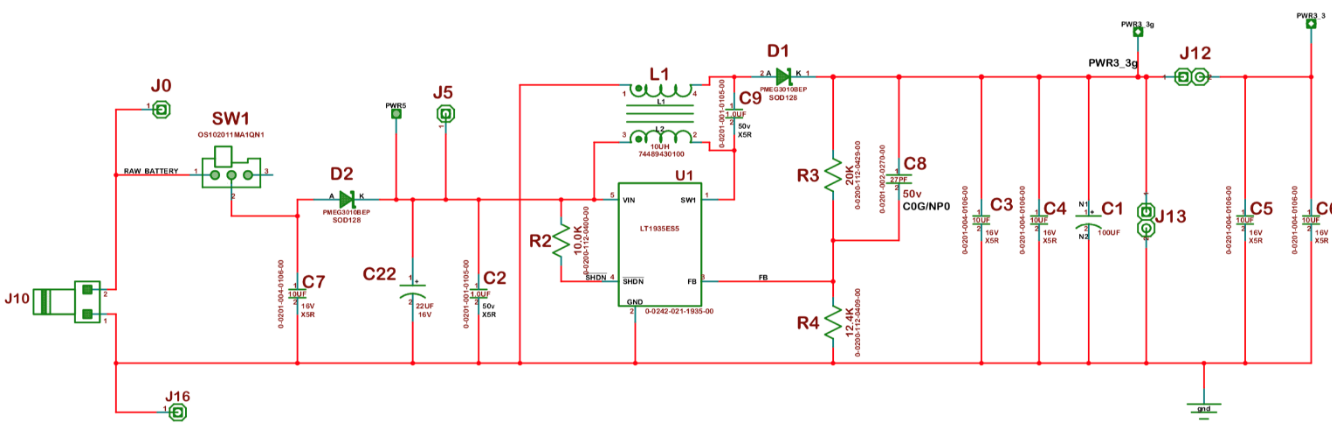

Figure 11.4 Buck Boos Converter/General Power Systems

11.3 User Interface

The user interface block contains a LCD, 8 LEDs, Switch 1, Switch2, and a Thumb Wheel. Once powered, a working 2019 Autonomous Car Prototype should have LEDs blinking, switches 1 and 2 that prompts the 2019 Autonomous Car Prototype to create circle, triangle, or infinity shapes and display messages on the LCD screen to the appropriate shape made. See Figure 4.5 and 4.6 below.

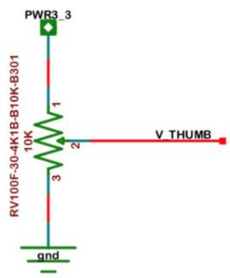

Figure 11.5 Potentiometer

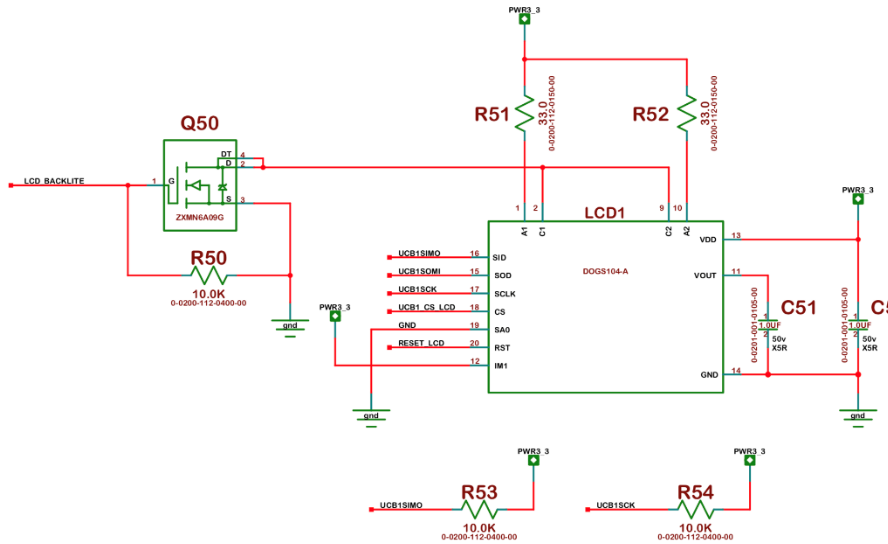

Figure 11.6 Buttons and LCD

11.4 Motor

The 2019 Autonomous Car Prototype contains two DC motors configured with an H–bridge control of the forward command of the wheels. The H-bridge is connected through pins J21 and J43, for the right and left controls respectively. A working motor should create the necessary shaped by turning off or on the forward command of either left or right wheels. See Figure 4.7 and 4.8 for left and right figure below.

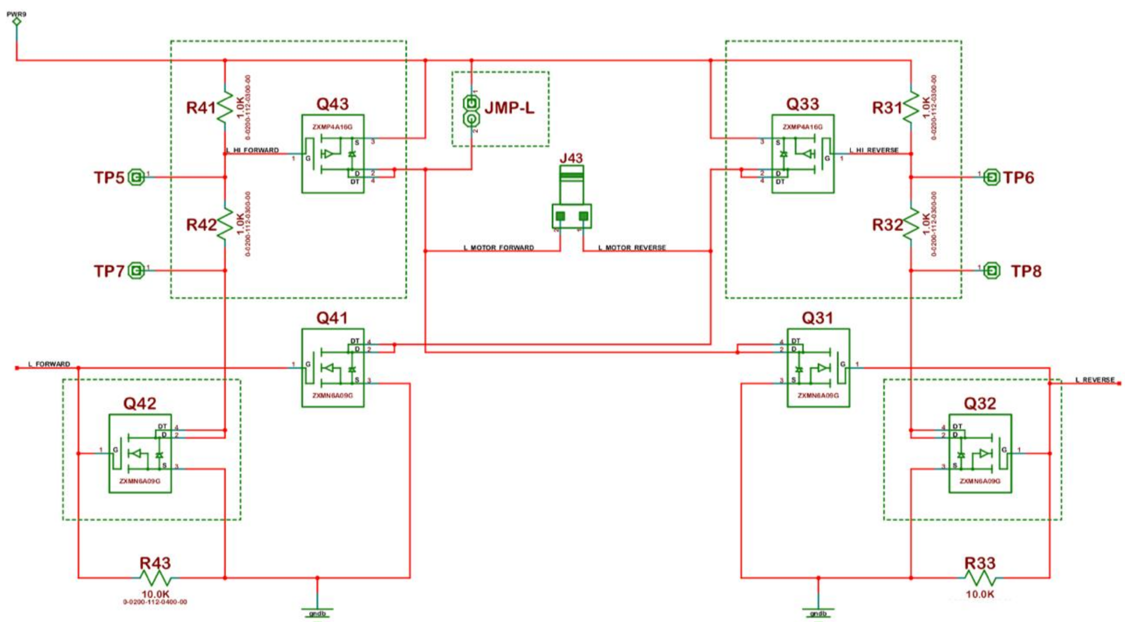

Figure 11.7 Left Motor Schematic

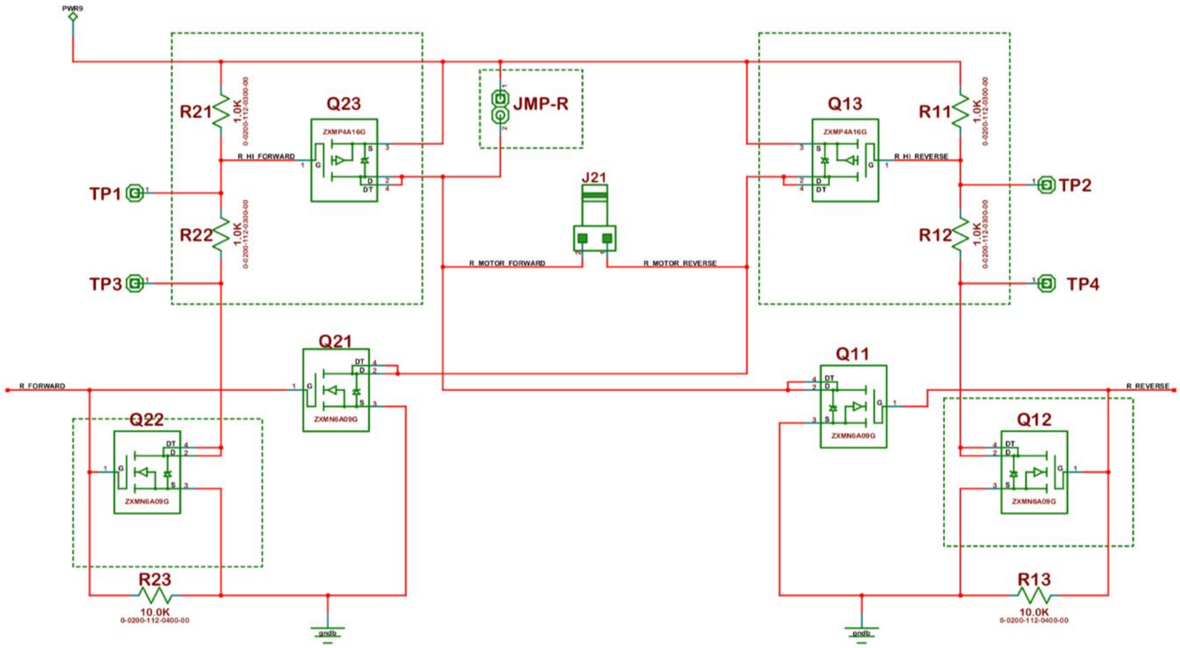

Figure 11.8 Right Motor Schematic

12. Power Analysis

Component	Current	Voltage	Power
IOT Module	100mA	3.38V	0.338W
IR Emitter	50mA	2.12V	0.106W
IR Sensor	50mA	1.37V	0.0685W
LCD Display	206mA	1.87V	0.38522W
Motors (50% Speed)	390mA	4.62V	1.8W
Entire Car	**554mA**	**4.9V**	**2.7146W**

Since the total power draw of the car is 2.7146 Watts, each battery delivers 678.65 mW (4 batteries). Eneloop batteries were used, but this will be calculated for typical AA batteries. At 0.2 Amp-hours, the car will last 21.6 minutes before running out of charge.

13. Test Process

Section 6 is a compilation of all the hardware tests done to ensure a perfectly working 2019 Autonomous Car Prototype. Most of the test process is done to check that the correct voltage is distributed on the board and the LCD is powered correctly.

13.1 Power Supply Test

Test the power supply with at least a 5 Volt Power Supply and an oscilloscope. First, test the power supply with the oscilloscope to make sure the output is correct. After that, follow the instructions on the diagram below.

If the tested numbers are incorrect, inspect the parts with a magnifying device to ensure your components are properly soldered to the board.

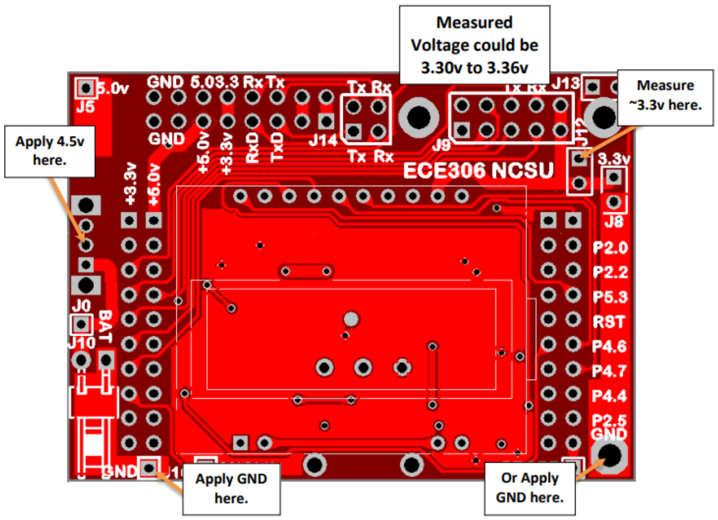

Figure 13.1 Power Board Test

13.2 Power Connector Test

To test this, you will need a Volt Meter with a Diode setting. Using the diagram and images below, measure with the positive probe on J0 and negative probe on J5; you should read an open circuit. After this, flip the probes and you should get around 165 ohms.

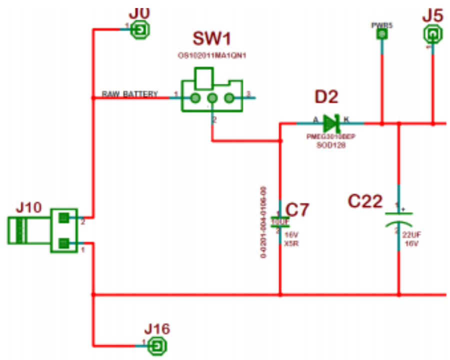

Figure 13.2 Switch Power Schematic

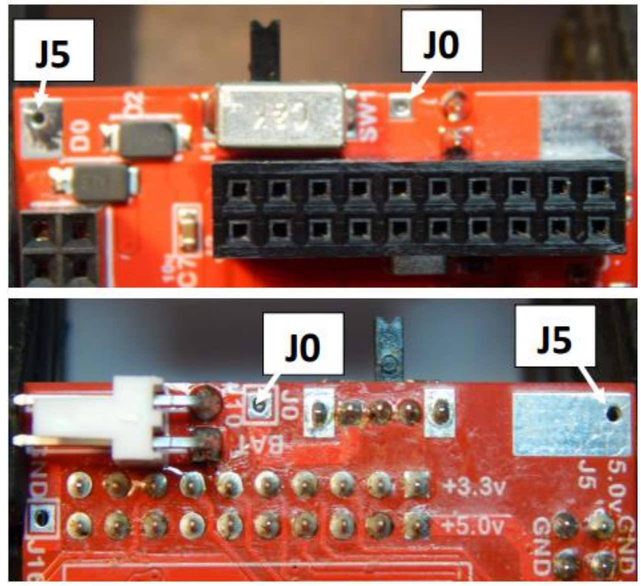

Figure 13.3 Switch Power Test

13.3 LCD Test

Power the board with the battery pack and the backlight should be on and working with readable characters. Press S1 and S2 to ensure the LCD changes.

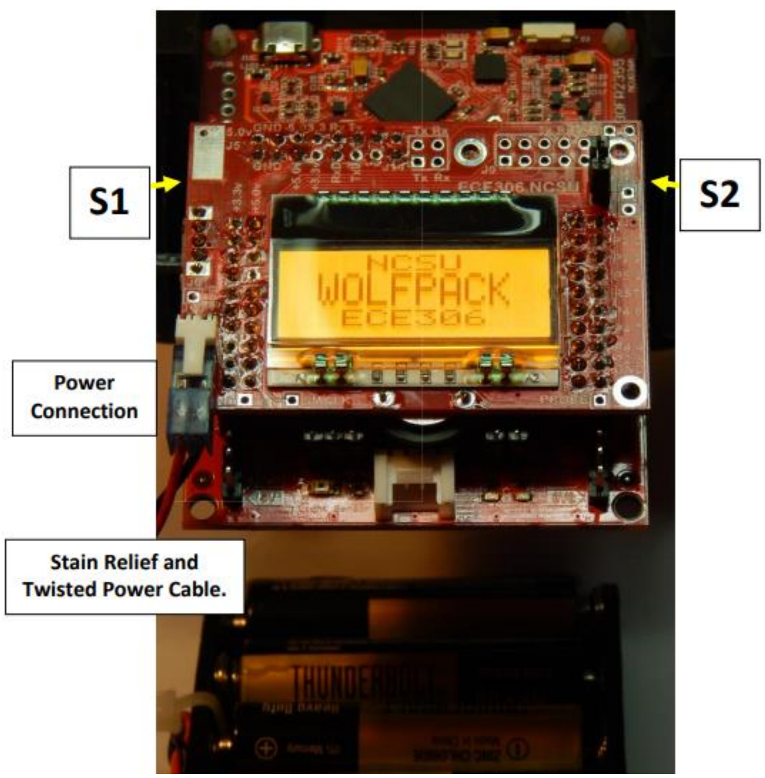

Figure 13.4 LCD Test

13.4 Finger Test

Any time software/hardware is modified, hold your finger on important components (FETs, Inductors, etc.) to feel for heat. You should not be able to feel any heat coming from the part. If you do, turn off the board and do further tests with an oscilloscope, volt-ohm meter, and your test code to diagnose and fix the problem.

13.5 Black Line Test

A mounted IR LED emitter and two detectors were used to detect the color of the surface that was directly under the detectors. Continuous sampling was done to take buffer values that could be considered black or white.

