Transpato

The Autonomous Car Solution for Shipping Centers

Team 2

Team Members

Steven Yan
Alec Probst
Bennett James
Nolan Joyce

Originator:AuthorT			
Checked:	Released:		
Filename: Project 10 Writeup Draft.docx			
Title: **Transpato** **The Autonomous Car Solution for Shipping Centers**			
Date: **05/01/2019**	Document Number: **0-0000-000-0000-01**	Rev: **5D**	Sheet: **1 of 73**

Revision	Description
1A	Project 02 Test Process Section added, hardware pictures added,
1B	Helped write Scope, created diagrams and edited articles
1C	Took pictures, created abbreviation and block diagrams
1D	Added Hardware Section, Formatted document, formatted pictures, added Scope.
2A	Revised small formatting mistakes, added to hardware section, added abbreviations, changed revision table
2B	Created the rest of the diagrams, wrote paragraphs for section 3 diagrams, worked on abbreviations
2C	Formatting, Pictures, miscellaneous revisions.
2D	Cover Page, Table of Contents, Formatting.
3A	Created Flowchart for Interrupts, Added code for Interrupt and ADC configuration,
3B	Edited all changes to figures and abbreviations. Created Timer Function description and Flowchart
3C	Added ports description, code, and flowchart
3D	Created flowchart for main, revised scope, added description for main. Attached code
4A	Created flowchart for Serial Communications, fixed code issue from revision 3D
4B	Added descriptions and code for the serial communication interrupts and initialization.
4C	Reworked figure numbers, compiled new revisions into final, and removed unnecessary details.
4D	Added tests for blackline following, serial communications. Revised code.
5A	Power Analysis, added code/description/figures for personal main.c, irledconfig.c, and switches.c files, fixed overall flowcharts/diagrams to add in communication, added IOT overview section
5B	Fixed tables of figures, updated scope, half to full hbridge description, flowcharts and descriptions for functions and a main function.
5C	Added Power Analysis, added Conclusion, added main flowchart, network config flowchart, menu flowchart, and forward flowchart, added relevant code.
5D	Added serial communication overview, hardware, and software descriptions. Added main function code and other function codes with corresponding flowcharts

Table of Contents

Table of Figures

1. Scope

We are experiencing a changing world. Malls are closing down across America due to lack of foot traffic business. Instagram influencers are making a living by promoting online clothing brands. The number of online shoppers increases every year as trust and reliability increases within the prominent platforms. Storefront locations are no longer the key to profitability. Now, the most efficient and autonomous shipping centers and distribution centers are the companies who are the most profitable.

We have developed the silver bullet for these companies: introducing Transpato the autonomous car solution for shipping centers. This robot was specifically engineered to thrive within warehouses. With the simple addition of black lines on the warehouse floor, Transpato robots are able to accomplish everything a human could do, but much faster. Transpato is also equipped with a top of the line IOT module in which commands can be sent using serial communications between Transpato and the user allowing for remote access to the movements and tasks it will perform. No more staffing headaches, no more variability: only your company, Transpato, and cash flow. Transpato can move packages around store houses by following the lines on the ground, distributing items faster and quicker than human counterparts.

To accomplish all this, Transpato is equipped with modern computing hardware, robust infrared emitter and detector sensors, highly-controllable motors, and a bright LCD display with an intuitive user interface. All this and more packed in a sleek and translucent chassis to allow for minimal disturbance to your existing business.

2. Abbreviations

Abbreviation	Definition
LED	Light Emitting Diode
LCD	Liquid Crystal Display
SW1	Power Switch
CPU	Central Processing Unit
N-FET	N-Type Field Effect Transistor
JMP	Wire Jumper
FRAM	Ferroelectric Random Access Memory
ADC	Analog to Digital Converter
ISR	Interrupt Service Routine
PWM	Pulse Width Modulation
TX	Transmit
RX	Receive

3. Overview

Listed below are the large systems inside of Transpato. The following sections will explain in detail, these large systems.

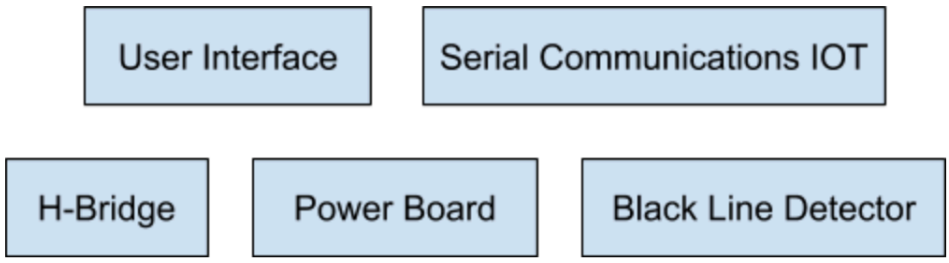

Figure 3.1: System Interface

3.1. User Interface

The user interface consists of a switch SW1 and two buttons S1 and S2. Battery power is supplied across the LCD display once switch SW1 is turned into the on position. The buttons work as input for the processor.