13.6 Communication Test

A test can be done to check serial communication with the PC. Using a jumper to connect pins TXD and RXD, enables the MSP430 to send back to the PC commands instructed in the Terminal Emulator. This will provide a hardware loop back with the PC using only the programming interface. Typing a string on the terminal emulator should show up on the terminal emulator.

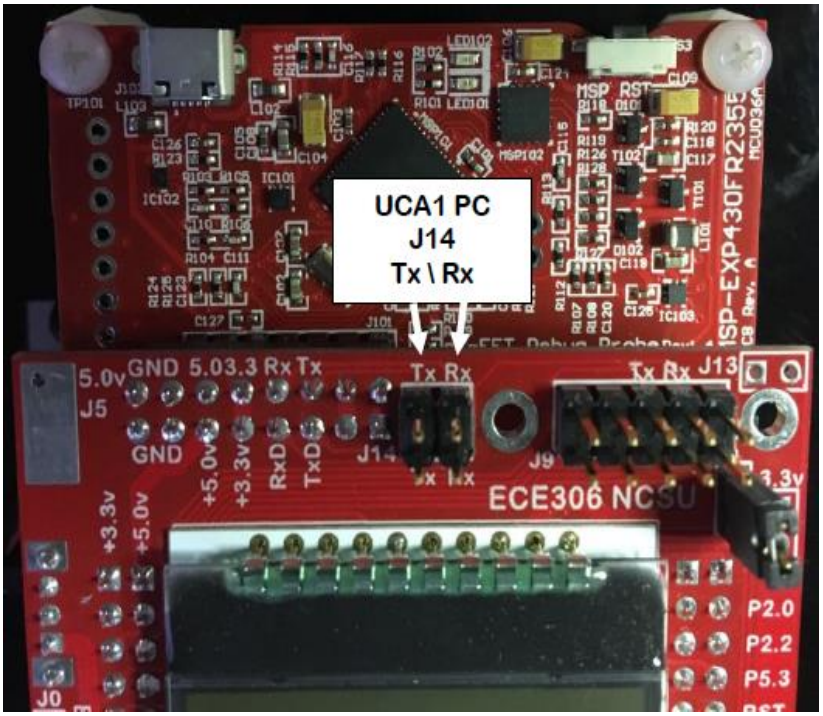

Figure 13.5 Communication Test

13.7 IOT Test

After soldering the IOT module and connector pin on the board, have the IOT reset and reprogram. Any time software/hardware is modified, hold your finger on the IOT board to feel for heat. You should not be able to feel any extreme heat coming from the part. If you do, remove the IOT board and do further tests with an oscilloscope, volt-ohm meter, and your test code to diagnose and fix possible shorting or other problems.

A test can be done to check IOT communication with the PC. The software must be set up to take characters received on one port and transmit each to the other port. Using a jumper to connect pins TX and RX on pins J9 enables the MSP430 to send response to the PC commands instructed in the Terminal Emulator. After receiving the correct response, the IOT board may be installed on the MSP430.

14. Software

The software is configured using a modular approach. Describe the code structure. Remember to identify the various functions and what operates when. This is a description of how your software is configured. You should be able to give this to one of your class mates and they would understand what you tried to do.

14.1 Main

The main file will be the first code that would operate upon compiling or debugging. It should contain multiple functions that initializes ports, clocks, variables, conditions, timers, LCD, Serial Communication, and Analog to Digital Converter. The main function contains a while loop that changes the state of the LEDs, turns a variable on at the positive edge of clock, calls other necessary functions and allows interrupt modules when appropriate.

14.2 Timers

This should initialize the timers needed for operation of the car. Called from the main function with an Init Timers function, this sets the clock for each timer and enables their interrupts to be used in other functions.

14.3 ADC / ADC Interrupt

The Analog to Digital Converter (ADC) is used to measure the thumbwheel position as well as both Infrared Light Detector. Because the converter is implemented using an interrupt, so it must have an initialization, Pragma vector, Interrupt Service Routine (ISR), and a statement to enable the interrupt. Within the Initialization, the ADC is configured to the proper settings. The “#Pragma” vector assigns the proper interrupt vector to the ISR using the macro that is specified by the MSP430 header file that is included in the code. The ISR utilizes a switch statement to determine which type of interrupt is flagged based on the vector. If it is a ADCMEM0 memory register with the conversion result interrupt flag, then another switch statement is used to store the values of each of the input devices.

14.4 Serial

The serial should initialize the USB and CPU transmitter or receiver buffer and index and set the appropriate baud rate. Initialization is called once in the main file and then interrupts according to the current state of serial communication: Wait, Receive, and Transmit. The interrupt called then receives characters from the receive buffer and stores them in a ring buffer while the transmit interrupt sends characters from an array into the transmit buffer.

14.5 Main Function – Michael

The Main function of Michael’s system begins by configuring the settings of the device to the proper settings. Next, the while loop continuously cycles through three functions. Program which contains the main functionality of the device, Network which updates the network status and displays the necessary information on the screen, and utilities which processes the background functionality of the device – Menu, switches, display.

14.6 Network – Michael

The Network function is responsible for updating the network information. Upon joining a new network, the function stores the IP Address and SSID to their respective Arrays, begins TCP connection, and pings a website regularly to maintain the connection throughout the system’s runtime.

14.7 Program – Michael

The Program controls which function is called based on what state the car is in during its operation. If the car is waiting for an input, it displays that the device is waiting for an input. Other wise if the device is expected to follow the black line it runs the black line function. Otherwise it runs the drive function which processes the input from the controller and processes the wheels as the controller states.

14.8 Main – Adam

The main function contains almost all of the variables used throughout the program. The function start with initializing the ports, clocks, conditions, timers, lcd, adc, serial, and clears a few arrays used in communication. It then clears out the display and calls the start screen function. Once that is done the function moves into the while always loop which makes the IOT module ping and display the ssid and ip address on the display. It alternates the green and red leds and calls display ADC and display process functions. The last thing it does is checks for a new command being sent to the car and issues the proper command actions if the command was legal.

14.9 Intercept White – Adam

This function allows the car to move forward until the left or the right sensor read the value of white preset. Once the car detects white the function tells the car to stop and to move onto the intercept black function.

14.10 Intercept Black – Adam

This function allows the car to continue to move forward until the left of the right sensor hits the value of black that has been preset. Once the car detects the black tape the function tells the car to stop and updates the variable used to keep track of the amount of time the waiting step takes.

14.11 Follow Circle – Adam

The follow circle function starts by initializing the ADC and turning all of the wheels off. The function then updates the display and then uses a series of if statements in order to ensure that the car follows the black tape below the sensors. If both sensors are currently on the black tape the car will move straight. If one of the sensors are on black then the car will move left or right. If the left sensor is greater than the right then the car will move left, if the right sensor is greater than the left then the car will move to the right. If the car is no longer detecting black on the left or the right side then the car will turn clockwise or counter clockwise in order to move more sharply onto the line. In the event that the car has not saved the left or the right sensor as the last side that was on black the car turns clockwise. This function allows the car to follow the black line with sharp turns with ease.

14.12 Interrupt Switch 1 – Adam

The interrupt switch 1 function checks for when the switch one on the car is pressed when the button is pressed and as soon as it is detected it ensures that only one switch action is done at a time. For my switch function I also made it able to update important variables and change the display lines.

14.13 Main – Maria

The main file will be the first code that would operate upon compiling or debugging. It should contain multiple functions that initializes ports, clocks, variables, conditions, timers, LCD, Serial Communication, and Analog to Digital Converter. The main function contains should turn on the IOT reset after 500ms and clear the LCD display before going into the continuous while loop. The while loop contains several functions including a variable that indicates the positive edge of clock, a display process, hexadecimal to BCD process for the time display, IOT commands, black line commands, and interrupt modules that runs when appropriate.

14.14 Command – Maria

The command function starts by waiting for a signal from the interrupts that a command has been received, upon receiving the signal the function then goes to transferring the UCA0 array into a local array then goes into the command state in which the code takes in the char command, convert the char time into an integer and turn on wheel ports that carry out the desired action.

14.15 Line – Maria

The line function is the function that goes forward until it is intercepted by a black line and follows the black line. This function starts by going forward until it reads the white value from either sensor, upon reading the white value it then goes to a state that reads a black value. Reading a black value initiates the black line follow command. If both sensors are currently on the black tape the car will move straight. If one of the sensors are on black then the car will move left or right. If the left sensor is greater than the right then the car will move left, if the right sensor is greater than the left then the car will move to the right. The black line would stop after finishing the desired course and restart the process.

14.16 Thumbwheel – Reece

This code analyzes the ADC input from the thumbwheel. Once reading this value, it compares the difference between it and the set threshold for the thumbwheel. In this case, it is set to the halfway point. If the thumbwheel ADC value is higher than the threshold (aka more in the right side than left), the right wheel will slow down and the left wheel will speed up, biasing the vehicle more towards the right. The opposite happens when the thumbwheel ADC value is lower than the threshold.

14.17 Main – Reece

This function initializes the ports, clocks, conditions, timers, LCD, ADC, and serial communications. In addition to this, there is a while loop within the function set to TRUE (aka infinite loop) that runs the essential functions (Thumbwheel calibration, ADC Display, Millisecond Timer, and Display Process). All other functions are called via interrupts.

14.18 Serial ISR – Reece

The UCA0 interrupt activates whenever UCA0RXBUF changes or if the UCA0TX interrupt flag is raised. In the case the RX flag is raised, the function will send the information into a ring array, looping back to the beginning if the information exceeds the size if the ring. In the case an escape character followed by an 'E' occurs, the function will reset the ring array on the next call. For the UCA0TX flag raised, the information in the array is sent out to the UCA0TX buffer every call until the 10th character is sent. After this, the TX interrupt is disabled.

15. Flow Chart

This section contains the flowcharts pertaining to all of the functions.

15.1 Main

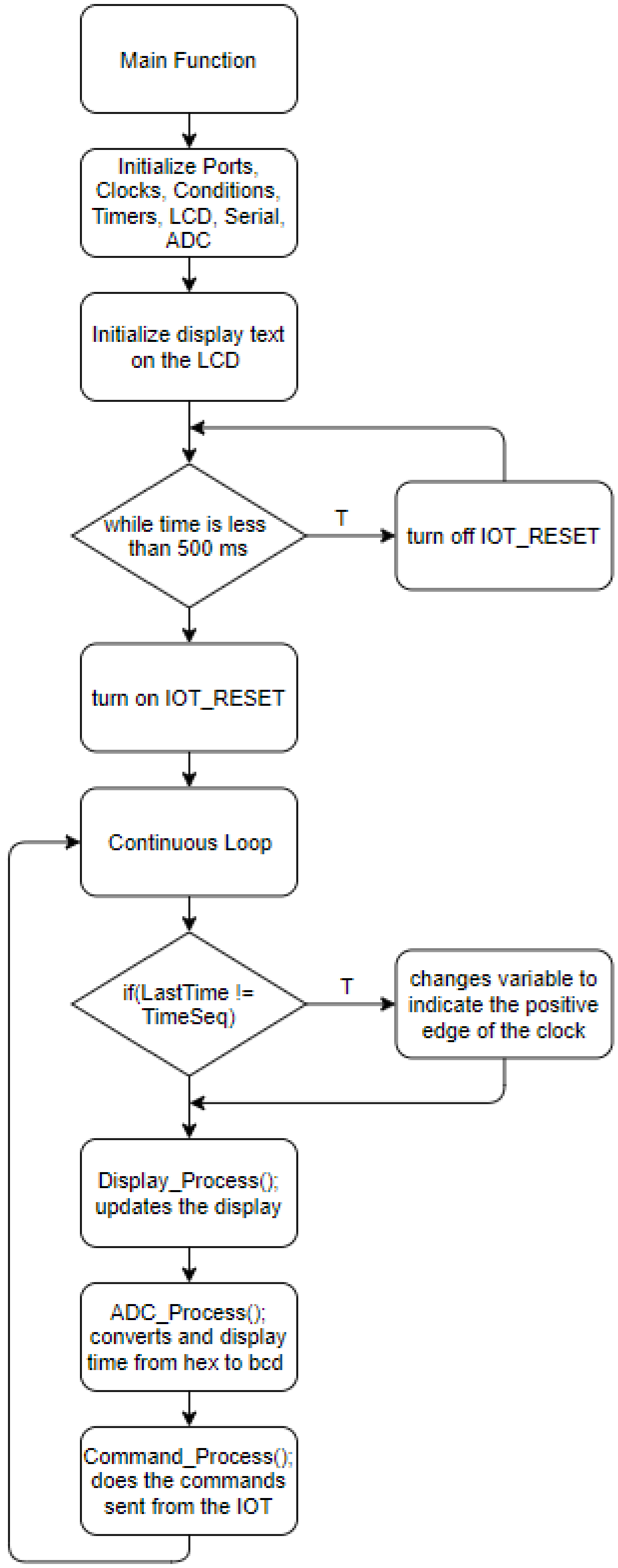

Figure 15.1 Main Function Flowchart

15.2 Timers

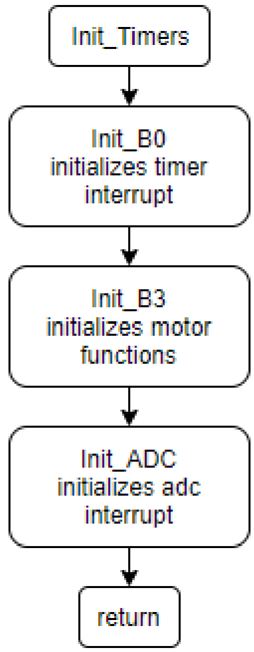

Figure 15.3 Init Timers Flowchart

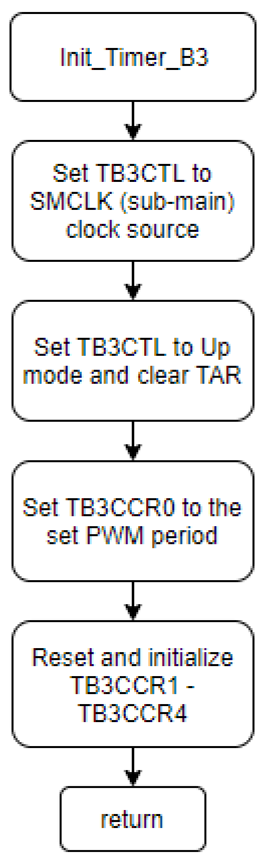

Figure 15.2 Initialize Timer B3 Flowchart

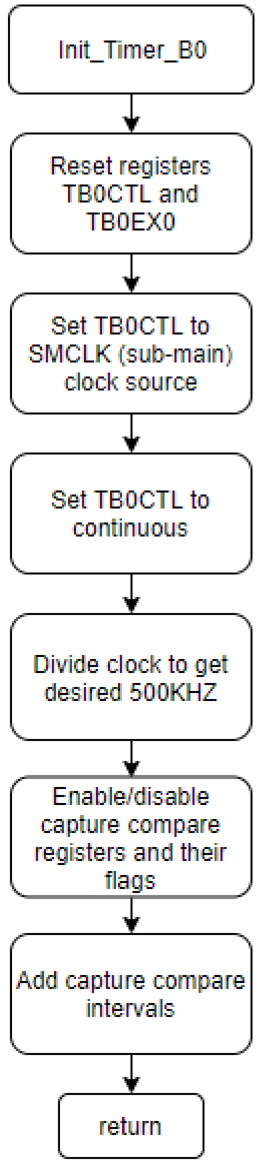

Figure 15.4 Initialize Timer B0 Flowchart

15.3 ADC

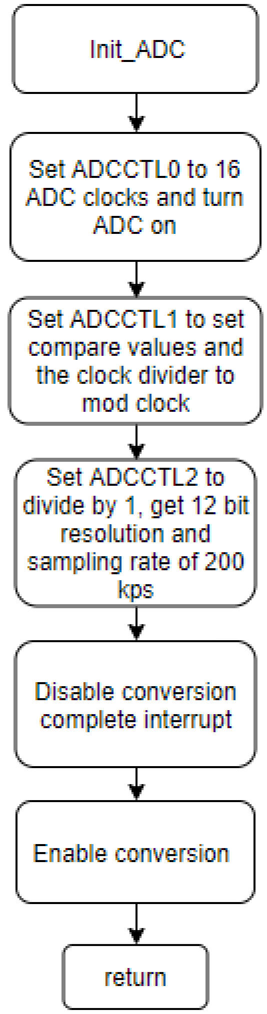

Figure 15.5 Initialize ADC

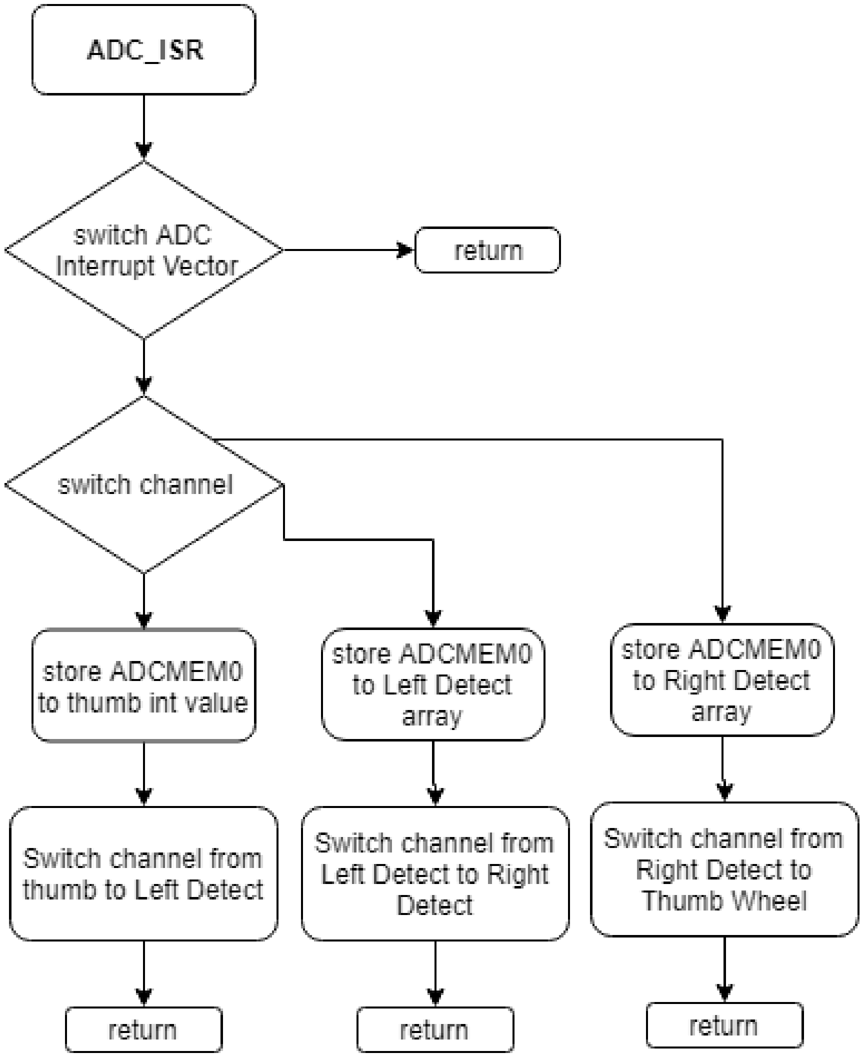

Figure 15.6 Interrupt ADC Flowchart - Michael

15.4 Serial

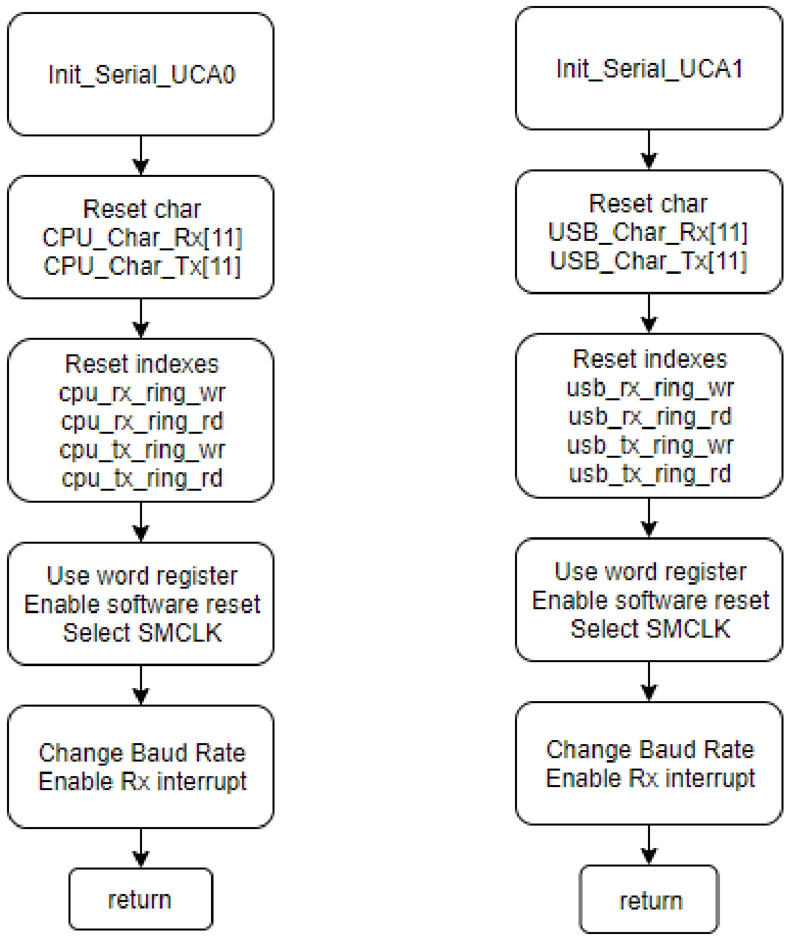

Figure 15.7 Initialize Serial Flowchart

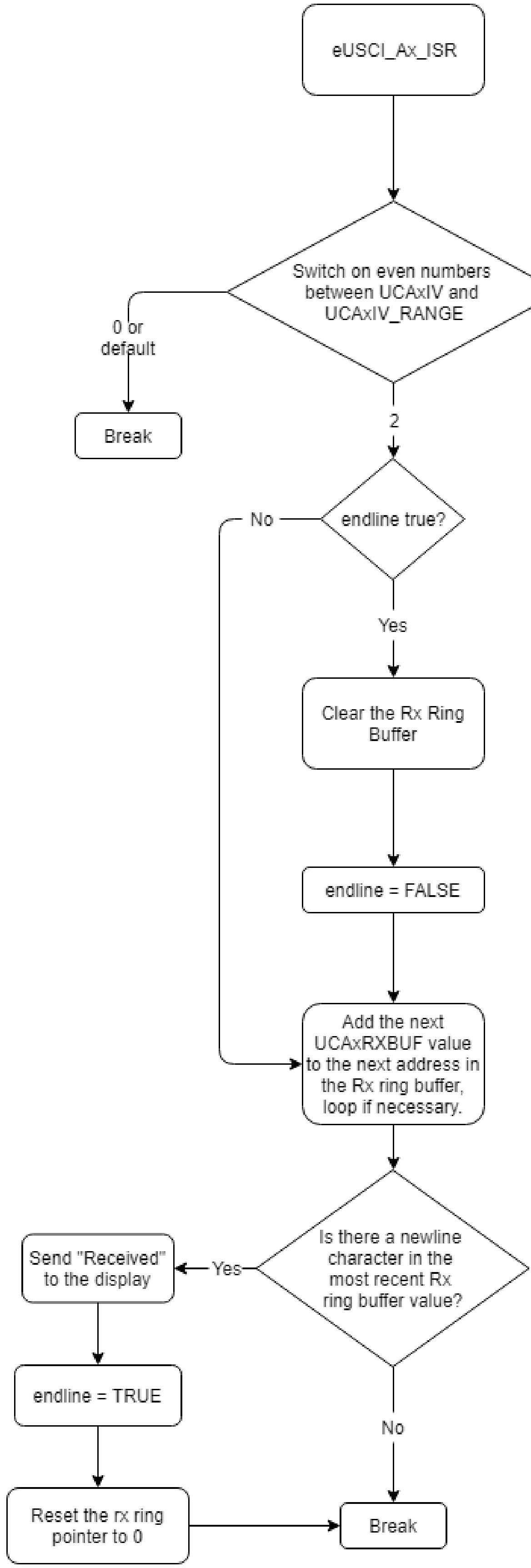

Figure 15.8 Interrupt Serial Flowchart

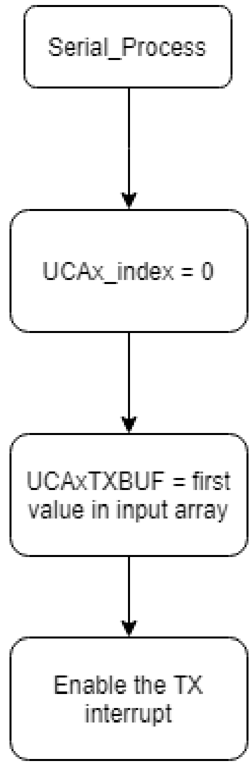

Figure 15.9 Serial TX Call Function

15.5 Main - Michael

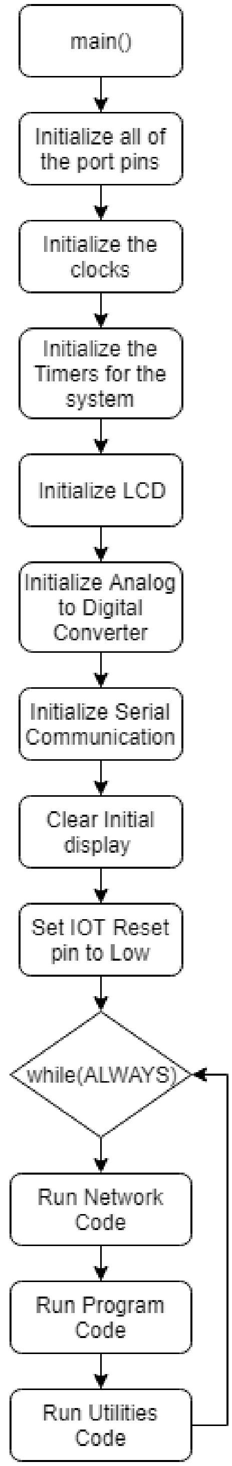

Figure 15.10 Main Flowchart - Michael

15.6 Network Flowchart - Michael

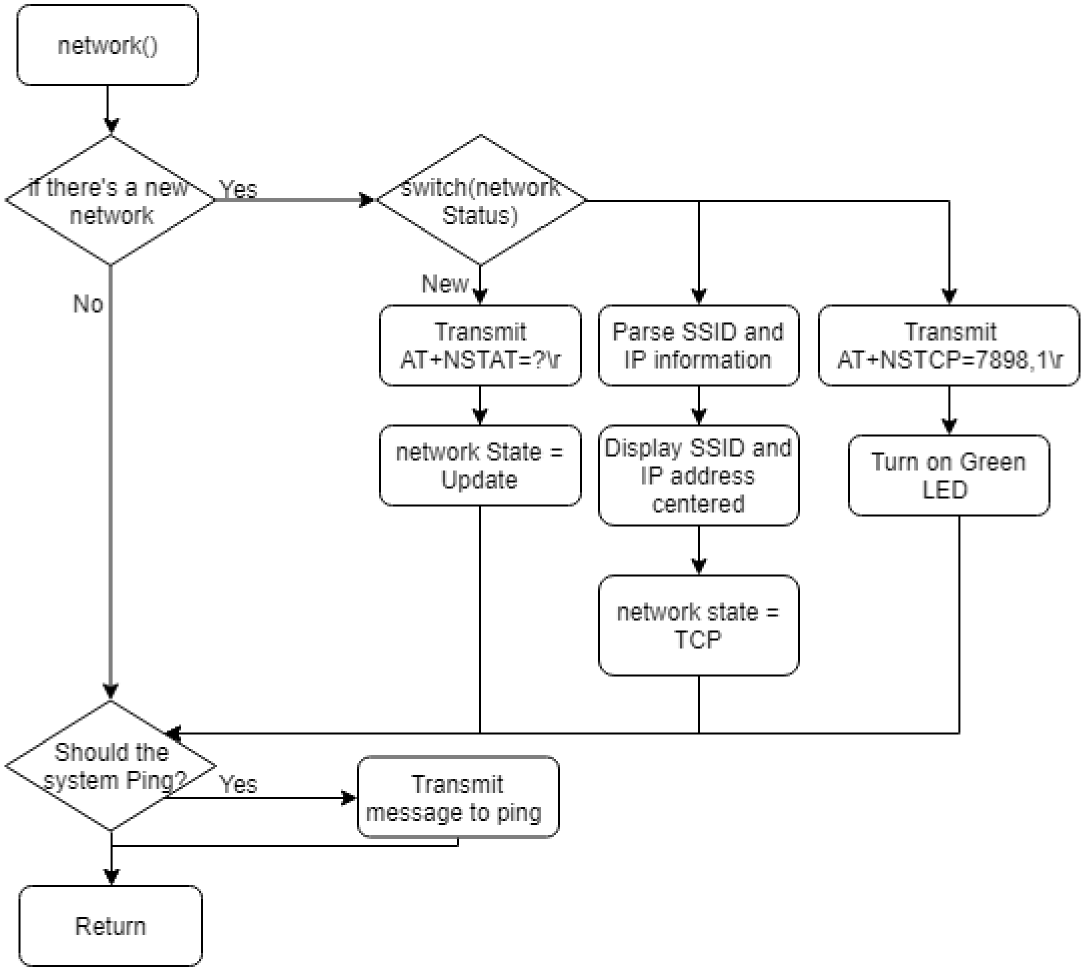

Figure 15.11 Network Flowchart - Michael

15.7 Program Flowchart - Michael

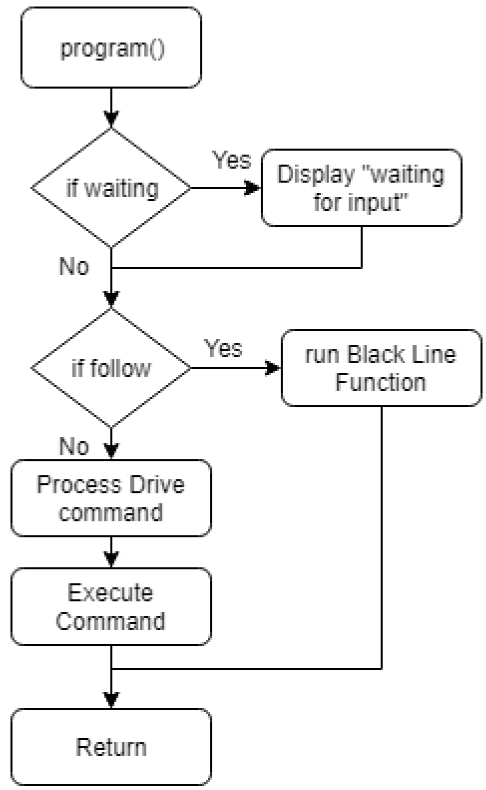

Figure 15.12 Program Flowchart - Michael

15.8 ADC Flowchart - Michael

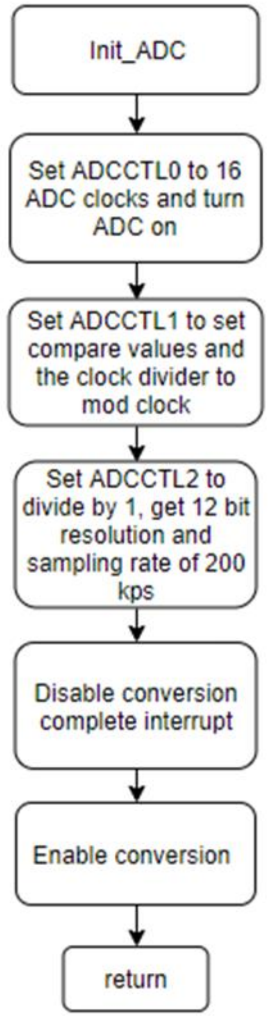

Figure 15.13 ADC Flowchart - Michael

15.9 Main - Reece

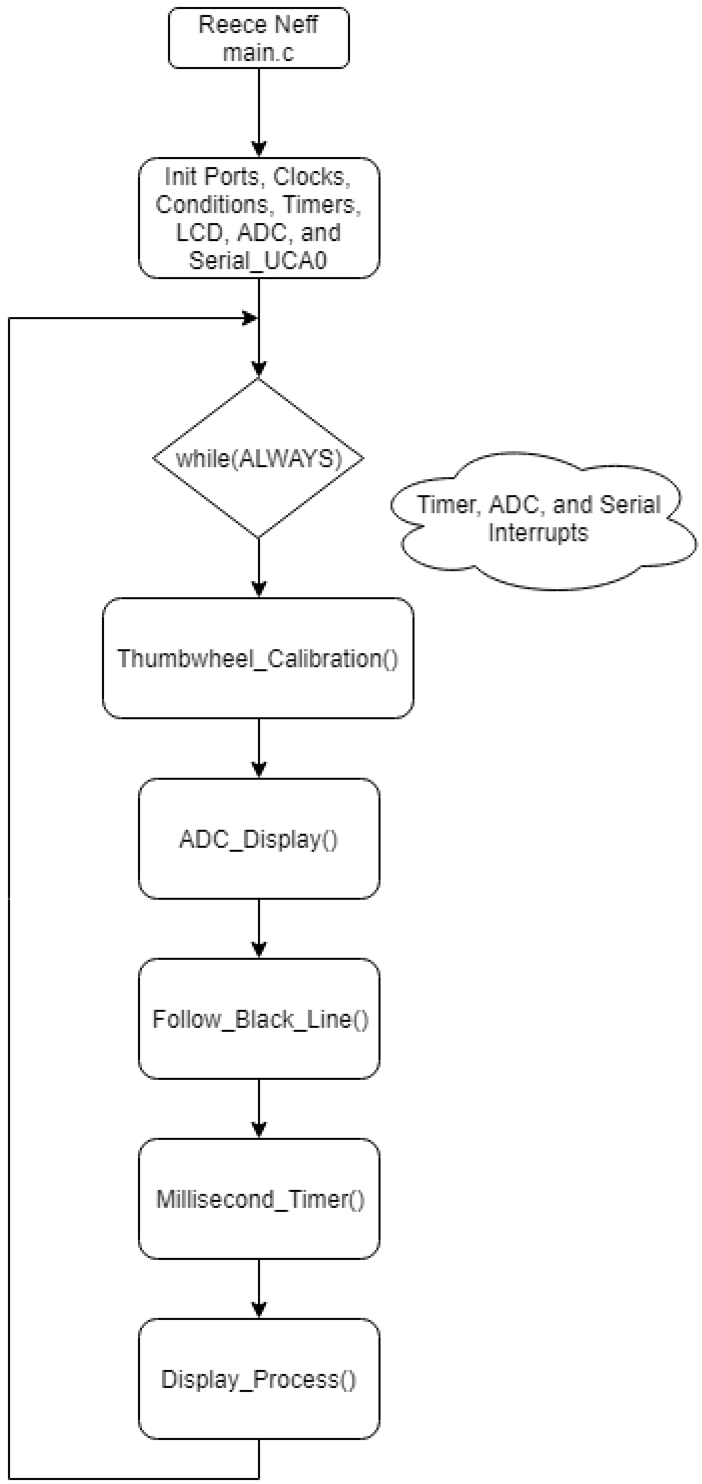

Figure 15.14 Main - Reece

15.10 Thumbwheel Flowchart - Reece

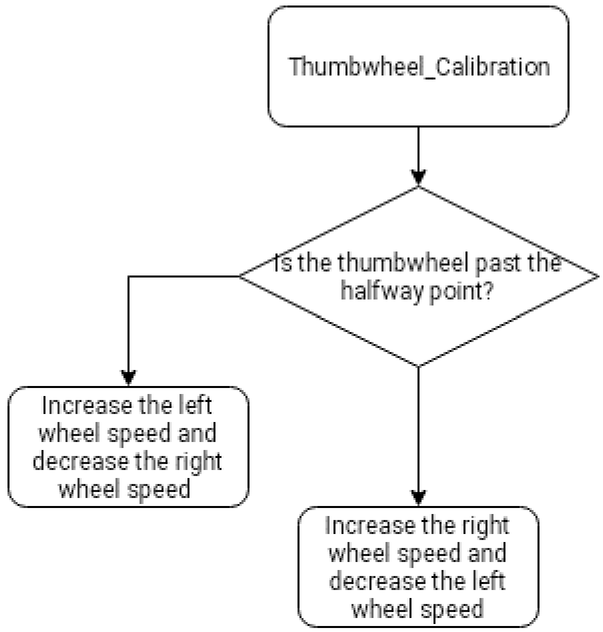

Figure 15.15 Thumbwheel Flow Chart - Reece

15.11 Serial ISR Flowchart - Reece

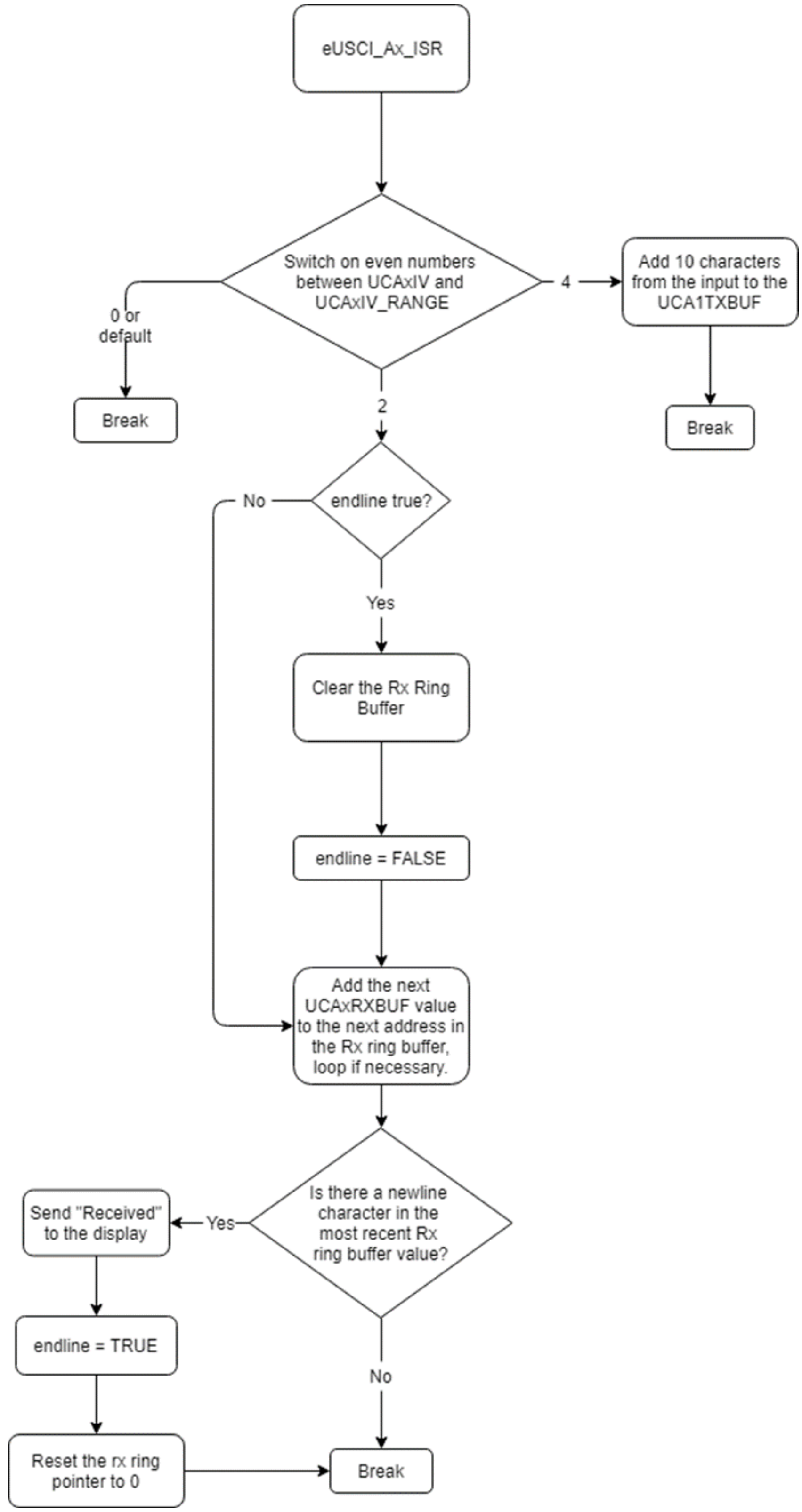

Figure 15.16 Serial ISR Flow Chart - Reece

15.12 Timers Flowchart - Reece

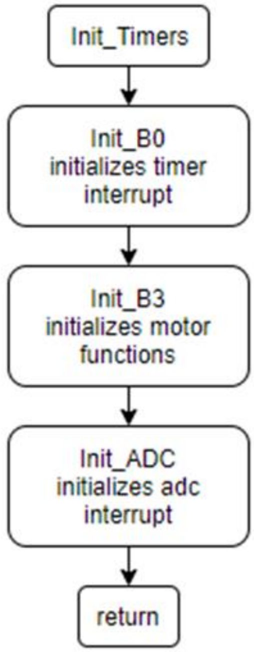

Figure 15.17 Init Timers Flow Chart - Reece

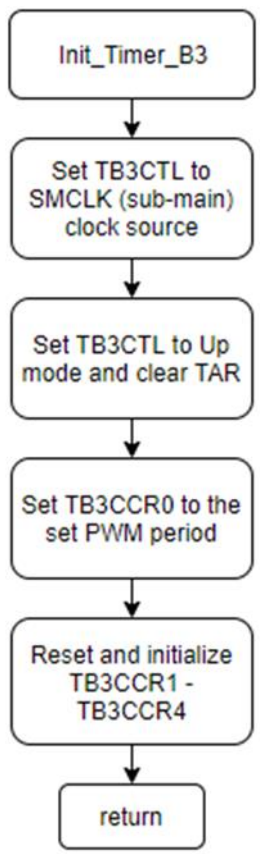

Figure 15.18 Timer_B3 Flow Chart - Reece

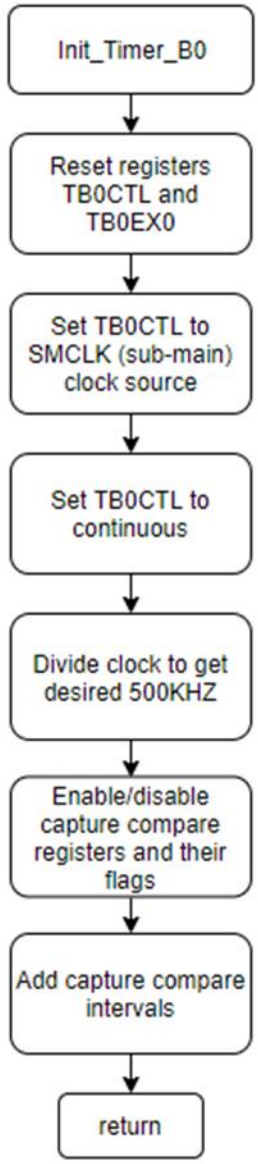

Figure 15.19 Timer_B0 Flow Chart - Reece

15.13 Main - Maria

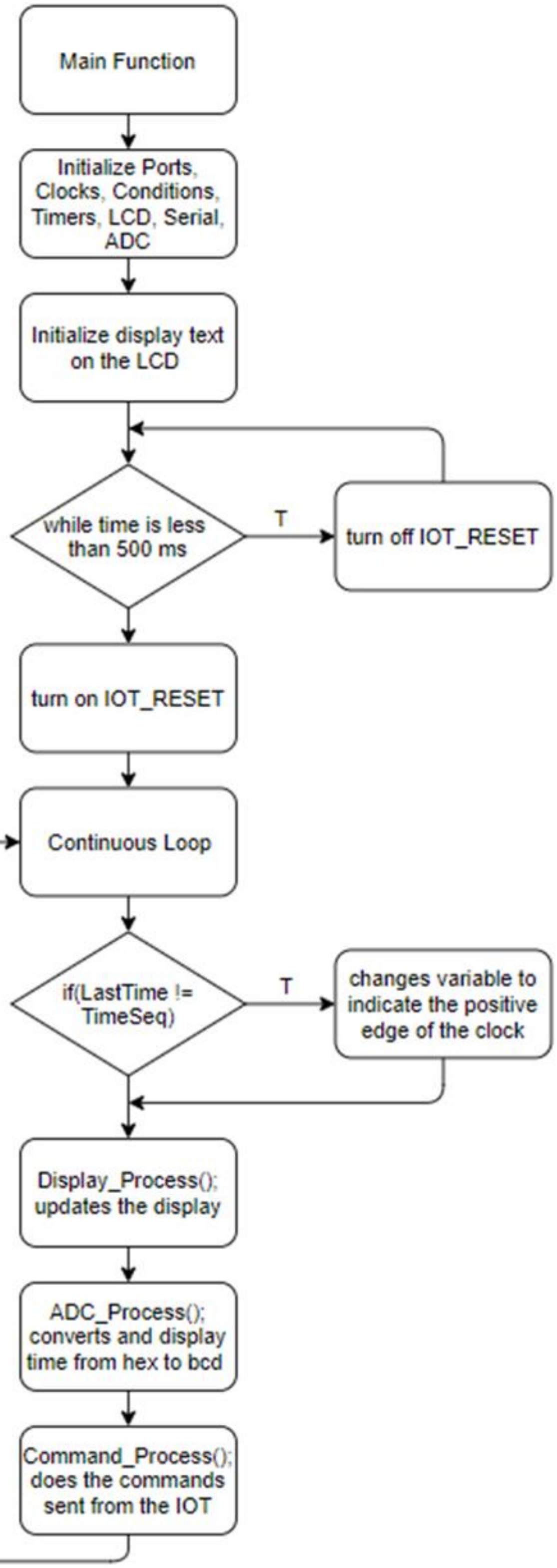

Figure 15.20 Main Flowchart - Maria

15.14 Command Flowchart - Maria

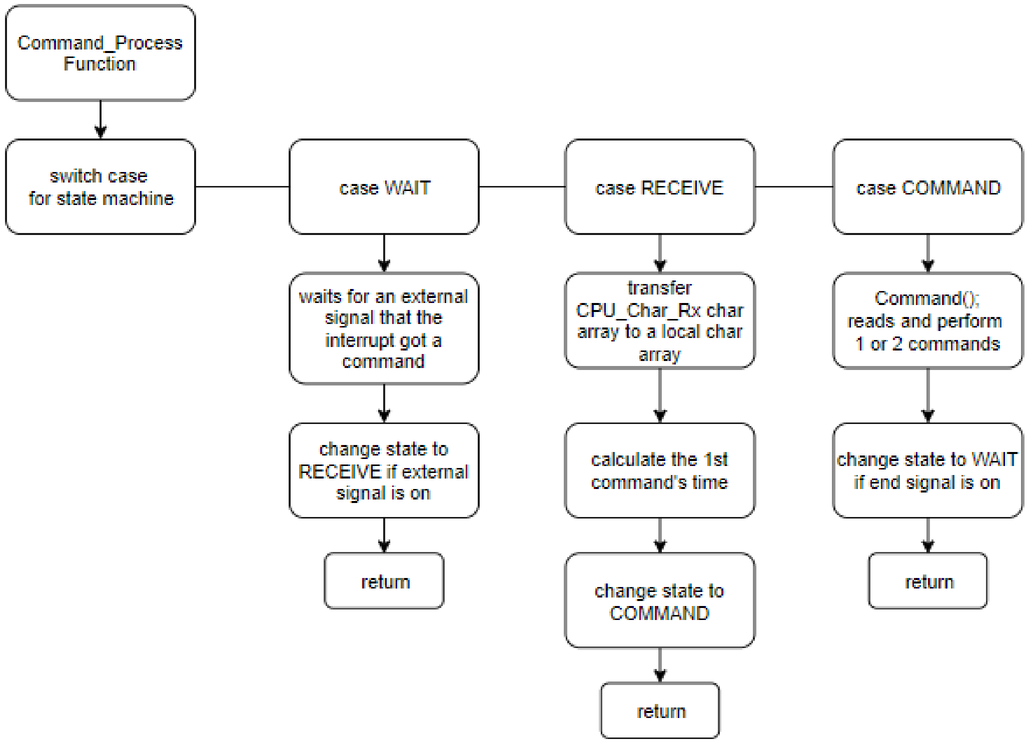

Figure 15.21 Command Process Flowchart - Maria

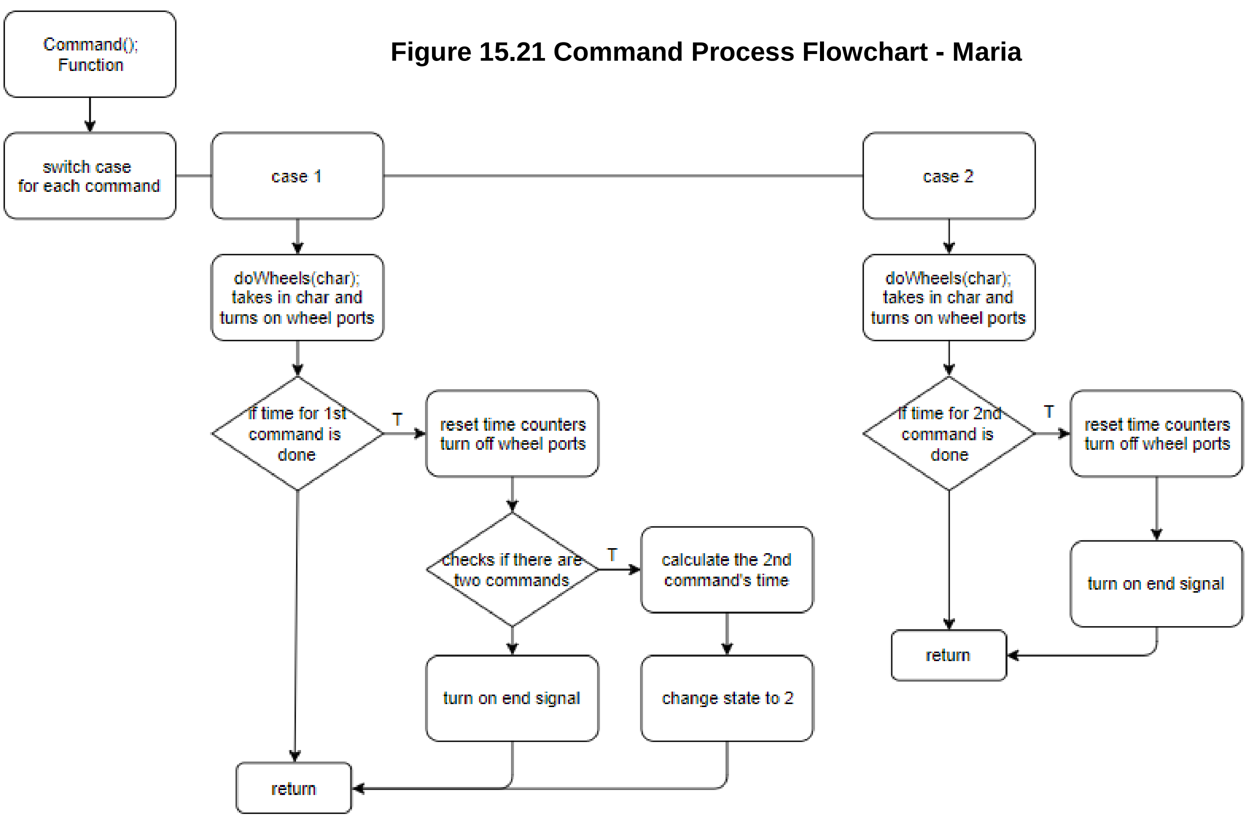

Figure 15.22 Command Flowchart - Maria

15.15 Serial Flowchart - Maria

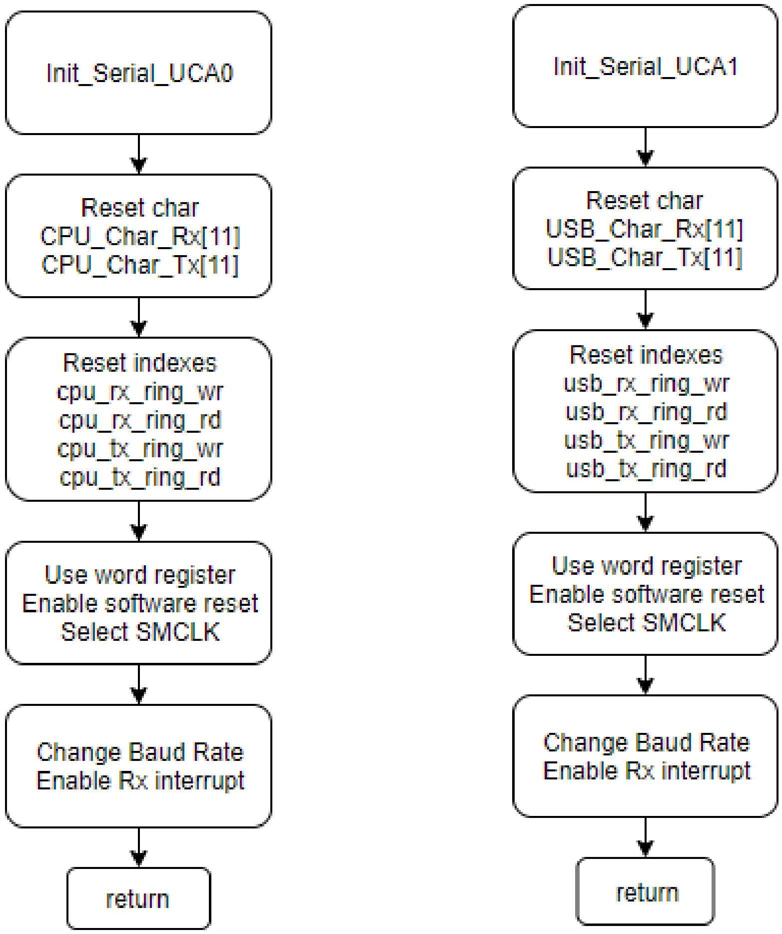

Figure 15.23 Serial Communication Flowchart - Maria

15.16 Line Flowchart - Maria

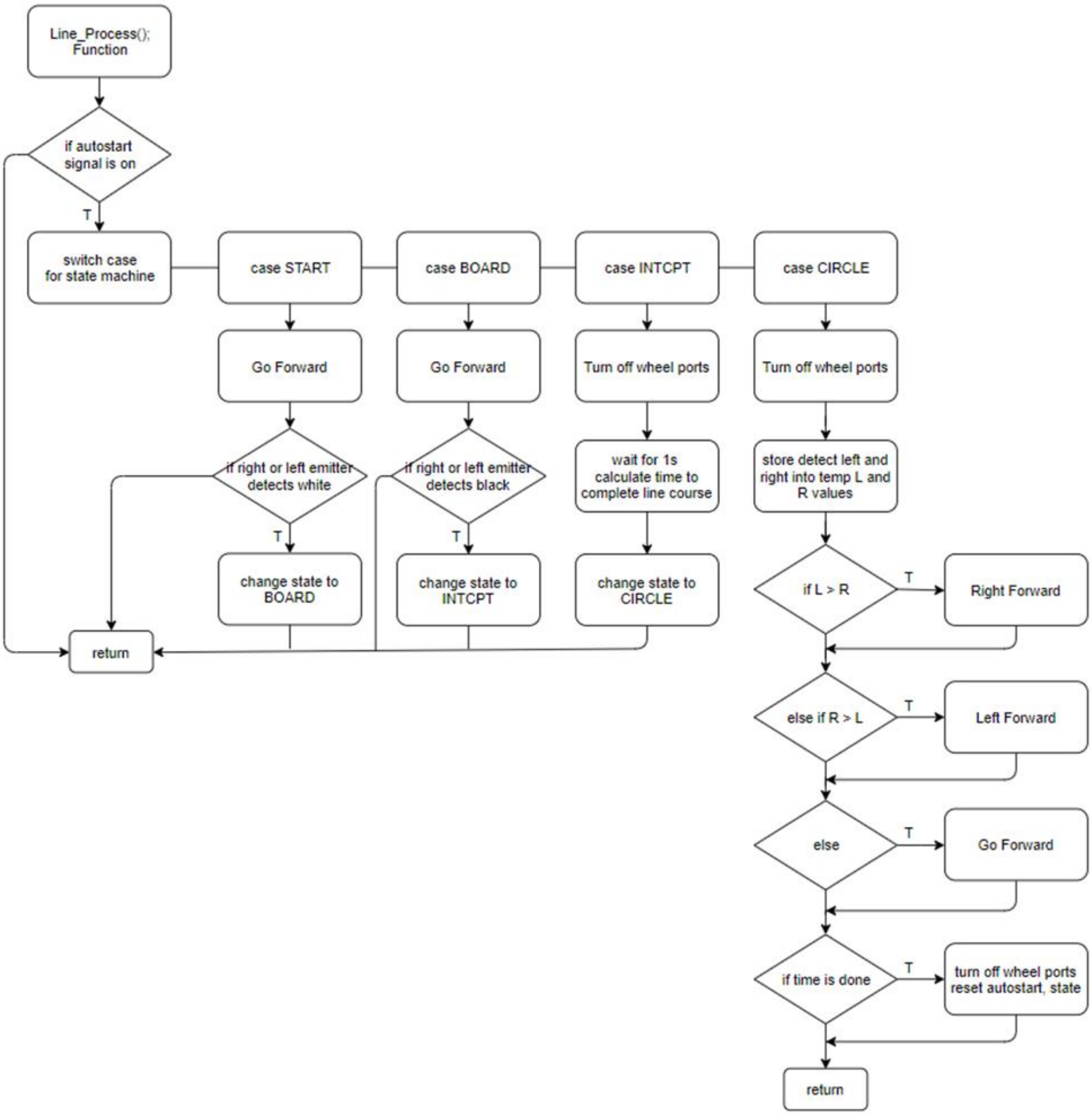

Figure 15.24 Line Following Flowchart - Maria

15.17 Main Flowchart - Adam

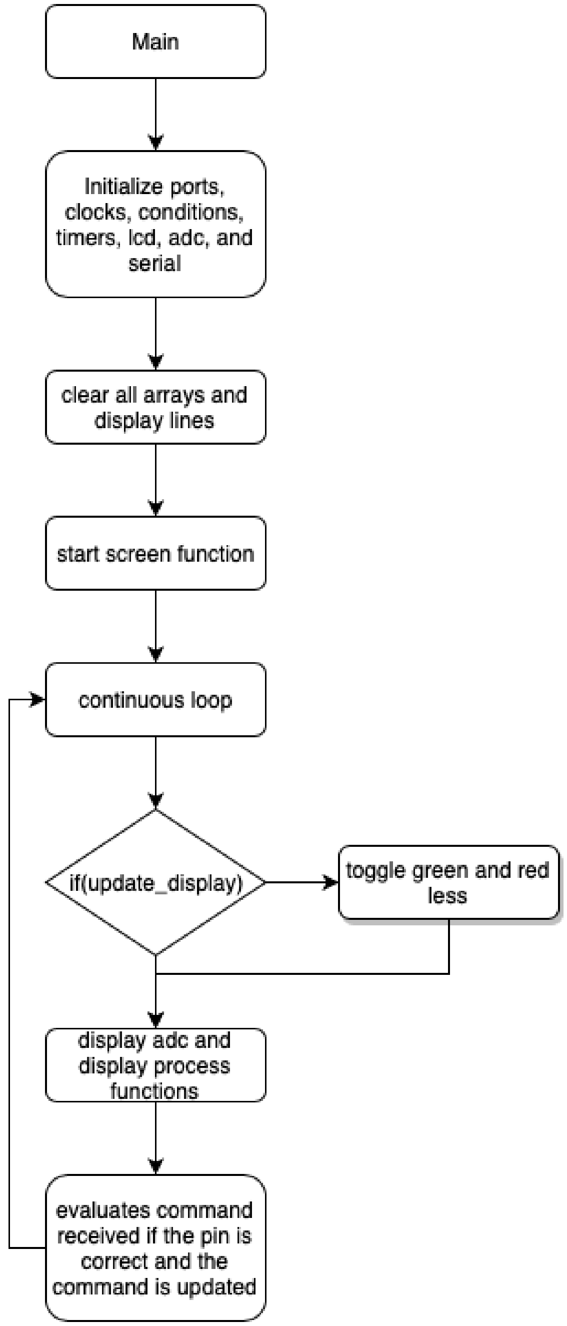

Figure 15.25 Main Flowchart - Adam

15.18 Intercept White Flowchart - Adam

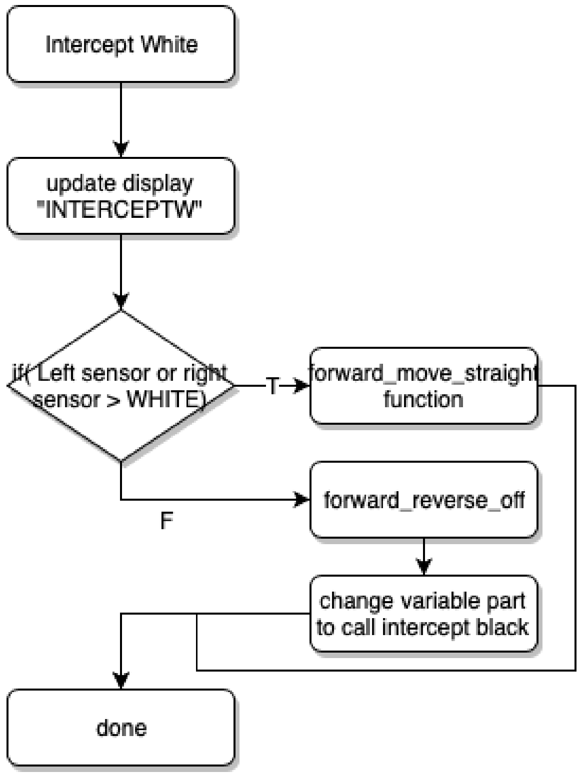

Figure 15.26 Intercept White Flowchart - Adam

15.19 Intercept Black Flowchart - Adam

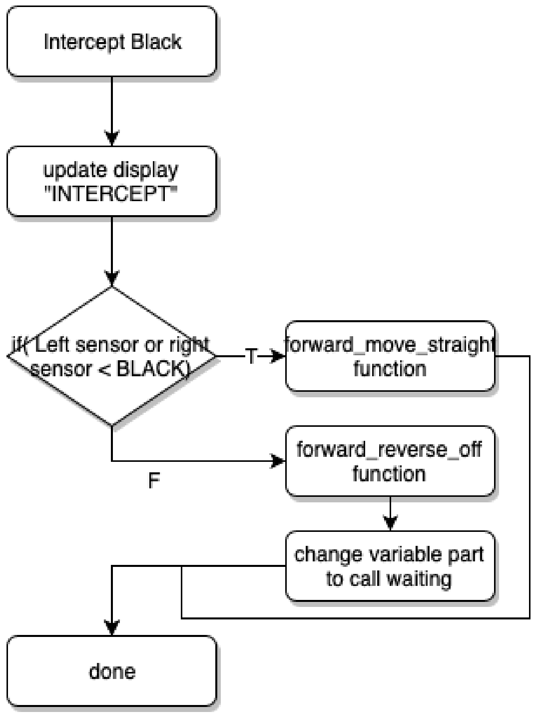

Figure 15.27 Intercept Black Flowchart - Adam

15.20 Follow Circle Flowchart - Adam

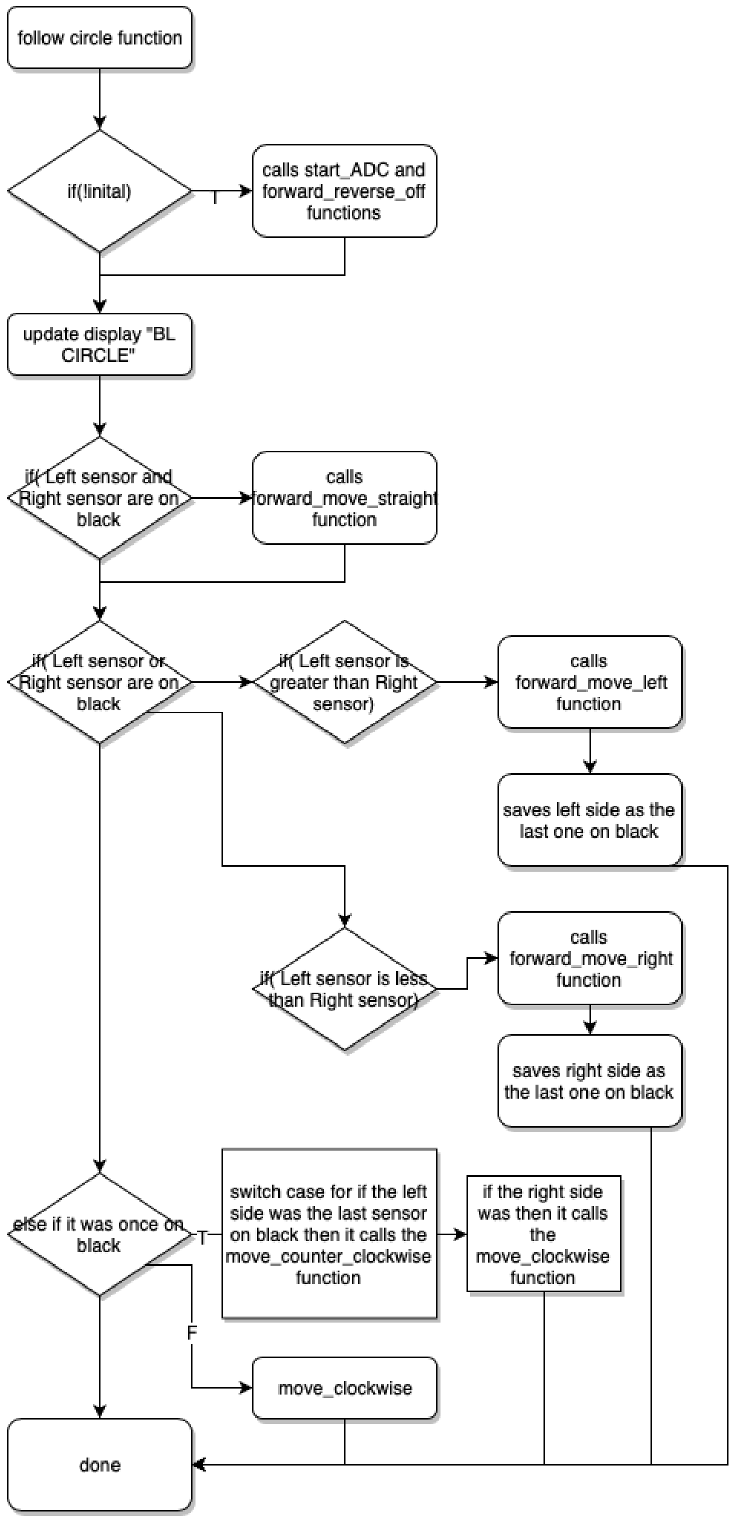

Figure 15.28 Follow Circle Flowchart - Adam

15.21 Interrupt Switch 1 Flowchart - Adam

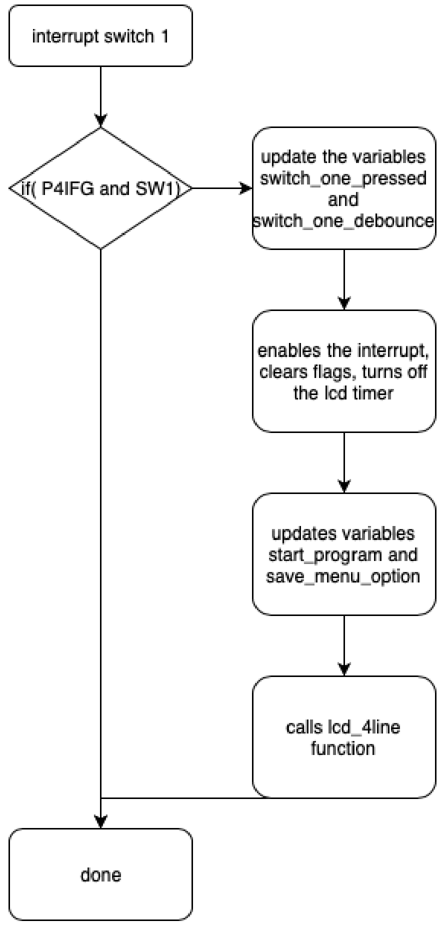

Figure 15.29 Interrupt Switch 2 Flowchart - Adam

16. Software Listing

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

16.1 ADC Code - Michael

```
//------------------------------------------------------------------------------
//
//  Description: This file contains the functions to initiate the ADC Registers
//  to set pins 2, 3, and 5 to measure the V_Detect_L, V_Detece_R, and V_Thumb
//  respectively.
//
//
//  Michael Barilla
//  Mar 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include "macros.h"
#include <string.h>

extern volatile unsigned char display_changed;
extern char display_line[FOURTH][COUNT_ELEVEN];
void Init_ADC(void){
  //----------------------------------------------------------------------------
  //    V_DETECT_L       (0x04) // Pin 2 A2
  //    V_DETECT_R       (0x08) // Pin 3 A3
  //    V_THUMB          (0x20) // Pin 5 A5
  //----------------------------------------------------------------------------

  //ADCCTL0 Register
    ADCCTL0  = RESET;     //Reset
    ADCCTL0 |= ADCSHT_2;        //16 ADC clocks
    ADCCTL0 |= ADCMSC;          //MSC
    ADCCTL0 |= ADCON;           //ADC ON

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

```
  //ADCCTL1 & ADCBUSY identifies a conversion is in process

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