Figure 3.2: User Interface

3.2. Power Board

The Power Board contains the Battery Input and the Processor Power Connectors. The Battery Input is where the battery is connected to the board. The buck-boost converter is responsible for maintaining a steady 5 V to be delivered to the MSP430. The Power Board converts the battery power into power that is transferred through the Processor Power Connectors to the Processor.

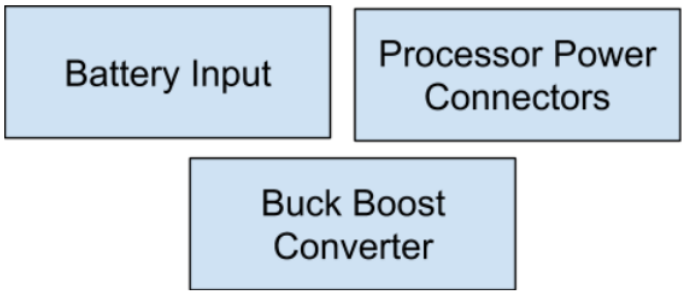

Figure 3.3: Power Board

3.3. H-Bridge

An H bridge is a fairly simply electronic circuit that switches the polarity of a voltage applied to a load. The H-Bridge contains the MOSFETS, Left Motor Connector, and Right Motor Connector. The MOSFETS allow the processor to control the direction of the Transpato. The Left and Right Motor Connectors are where the Motors attach to the Transpato and supply power to them.

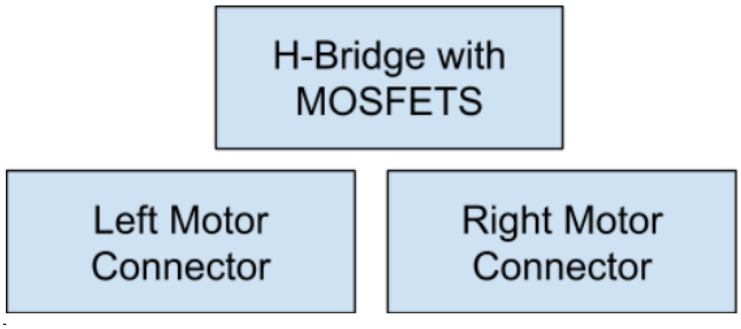

Figure 3.4: H-Bridge

3.4. Black Line Detector

The emitter is an IR LED in which the left and right detector are angled inwards towards. The values for the left and right detector are calibrated based upon the when on the line and off the line. The left and right detector will then communicate with the motors to stay on the line.

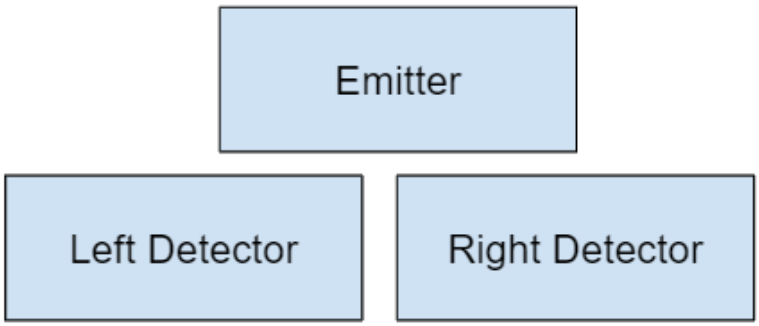

Figure 3.5: Black Line Detector

3.5. Serial Communication

The Serial Communications IOT contains the IOT Connector which allows the IOT Module to interface with the processor, the IOT Board which the IOT module attaches to, and the IOT Module itself which deals with communication from the processor.

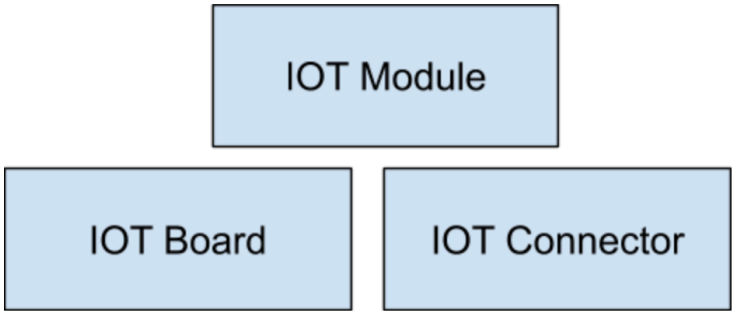

Figure 3.6 Serial Communication

4. Hardware

We are using the Texas Instruments MSP430FR2355 LaunchPad Development Kit, which is state-of-the-art technology that includes everything needed to start developing on the MSP430FRx FRAM microcontroller platform. The board has easy-to-use debugging features, 2 onboard buttons and 2 LEDs, Grove sensor connectors, and an ambient light sensor.