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

    ADCIE &= ~ADCIE0;            // disable ADC conv complete interrupt
    ADCCTL0 &= ~ADCENC;          // ADC enable conversion.
    ADCCTL0 &= ~ADCSC;           // ADC start conversion.
}
//------------------------------------------------------------------------------
//
//  Description: This file contains the interrupt function for the ADC
//
//
//  Michael Barilla
//  Mar 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include "macros.h"

unsigned int channel;
unsigned int V_Detect_R[MAX_ARRAY];
unsigned int V_Detect_L[MAX_ARRAY];
unsigned int R_count;
unsigned int L_count;
unsigned int ADC_Thumb;


#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void){
  switch(__even_in_range(ADCIV,ADCIV_ADCIFG)){
    case ADCIV_NONE:
      break;
    case ADCIV_ADCOVIFG:        //When a conversion result is written to the ADCMEM0
```

```
                                    // before its previosu conversion result was read.
        break;
      case ADCIV_ADCTOVIFG:       //ADC conversion-time overflow
        break;
      case ADCIV_ADCHIIFG:        //Window comparator interrupt flags
        break;
      case ADCIV_ADCLOIFG:        //Window comparator interrupt flag
        break;
      case ADCIV_ADCINIFG:        //Window comparator interrupt flag
        break;
      case ADCIV_ADCIFG:          //ADCMEM0 memory register with the convesion result
        ADCCTL0 &= ~ADCENC;
        switch(channel++){
          case THUMB:
            ADC_Thumb = ADCMEM0;          //Capture Thumb ADC Value
            ADCMCTL0 &= ~ADCINCH_5;        //Set In Channel from 5 to 2
            ADCMCTL0 |= ADCINCH_2;
            break;
          case LDET:
            V_Detect_L[L_count++%MAX_ARRAY] = ADCMEM0;         //Capture VDETL ADC Value
            ADCMCTL0 &= ~ADCINCH_2;        //Set In Channel from 2 to 3
            ADCMCTL0 |= ADCINCH_3;
            break;
          case RDET:
            V_Detect_R[R_count++%MAX_ARRAY] = ADCMEM0;         //Capture VDETR ADC Value
            ADCMCTL0 &= ~ADCINCH_3;        //Set In Channel from 3 to 5
            ADCMCTL0 |= ADCINCH_5;
            channel = THUMB;              //Reset Channel
            break;
          default:
            break;
        }
        ADCCTL0 |= ADCENC;
        ADCCTL0 |= ADCSC;

        break;
      default:
        break;
    }
  }
```

16.2 Main Code - Michael

```
//------------------------------------------------------------------------------
//
```

```
//  Description: This file contains the Main Routine - "While" Operating System
//
//
//  Jim Carlson – Michael Barilla
//  Jan 2018
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------

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

  // Global Variables
volatile char slow_input_down;
extern char display_line[FOURTH][COUNT_ELEVEN];
extern char *display[FOURTH];
unsigned char display_mode;
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
unsigned int test_value;
char chosen_direction;
char change;
unsigned int Last_Time_Sequence;
unsigned int cycle_time;
unsigned int time_change;
unsigned int time_change2;




extern unsigned int menu;
extern unsigned int menu_select;




void main(void){
//------------------------------------------------------------------------------
// Main Program
// This is the main routine for the program. Execution of code starts here.
// The operating system is Back Ground Fore Ground.
//
```

```
//------------------------------------------------------------------------------
// Disable the GPIO power-on default high-impedance mode to activate
// previously configured port settings

  PM5CTL0 &= ~LOCKLPM5;

  Init_Ports();                         // Initialize Ports
  Init_Clocks();                        // Initialize Clock System
  Init_Conditions();                    // Initialize Variables and Initial Conditions
  Init_Timers();                        // Initialize Timers
  Init_LCD();                           // Initialize LCD
  Init_ADC();                           // Initialize ADC
  Init_Serial();                        // Initialize Serial
  //SeBegin_ADC();



// Place the contents of what you want on the display, in between the quotes
// Limited to 10 characters per line
//

  strcpy(display_line[FIRST_LINE], "          ");
  strcpy(display_line[SECOND_LINE], "          ");
  strcpy(display_line[THIRD_LINE], "          ");
  strcpy(display_line[FOURTH_LINE], "          ");

  Display_Update(FOURTH_LINE,SECOND_LINE,FIRST_LINE,FIRST_LINE);
  P5OUT |= IOT_RESET;
//------------------------------------------------------------------------------
// Begining of the "While" Operating System
//------------------------------------------------------------------------------
  while(ALWAYS) {                       // Can the Operating system run
    program();
    network();
    utilities();

  }

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

16.3 Network Code - Michael

```
//------------------------------------------------------------------------------
//
```

```
//  Description: This file contains the functions to update the network
//   and maintain it during the code operation
//
//
//  Michael Barilla
//  Apr 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------

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



char statusNTWK;
char IOTcommand[SMALL_RING_SIZE];
char IP[SMALL_RING_SIZE] = "                ";
char SSID[COUNT_ELEVEN] = "          ";

extern char display_line[FOURTH][COUNT_ELEVEN];

unsigned int sendPing = HIGH;
char PING[LARGE_RING_SIZE] = "AT+PING=www.pixar.com,3\r";

int dotCount = RESET;
int firstDot=RESET;
int secondDot=RESET;
int thirdDot=RESET;
int offset1 = RESET;
int offset3 = RESET;
int offset4 = RESET;

void network(void){
    //IOT_Reset();
    updateNetwork();
    ping();
}

void updateNetwork(void){
  int i = RESET;
```

```
switch(statusNTWK){
case NEW:
    statusNTWK = UPDATE;
    strcpy(IOTcommand,"AT+NSTAT=?\r");
    Transmit_Command_A0();
    break;
case DISPLAY:
  for(i=COUNT_ELEVEN; i > RESET; i--){
    if(SSID[i] == '\0') offset1++;
  }
  SSID[COUNT_TEN] = '\0';
  IP[COUNT_FIFTEEN] = '\0';

  dotCount = RESET;
  for(i=RESET; i<sizeof(IP); i++){
    if(IP[i] == '.') dotCount++;
    if(dotCount == FIRST && !firstDot) firstDot = i;
    else if(dotCount == SECOND && !secondDot) secondDot = i;
    else if(dotCount == THIRD && !thirdDot) thirdDot = i;
  }
  offset3 = MIDPOINT-firstDot;
  offset4 = thirdDot - MIDPOINT;

  /*strcpy(display_line[FIRST_LINE],"          ");
  strcpy(display_line[SECOND_LINE],"IP address");
  strcpy(display_line[THIRD_LINE],"          ");
  strcpy(display_line[FOURTH_LINE],"          ");

  for(i=RESET; i<COUNT_ELEVEN; i++){
    display_line[FIRST_LINE][i+(offset1>>FIRST)] = SSID[i];
  }
  for(i=RESET; i<SMALL_RING_SIZE; i++){
    if(i<secondDot) display_line[THIRD_LINE][i+offset3] = IP[i];
    if(i>secondDot) display_line[FOURTH_LINE][i-offset4] = IP[i];
  }*/
  statusNTWK = TCP;
case TCP:
  strcpy(IOTcommand, "AT+NSTCP=7898,1\r");
  Transmit_Command_A0();
  statusNTWK=WAIT;
  GREEN_LED_ON;
```

```
  default: break;
  }
}

void ping(void){
  if(sendPing && (statusNTWK == WAIT)){
    sendPing = LOW;
      for(int i=RESET; i<LARGE_RING_SIZE; i++)
        out_character_A0(PING[i]);
  }
}
```

16.4 Program Code - Michael

```
//------------------------------------------------------------------------------
//
//  Description: This file contains the functions to update the network
//   and maintain it during the code operation
//
//
//  Michael Barilla
//  Apr 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------

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



char driveCommand[SMALL_RING_SIZE];
extern char display_line[FOURTH][COUNT_ELEVEN];
char firstDrive[FOURTH];
char secondDrive[FOURTH];
unsigned int readyDrive;

unsigned int distance;
char direction;

char driveState = LOAD;
unsigned int startTimer;
unsigned int turnTimer;
unsigned int stopCount;
```

```
unsigned int blackLineR;
unsigned int blackLineL;
char stateBlackLine = START;
unsigned int capture;
unsigned int follow;
unsigned int oneTime;

int pwmL;
int pwmR;

extern unsigned int time_count_en;
extern unsigned int waitTime;
extern unsigned int endWait;

unsigned int left = LOW;
unsigned int right = HIGH;
extern unsigned int distance;
extern unsigned int endWait;
unsigned int searchState = RESET;

extern unsigned int doneTurn;
extern unsigned int doneDrive;
extern unsigned int turn_count;
extern unsigned int menu_select;
unsigned int waiting = HIGH;
extern unsigned int time_count_en;

extern int secondDot;
extern int offset3;
extern int offset4;
extern char IP[SMALL_RING_SIZE];
extern unsigned int midpoint;
extern unsigned int white;
extern unsigned int black;

void program(void){
  lcd_4line();
  if(waiting) {
    strcpy(display_line[FIRST_LINE],"  Waiting ");
    strcpy(display_line[SECOND_LINE],"For  Input");
    strcpy(display_line[THIRD_LINE], "          ");
    time_display();
  }
```

```
  if(!follow)
    drive();
  else
    blackLine();
}
```

16.5 Main Code - Adam

```
//------------------------------------------------------------------------------
//
//  Description: This file contains the Main Routine - "While" Operating System
//
//
//  Adam Casper
//  Jan 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------

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



// Function Prototypes
void main(void);
void Init_Conditions(void);
void Init_LEDs(void);

// Global Variables
volatile char slow_input_down;
extern char display_line[ROWS][COLS];
extern char *display[ROWS];
unsigned char display_mode;
extern volatile unsigned char display_changed;
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
extern unsigned int Last_Time_Sequence;
extern unsigned int cycle_time;
extern unsigned int time_change;
unsigned int test_value;
```

```
char chosen_direction;
char change;
unsigned int blink_count;
extern unsigned int switch_one_pressed;
extern unsigned int switch_two_pressed;
extern unsigned int V_det_L;
extern unsigned int V_det_R;
extern char display_V_det_L [COLS];
extern char adc_char[ROWS];
extern char display_V_det_R [COLS];
unsigned int calibration = RESET;
unsigned int frequency = RESET;
unsigned int action = RESET;
extern unsigned int save_menu_option = RESET;
extern unsigned int done_string;
extern unsigned int Half_Second;
extern unsigned int transmit_once;
extern unsigned int start_program;
char action_com[SMALL_RING_SIZE];
char action_usb[COLS];
extern unsigned int transmit;
extern volatile unsigned int iot_rx_ring_wr;
extern volatile unsigned int usb_rx_ring_wr;
extern volatile char USB_Char_Rx[SMALL_RING_SIZE];
extern volatile char IOT_Char_Rx[SMALL_RING_SIZE];
extern unsigned int second_count;
extern char response_array[RESPONSE];
extern unsigned int ip_done_string;
extern char ip_array[IPARRAYSIZE];
extern unsigned int com_pointer;
extern char com_array[IPARRAYSIZE];
extern unsigned int com_done_string;
unsigned int com_one_done = RESET;
int time = RESET;
unsigned int action_time_sync = LOW;
extern unsigned int second;
extern unsigned int start_timer;
extern unsigned int part;
unsigned int arrival_sqaure = LOW;
extern unsigned int timer_count;
extern unsigned int times;
extern unsigned int look_adc;
extern unsigned int ssid_ip_display_done;
extern unsigned int inital;
```

```
unsigned int exit_once = RESET;
extern unsigned int travel_away;
unsigned int last_part = GOBACK;

void main(void){
//------------------------------------------------------------------------------
// Main Program
// This is the main routine for the program. Execution of code starts here.
// The operating system is Back Ground Fore Ground.
//
//------------------------------------------------------------------------------
// Disable the GPIO power-on default high-impedance mode to activate
// previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;
  Init_Ports();                         // Initialize Ports
  Init_Clocks();                        // Initialize Clock System
  Init_Conditions();                    // Initialize Variables and Initial Conditions
  Init_Timers();                        // Initialize Timers
  Init_LCD();                           // Initialize LCD
  Init_ADC();
  Init_Serial_UCA0();
  Init_Serial_UCA1();
  clear_array();



// Place the contents of what you want on the display, in between the quotes
// Limited to 10 characters per line
//

  strcpy(display_line[FIRST_LINE], "          ");
  display_changed = TRUE;
  strcpy(display_line[SECOND_LINE], "          ");
  display_changed = TRUE;
  strcpy(display_line[THIRD_LINE], "          ");
  display_changed = TRUE;
  strcpy(display_line[FOURTH_LINE], "          ");
  display_changed = TRUE;
//  enable_display_update();
//  Display_Update(3,1,0,0);

  P5OUT |= IOT_RESET;               //Set IOT_RESET [HIGH]
  start_screen();
```

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

ping();
display_ssid_ip();

if(update_display){
  GREEN_LED_TOGGLE;
  RED_LED_TOGGLE;
}
display_ADC();
Display_Process();



if(((com_array[COMMAND_ONE] == '1') && (com_array[COMMAND_TWO] == '2') && (com_array[COMMAND_THREE] ==
'3') && (com_array[COMMAND_FOUR] == '4')) || (com_array[COMMAND_ONE] == '^')) {

start_timer = HIGH;

       if(!action_time_sync){
        part = RESET;
        second = RESET;
        Half_Second = RESET;
        action_time_sync = HIGH;
        }

    switch(action_usb[COL_TWO]) {
    case '^':
        strcpy(response_array, "I'm here  ");
        transmit_once = LOW;
        Transmit_Text_COM();
      break;
    case 'F':
        strcpy(response_array, "115,200   ");
        transmit_once = LOW;
        Transmit_Text_COM();
        UCA0BRW = FOUR;
        UCA0MCTLW = T115200;
      break;
    case 'S':
        strcpy(response_array, "9,600   ");
        transmit_once = LOW;
        Transmit_Text_COM();
```

```
        UCA0BRW = FIFTY_TWO;
        UCA0MCTLW = T9600;
      break;
  }
  if(!com_one_done){
  switch(com_array[COMMAND_FIVE]) {
    case 'F':
      End_ADC();
      strcpy(display_line[FIRST_LINE], " FORWARD  ");
      display_changed = TRUE;
      if(((com_array[COMMAND_SIX]) - HEX_TO_DEC) > second) {
        Forward_On();
      } else {
        strcpy(display_line[FIRST_LINE], "          ");
        display_changed = TRUE;
        Forward_Reverse_Off();
        second = RESET;
        Half_Second = RESET;
        com_one_done = HIGH;
      }
      break;
    case 'B':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "   BACK   ");
      display_changed = TRUE;
      if(((com_array[COMMAND_SIX]) - HEX_TO_DEC) > second) {
        Reverse_On();
      } else {
        strcpy(display_line[FIRST_LINE], "          ");
        display_changed = TRUE;
        Forward_Reverse_Off();
        second = RESET;
        Half_Second = RESET;
        com_one_done = HIGH;
      }
      break;
    case 'R':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "  RIGHT   ");
      display_changed = TRUE;
      if(((com_array[COMMAND_SIX]) - HEX_TO_DEC) > Half_Second) {
        Move_Counter_Clockwise();
      } else {
        strcpy(display_line[FIRST_LINE], "          ");
```

```
        display_changed = TRUE;
        Forward_Reverse_Off();
        second = RESET;
        Half_Second = RESET;
        com_one_done = HIGH;
      }
      break;
    case 'L':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "   LEFT   ");
      display_changed = TRUE;
      if(((com_array[COMMAND_SIX]) - HEX_TO_DEC) > Half_Second) {
        Move_Clockwise();
      } else {
        strcpy(display_line[FIRST_LINE], "          ");
        display_changed = TRUE;
        Forward_Reverse_Off();
        second = RESET;
        Half_Second = RESET;
        com_one_done = HIGH;
      }
      break;
    case 'P':
      Start_ADC();
      project();
      second = RESET;
      Half_Second = RESET;
      //com_one_done = HIGH;
      break;
    case 'T':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "ARRIVED   ");
      display_line[FIRST_LINE][COL_TEN] = (arrival_sqaure++%NINE) + ASCII;
      display_changed = TRUE;
      com_array[COMMAND_FIVE] = NULL;
      second = RESET;
      Half_Second = RESET;
      com_one_done = HIGH;
      break;
    case 'S':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "   STOP   ");
      display_changed = TRUE;
      Forward_Reverse_Off();
```

```
      inital = RESET;
      part = CALIBRATE;
      travel_away = RESET;
      second = RESET;
      Half_Second = RESET;
      com_one_done = HIGH;
      break;
    case 'E':
      if(!travel_away) {
      times = timer_count + TIMER_TEN;
      last_part = GOBACK;
      travel_away = HIGH;
      }
      switch(last_part) {
        case GOBACK:
        goback();
        break;
      case STOPP:
        stopp();
        break;
      }
      second = RESET;
      Half_Second = RESET;
      //com_one_done = HIGH;
      break;
    case 'Z':
      follow_circle();
      second = RESET;
      Half_Second = RESET;
      //com_one_done = HIGH;
      break;
    case 'A':
      look_adc = HIGH;
      second = RESET;
      Half_Second = RESET;
      //com_one_done = HIGH;
      break;
    case 'X':
      look_adc = RESET;
      ssid_ip_display_done = RESET;
      second = RESET;
      Half_Second = RESET;
      //com_one_done = HIGH;
      break;
```

```
    }
  }
    if(com_one_done) {
    switch(com_array[COMMAND_SEVEN]) {
      case 'F':
        End_ADC();
        strcpy(display_line[FIRST_LINE], " FORWARD  ");
        display_changed = TRUE;
        if(((com_array[COMMAND_EIGHT]) - HEX_TO_DEC) > second) {
          Forward_On();
        } else {
          strcpy(display_line[FIRST_LINE], "          ");
          display_changed = TRUE;
          Forward_Reverse_Off();
          clear_array();
          second = RESET;
          Half_Second = RESET;
          com_one_done = LOW;
        }
        break;
      case 'B':
        End_ADC();
        strcpy(display_line[FIRST_LINE], "   BACK   ");
        display_changed = TRUE;
        if(((com_array[COMMAND_EIGHT]) - HEX_TO_DEC) > second) {
          Reverse_On();
        } else {
          strcpy(display_line[FIRST_LINE], "          ");
          display_changed = TRUE;
          Forward_Reverse_Off();
          clear_array();
          second = RESET;
          Half_Second = RESET;
          com_one_done = LOW;
        }
        break;
      case 'R':
        End_ADC();
        strcpy(display_line[FIRST_LINE], "  RIGHT   ");
        display_changed = TRUE;
        if(((com_array[COMMAND_EIGHT]) - HEX_TO_DEC) > Half_Second) {
          Move_Counter_Clockwise();
        } else {
          strcpy(display_line[FIRST_LINE], "          ");
```

```
        display_changed = TRUE;
        Forward_Reverse_Off();
        clear_array();
        second = RESET;
        Half_Second = RESET;
        com_one_done = LOW;
      }
      break;
    case 'L':
      End_ADC();
      strcpy(display_line[FIRST_LINE], "   LEFT   ");
      display_changed = TRUE;
      if(((com_array[COMMAND_EIGHT]) - HEX_TO_DEC) > Half_Second) {
        Move_Clockwise();
      } else {
        strcpy(display_line[FIRST_LINE], "          ");
        display_changed = TRUE;
        Forward_Reverse_Off();
        clear_array();
        second = RESET;
        Half_Second = RESET;
        com_one_done = LOW;
      }
      break;
    default:
      clear_array();
      break;
  }
  }
}


//------------------------------------------------------------------------------
}


}
```

16.6 Intercept White Code - Adam

```
void interceptW(void) {
    strcpy(display_line[FIRST_LINE], "INTERCEPTW");
    display_changed = TRUE;
    if((V_det_L > DETECT_WHITE) || (V_det_R > DETECT_WHITE)) {
    Forward_Move_Straight();
```

```
    }
    else {
      on_white = HIGH;
      Forward_Reverse_Off();
      part = INTERCEPT;
    }
}
```

16.7 Intercept Black Code - Adam

```
void intercept(void) {
    strcpy(display_line[FIRST_LINE], "INTERCEPT ");
    display_changed = TRUE;
    if((V_det_L < DETECT_BLACK) || (V_det_R < DETECT_BLACK)) {
    Forward_Move_Straight();
    }
    else {
      on_black = HIGH;
      Forward_Reverse_Off();
      times = timer_count + TIMER_TWENTY;
      part = WAITINGC;
    }
}
```

16.8 Follow Black Line Code - Adam

```
void follow_circle(void) {
  if(!inital) {
  Start_ADC();
  Forward_Reverse_Off();
  inital = HIGH;
  }
  strcpy(display_line[FIRST_LINE], "BL CIRCLE ");
  display_changed = TRUE;

    if((V_det_L > ONTHE_BLACK) && (V_det_R > ONTHE_BLACK)) {
      Forward_Move_Straight();
    }
    else if((V_det_L > DETECT_BLACK) || (V_det_R > DETECT_BLACK)){
    if(V_det_L > V_det_R) {
      Forward_Move_Left();
      last_black = LEFT_SIDE;
    }
```

```
   if(V_det_L < V_det_R) {
      Forward_Move_Right();
      last_black = RIGHT_SIDE;
    }
  }
  else if(on_black){
    switch(last_black) {
      case LEFT_SIDE:
        Move_Counter_Clockwise();
        break;
      case RIGHT_SIDE:
        Move_Clockwise();
        break;
    }
    }
    else{
      Move_Clockwise();
    }
}
```

16.9 Interrupt Switch 1 code - Adam

```
//------------------------------------------------------------------------------
//
//  Description: This file contains the interrupt switch 1
//
//
//  Adam Casper
//  Jan 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include "functions.h"
#include "msp430.h"
#include "macros.h"
#include <string.h>

unsigned int switch_one_pressed;
unsigned int switch_one_debounce;
extern unsigned int save_menu_option;
extern unsigned int action;
unsigned int action_change = RESET;
extern unsigned int start_program;
```

```
#pragma vector = PORT4_VECTOR
__interrupt void switchP4_interrupt(void) {
  //switch 1
  if(P4IFG & SW1) {
    switch_one_pressed = HIGH;
    switch_one_debounce = ON;
    TB0CCTL1 |= CCIE;                // CCR1 enable interrupt
    P4IFG &= ~SW1;                    //Clear all P2.6 interrupt flags
    P4IE &= ~SW1;                     //P2.6 interrupt disabled
    P6OUT &= ~LCD_BACKLITE;
    TB0CCTL0 &= ~CCIE;                // turning off LCD timer

    start_program = HIGH;
    save_menu_option++;
    lcd_4line();

}
}
```

16.10 Main Code - Maria

```
//------------------------------------------------------------------------------
//  Description: This file contains the Main Routine - "While" Operating System
//  Maria Samia
//  Feb 1 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

// strings
extern char display_line[ROW4][COL11];
extern volatile unsigned char display_changed;
// Global Variables
volatile char slow_input_down;
unsigned char display_mode;
extern char *display[ROW4];
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
extern volatile unsigned int Time_Sequence = OFF;
extern volatile char one_time = OFF;
```

```
unsigned int test_value;
char chosen_direction;
char change;
// added globals
unsigned int Last_Time = OFF;
extern unsigned int onesectimer;

void main(void){
//------------------------------------------------------------------------------
// Main Program
// This is the main routine for the program. Execution of code starts here.
// The operating system is Back Ground Fore Ground.
//------------------------------------------------------------------------------
// Disable the GPIO power-on default high-impedance mode to activate
// previously configured port settings

        PM5CTL0 &= ~LOCKLPM5;
        Init_Ports();                        // Initialize Ports
        Init_Clocks();                       // Initialize Clock System
        Init_Conditions();                   // Initialize Variables and Initial Conditions
        Init_Timers();                       // Initialize Timers
        Init_LCD();                          // Initialize LCD
        Init_Serial_UCA1();
        Init_Serial_UCA0();

// Place the contents of what you want on the display, in between the quotes
// Limited to 10 characters per line
      strcpy(display_line[LINE0], "          ");
      update_string(display_line[LINE0], LINE0);
      strcpy(display_line[LINE1], "          ");
      update_string(display_line[LINE1], LINE1);
      strcpy(display_line[LINE2], "          ");
      update_string(display_line[LINE2], LINE2);
      strcpy(display_line[LINE3], "          ");
      update_string(display_line[LINE3], LINE3);
      display_changed = ON;             // change in display text array

//  Display_Update(3,1,0,0);
        while(onesectimer < DEC1){
                P5OUT &= ~IOT_RESET;
        }
        P5OUT |= IOT_RESET;
//------------------------------------------------------------------------------
// Begining of the "While" Operating System
```

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

        if (Last_Time != Time_Sequence){
                Last_Time = Time_Sequence;
                one_time = ON;
        }
        Display_Process();
        ADC_Process();
        Command_Process();
        Line_Process();
  }
}
```

16.11 Command Code - Maria

```
//------------------------------------------------------------------------------
//  Description: This file contains the IOT command process
//  Maria Samia
//  March 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

// strings
extern char display_line[ROW4][COL11];
extern volatile unsigned char display_changed;

extern volatile char CPU_Char_Rx[SMALL_RING_SIZE];
extern volatile char USB_Char_Rx[SMALL_RING_SIZE];
extern unsigned int onesectimer;
extern unsigned int doneCommand;
extern char com[COL11] = {OFF};
extern int temptimer1 = OFF;
extern int temptimer2 = OFF;
extern int end = OFF;
char part = WAIT;
char cases = 'A';

void Command_Process(void){
        switch(part){
                case WAIT:
                        if(doneCommand){ part = RECEIVE; }
```

```
                        break;
                case RECEIVE:
                        doneCommand = OFF;
                        Received();
                        part = COMMAND;
                        temptimer1 = onesectimer + getTime(com[LINE2]);
                        break;
                case COMMAND:
                        if(!end){ Command(); }
                        else{
                                end = OFF;
                                Pause();
                                part = WAIT;
                        } break;
        }
}

//------------------------------------------------------------------------------
// Command Function
// Input format: ^F2^B2
// This function can either take on or two commands
// Case A is the state for one command and case B is for the second
//------------------------------------------------------------------------------
void Command(void){
        switch(cases){
        case 'A':
                doWheels(com[LINE1]);
                if(onesectimer >= temptimer1){
                        com[LINE1] = 'P'; temptimer1 = OFF;
                        if(com[LINE3] == '^'){
                                cases = 'B'; Pause();
                                temptimer2 = onesectimer + getTime(com[LINE5]);
                        }
                        else{ end = ON; }
                } break;
        case 'B':
                doWheels(com[LINE4]);
                if(onesectimer >= temptimer2){
                        com[LINE4] = 'P';
                        temptimer2 = OFF;
                        end = ON; cases = 'A';
                } break;
        }
}
```

```
void Received(void){
        unsigned int i;
        for(i = OFF; i < COL11; i++){
                com[i] = CPU_Char_Rx[i];
        }
}

// function takes in a char and updates wheels
void doWheels(char command){
        switch(command){
                case 'F': Forward(); break;
                case 'B': Reverse(); break;
                case 'L': Left(); break;
                case 'R': Right(); break;
                default: Pause(); break;
        }
}

// function takes in a char and calculates desired time
int getTime(char time){
        int temp = time - '0';
        return temp*DEC2;
}
```

16.12 Serial Code - Maria

```
//------------------------------------------------------------------------------
//  Description: This file contains the Initialization of Serial
//  Maria Samia
//  March 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

// strings
extern char display_line[ROW4][COL11];
extern volatile unsigned char display_changed;
// extern signal for A1
extern volatile unsigned int usb_rx_ring_wr = OFF;
extern volatile unsigned int usb_rx_ring_rd = OFF;
extern volatile unsigned int usb_tx_ring_wr = OFF;
extern volatile unsigned int usb_tx_ring_rd = OFF;
```

```
extern volatile char USB_Char_Rx[SMALL_RING_SIZE] = {OFF};
extern volatile char USB_Char_Tx[SMALL_RING_SIZE] = {OFF};
// extern signal for A0
extern volatile unsigned int cpu_rx_ring_wr = OFF;
extern volatile unsigned int cpu_rx_ring_rd = OFF;
extern volatile unsigned int cpu_tx_ring_wr = OFF;
extern volatile unsigned int cpu_tx_ring_rd = OFF;
extern volatile char CPU_Char_Rx[SMALL_RING_SIZE] = {OFF};
extern volatile char CPU_Char_Tx[SMALL_RING_SIZE] = {OFF};
extern unsigned int getString = OFF;
extern unsigned int selBRATE = OFF;

void Init_Serial_UCA0(void){
        unsigned int j;
        for(j = OFF; j < SMALL_RING_SIZE; j++){
         CPU_Char_Rx[j] = POFF;
        }
        cpu_rx_ring_wr = OFF;
        cpu_rx_ring_rd = OFF;

        for(j = OFF; j < SMALL_RING_SIZE; j++){
         CPU_Char_Tx[j] = POFF;
        }
        cpu_tx_ring_wr = OFF;
        cpu_tx_ring_rd = OFF;

        // Configure UART 0
        UCA0CTLW0 = OFF;
        UCA0CTLW0 |= UCSWRST;
        UCA0CTLW0 |= UCSSEL__SMCLK;

        switch(selBRATE){
                case LINE0:    UCA0BRW = BAUD115200;
                                UCA0MCTLW = RATE115200; break;
                case LINE1: UCA0BRW = BAUD115200;                    // 115200
                                UCA0MCTLW = RATE115200; break;
                case LINE2: UCA0BRW = BAUD9600;                              // 9600
                                UCA0MCTLW = RATE9600; break;
        default: break;
        }

        UCA0CTLW0 &= ~UCSWRST;
        UCA0IE |= UCRXIE;
```

```
}

void Init_Serial_UCA1(void){
unsigned int i;
        for(i = OFF; i < SMALL_RING_SIZE; i++){
                USB_Char_Rx[i] = POFF;
        }
        usb_rx_ring_wr = OFF;
        usb_rx_ring_rd = OFF;

        for(i = OFF; i < SMALL_RING_SIZE; i++){
                USB_Char_Tx[i] = POFF;
        }
        usb_tx_ring_wr = OFF;
        usb_tx_ring_rd = OFF;

        getString = OFF;

        // Configure UART 0
        UCA1CTLW0 = OFF;
        UCA1CTLW0 |= UCSWRST;
        UCA1CTLW0 |= UCSSEL__SMCLK;

        UCA1BRW = BAUD115200;                                // basic
        UCA1MCTLW = RATE115200;

        UCA1CTLW0 &= ~UCSWRST;
        UCA1IE |= UCRXIE;
}
```

16.13 Line Code - Maria

```
//------------------------------------------------------------------------------
//  Description: This file contains black line follower code
//  Maria Samia
//  March 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"
#include <string.h>

extern volatile unsigned int V_Detect_L;
extern volatile unsigned int V_Detect_R;
extern volatile unsigned int textcount;
```

```
extern unsigned int timedelay;
extern unsigned int autostart;
unsigned int square = OFF;
unsigned int tempR = OFF;
unsigned int tempL = OFF;
char lstate = START;

void Line_Process(void){
        if(autostart){
                switch(lstate){
                        case START: Start(); break;
                        case BOARD: Board(); break;
                        case INTCPT: Intercept(); break;
                        case CIRCLE: Circle(); break;
                        default: break;
                }
        }
}

void Start(void){
        Forward();
        if(V_Detect_L < WHITE || V_Detect_R < WHITE){
                lstate = BOARD;
        }
}

void Board(void){
         Forward();
        if(V_Detect_L > BLACK || V_Detect_R > BLACK){
                Pause();
                timedelay = textcount + ONE_SEC;
                lstate = INTCPT;
        }
}

void Intercept(void){
        Pause();
        if(textcount >= timedelay){
                timedelay = textcount + ONE_CYCLE;
                lstate = CIRCLE;
        }
}

void Circle(void){
```

```
        Pause();
        tempL = V_Detect_L;
        tempR = V_Detect_R;
        if(tempL > tempR){
                Right_FON;
                Left_FOFF;
        }
        else if(tempL < tempR){
                Right_FOFF;
                Left_FON;
        }
        else{
                Forward();
        }
        if(textcount >= timedelay){
                Pause();
                lstate = START;
                autostart = OFF;
        }
}
```

16.14 Main Code - Reece

```
//------------------------------------------------------------------------------
//  Description: This file contains the Main Routine - "While" Operating System
//  Reece Neff
//  Mar 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.1)
//------------------------------------------------------------------------------

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

// Global Variables
volatile char slow_input_down;
extern char display_line[DISPLAY_ROW_LIMIT][DISPLAY_COLUMN_LIMIT];
extern char *display[DISPLAY_ROW_LIMIT];
extern unsigned char display_mode;
extern volatile unsigned char display_changed;
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
volatile unsigned int Time_Sequence;
volatile char one_time;
```

```
unsigned int test_value;
char chosen_direction;
char change;
unsigned int Last_Time_Sequence;
unsigned int cycle_time;
unsigned int time_change;
volatile unsigned char event;
char queue;
char state;
extern volatile unsigned int second_marker;

void main(void){
  //------------------------------------------------------------------------------
  // Main Program
  // This is the main routine for the program. Execution of code starts here.
  // The operating system is Back Ground Fore Ground.
  //
  //------------------------------------------------------------------------------
  // Disable the GPIO power-on default high-impedance mode to activate
  // previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;
  Init_Ports();                        // Initialize Ports
  Init_Clocks();                       // Initialize Clock System
  Init_Conditions();                   // Initialize Variables and Initial Conditions
  Init_Timers();                       // Initialize Timers
  Init_LCD();                          // Initialize LCD
  Init_ADC();                          // Initialize ADC
  Init_Serial_UCA0();                  // Initialize Serial Port for IOT
  // Serial_Process();
  // Place the contents of what you want on the display, in between the quotes
  // Limited to 10 characters per line
  //



  strcpy(display_line[FIRST_ROW], "  Waiting ");
  update_string(display_line[FIRST_ROW], FIRST_ROW);
//  strcpy(display_line[SECOND_ROW], " WOLFPACK ");
 // update_string(display_line[SECOND_ROW], SECOND_ROW);
  //strcpy(display_line[THIRD_ROW], "   Baud   ");
  //update_string(display_line[THIRD_ROW], THIRD_ROW);
  strcpy(display_line[FOURTH_ROW], "  115,200 ");
  update_string(display_line[FOURTH_ROW], FOURTH_ROW);
  display_changed = TRUE;
```

```
//  out_character('U');

  //  Display_Update(3,1,0,0);

  //----------------------------------------------------------------------------
  // Begining of the "While" Operating System
  //----------------------------------------------------------------------------
  while(ALWAYS) {                      // Can the Operating system run
/*    switch(Time_Sequence){
    case S1250:                        //
      if(one_time){
        Init_LEDs();
//        lcd_BIG_mid();
        display_changed = TRUE;
        one_time = FALSE;
      }
      Time_Sequence = INITIAL_TIME_SEQUENCE;             //
      break;
    case S1000:                        //
      if(one_time){
        GREEN_LED_ON;            // Change State of LED 5
        one_time = FALSE;
      }
      break;
    case S750:                         //
      if(one_time){
        RED_LED_ON;            // Change State of LED 4
        GREEN_LED_OFF;           // Change State of LED 5
        one_time = FALSE;
      }
      break;
    case S500:                         //
      if(one_time){
//        lcd_4line();
        GREEN_LED_ON;            // Change State of LED 5
        display_changed = TRUE;
        one_time = FALSE;
      }
      break;
    case  S250:                        //
      if(one_time){
        RED_LED_OFF;           // Change State of LED 4
```

```
        GREEN_LED_OFF;            // Change State of LED 5
        one_time = FALSE;
      }
      break;                          //
    default: break;
    }
*/

///    ADC_Display();

    Thumbwheel_Calibration();
//    Follow_Black_Line();

    ADC_Display();

    Follow_Black_Line();

    Millisecond_Timer();

    Display_Process();





//    Millisecond_Timer();
/*
    if(Last_Time_Sequence != Time_Sequence){
      Last_Time_Sequence = Time_Sequence;
      cycle_time++;
      time_change = TRUE;
    }
    //    Wheels();
*/
  }
  //-----------------------------------------------------------------------------
}
```

16.15 Thumbwheel Code - Reece

```
void Thumbwheel_Calibration(void){
  if (ADC_THUMB_MEM < THRESHOLD)
  {
    Left_Speed = FIFTY_PERCENT_DUTY_CYCLE + THRESHOLD - ADC_THUMB_MEM;
    Right_Speed = FIFTY_PERCENT_DUTY_CYCLE + ADC_THUMB_MEM - THRESHOLD;
```

```
  }
  else
  {
    Right_Speed = FIFTY_PERCENT_DUTY_CYCLE + ADC_THUMB_MEM - THRESHOLD;
    Left_Speed = FIFTY_PERCENT_DUTY_CYCLE + THRESHOLD - ADC_THUMB_MEM;
  }
}
```

16.16 Serial ISR - Reece

```
#pragma vector=EUSCI_A0_VECTOR
__interrupt void eUSCI_A0_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA0IV,UCAxIV_RANGE)){
  case 0: // Vector 0 - no interrupt
    break;
  case 2: // Vector 2 – RXIFG
    if(endline == TRUE)
    {
      int i;
      for(i=INITIAL; i<SMALL_RING_SIZE; i++){
        IOT_Char_Rx[i] = INITIAL; // IOT Rx Buffer
      }
      endline = FALSE;
    }
    temp = iot_rx_ring_wr++;
    IOT_Char_Rx[temp] = UCA0RXBUF; // RX -> IOT_Char_Rx character
    if (iot_rx_ring_wr >= (sizeof(IOT_Char_Rx))){
      iot_rx_ring_wr = BEGINNING; // Circular buffer back to beginning
    }
    if (IOT_Char_Rx[temp] == 'E' && IOT_Char_Rx[temp-ONE] == ESC)
    {
      //strcpy(display_line[FIRST_ROW], " Received ");
      //update_string(display_line[FIRST_ROW], FIRST_ROW);
      //display_changed = TRUE;
      iot_rx_ring_wr = BEGINNING;
      Commands();
      endline = TRUE;
    }
    if(UCA0RXBUF == 'r' || IPCHK0 == TRUE)
    {
      IPCHK0 = TRUE;
      if(UCA0RXBUF == '=' || IPCHK1 == TRUE)
      {
        IPCHK1 = TRUE;
        if(UCA0RXBUF != ' ' && UCA0RXBUF != '=')
        {
          temp = iot_ip_ring_wr++;
          IP_Char_Rx[temp] = UCA0RXBUF; // RX -> IP_Char_Rx character
          if(iot_ip_ring_wr >= (sizeof(IP_Char_Rx))){
            iot_ip_ring_wr = ONE; // Circular buffer back to beginning
          }
          if(UCA0RXBUF == '.')
          {
            Period1++;
          }
          if(Period1 == DOUBLE)
          {
            strcpy(display_line[SECOND_ROW], (char *)IP_Char_Rx);
            update_string(display_line[SECOND_ROW], SECOND_ROW);
            strcpy(display_line[FIRST_ROW], "   ncsu   ");
            update_string(display_line[FIRST_ROW], FIRST_ROW);
            Period1 = INITIAL;
            display_changed = TRUE;
            iot_ip_ring_wr = DOUBLE;
            for(i=INITIAL; i<DISPLAY_COLUMN_LIMIT-ONE; i++){
              IP_Char_Rx[i] = ' '; // USB Rx Buffer
```

```
            }
            IP_Char_Rx[DISPLAY_COLUMN_LIMIT-ONE] = INITIAL;
            IP_Char_Rx[ONE] = UCA0RXBUF;
          }
        }
        else if(UCA0RXBUF == ' ')
        {
          strcpy(display_line[THIRD_ROW], (char *)IP_Char_Rx);
          update_string(display_line[THIRD_ROW], THIRD_ROW);
          IPCHK0 = FALSE;
          IPCHK1 = FALSE;
          Period1 = INITIAL;
          display_changed = TRUE;
          iot_ip_ring_wr = ONE;
        }
      }
    }


    temp = UCA0RXBUF;
    UCA1TXBUF = temp;
    break;
  case 4: // Vector 4 - TXIFG
    switch(UCA0_index++){
    case 0: //
    case 1: //
    case 2: //
    case 3: //
    case 4: //
    case 5: //
    case 6: //
    case 7: //
    case 8: //
      //      UCA0TXBUF = test_command[UCA0_index];
      //      break;
    case 9: //
      //      UCA0TXBUF = 0x0D;
      //      break;
    case 10: // Vector 0 - no interrupt
      UCA0TXBUF = IOT_Char_Tx[UCA0_index];
      //     UCA0TXBUF = 0x0A;
      break;
    default:
      UCA0IE &= ~UCTXIE; // Disable TX interrupt
      break;
    } break;
  default: break;
  }
}
```

16.17 Timer - Reece

```
void Init_Timer_B0(void) {
  //------------------------------------------------------------------
  // Timer B0 initialization sets up both B0_0, B0_1-B0_2 and overflow
  TB0CTL = RESET_REGISTER; // Clear TB0 Control Register
  TB0EX0 = RESET_REGISTER; // Clear TBIDEX Register
  TB0CTL |= TBSSEL__SMCLK; // SMCLK source
  TB0CTL |= MC__CONTINOUS; // Continuous up to 0xFFFF and overflow
  TB0CTL |= ID__2; // Divide clock by 2
  TB0EX0 |= TBIDEX__8; // Divide clock by an additional 8
  TB0CTL |= TBCLR; // Resets TB0R,
  // Capture Compare 0
  //#pragma vector = TIMER0_B0_VECTOR
  // Capture Compare 0
  TB0CCR0 = TB0CCR0_INTERVAL; // CCR0
  TB0CCTL0 |= CCIE; // CCR0 enable interrupt
  // Capture Compare 1,2, Overflow
  //#pragma vector = TIMER0_B1_VECTOR
  // Capture compare 1
  TB0CCR1 = TB0CCR1_INTERVAL;// CCR1
  TB0CCTL1 &= ~CCIE; // CCR1 disable interrupt
```

```
  // Capture compare 2
  // TB0CCR2 = TB0CCR2_INTERVAL;// CCR2
  // TB0CCTL2 |= CCIE; // CCR2 enable interrupt
  // Overflow
  TB0CTL &= ~TBIE; // Disable Overflow Interrupt
  TB0CTL &= ~TBIFG; // Clear Overflow Interrupt flag
}


void Init_Timer_B3(void) {
//------------------------------------------------------------------------------
// SMCLK source, up count mode, PWM Right Side
// TB3.1 P6.0 R_FORWARD
// TB3.2 P6.1 L_FORWARD
// TB3.3 P6.2 R_REVERSE
// TB3.4 P6.3 L_REVERSE
//------------------------------------------------------------------------------
 TB3CTL = TBSSEL__SMCLK; // SMCLK
 TB3CTL |= MC__UP; // Up Mode
 TB3CTL |= TBCLR; // Clear TAR

 TB3CCR0 = WHEEL_PERIOD; // PWM Period

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

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

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

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

17. Conclusion

The largest lesson learned was that the construction of this device had to be meticulous from the start. Throughout the process it was essential that we tested each component using a variety of tests, the debugging watch window and tools like multimeters. That ensured a solid foundation for the system. Even with all of these tests completed, some bugs still arose because of the brand-new FRAM (MSP430) board that we were using in combination with the IOT modules. For example, the issue where the IOT module would short out connecting the power to ground directly or the issue with the external power supply that would not allow the board to turn on upon initially flipping the switch.

Once the device was built and the major bugs were addressed the focus had to switch to an optimization mentality. To ensure that the device performed as expected, maintaining a sufficient battery level for as long as possible was a top priority. That meant that the system had to consume as minimal power possible which would prolong the device's lifespan.

Another major lesson learned through the production of this system was ensure that the device was created to be robust and consistent across various environmental and situational factors. Building in different commands for the direction to find the black line or a low power mode when the batteries were starting to lose their strength. A similar issue that arose was the difficulty of controlling the car using the IOT module on a private wireless network with ample disturbance from other devices. This could be remedied somewhat within the code to result in a more accurate with steering throughout the course.

In summary, the lessons that we learned during the production of this car were working with new technology like our FRAM boards and IOT modules will create unexpected bugs that have to be addressed (i.e. power on bug and IOT shorting problem), creating minimal power consumption devices is essential to their success, being proactive when building the system is important to create a robust and consistent device that will consistently perform its specified task and developing various methods of problem solving for both hardware and software to ensure the programming or soldering is done satisfactory.