4.1. User Interface

The first thing we did was break the board into the three separate pieces that make up the board. We then ran the Control Board through the pick and place machines to include the correct resistors and capacitors. Then we hand soldered the larger components onto the board: tantalum capacitor, an inductor, a 5-pin integrated circuit, a switch, a potentiometer, multiple connectors, and a backlit LCD display.

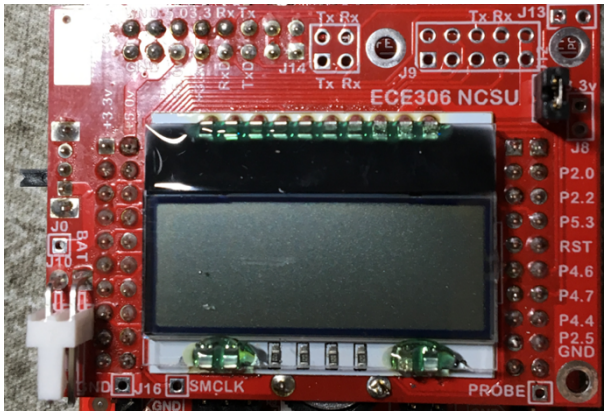

Figure 4.1: Control Board (Top)

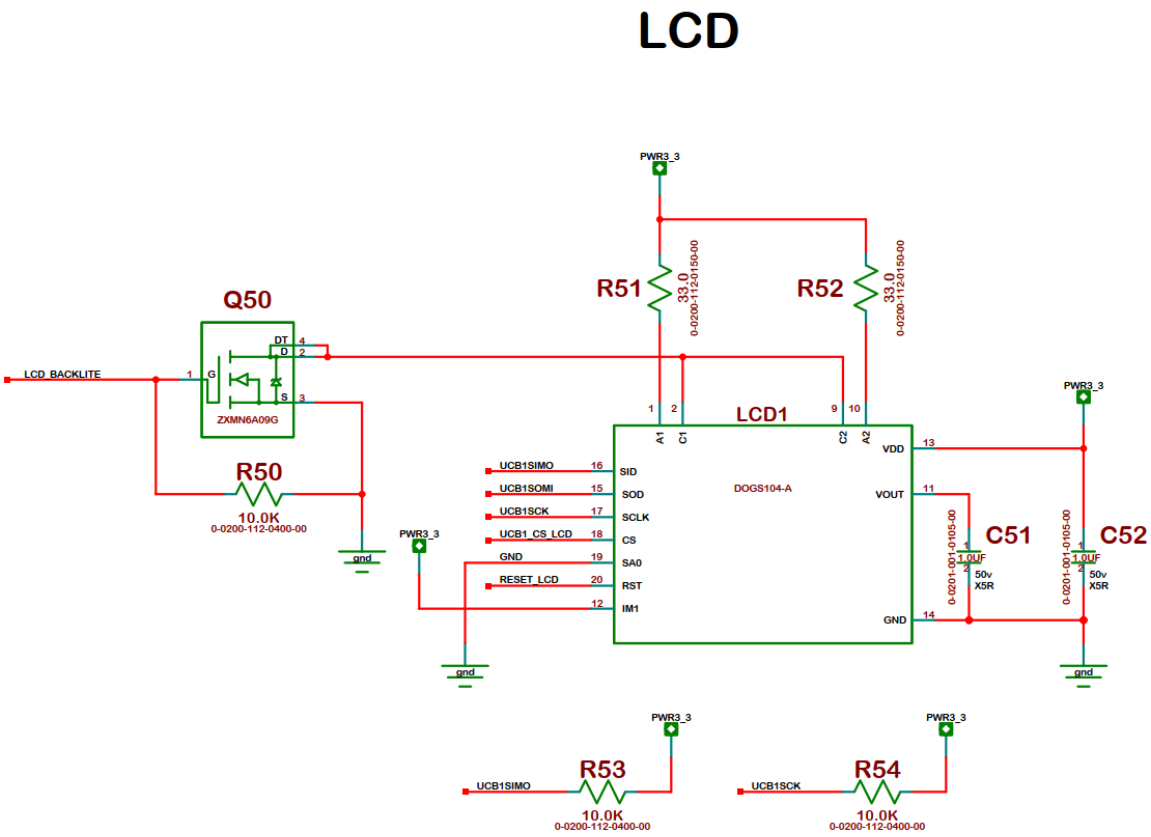

Figure 4.2: LCD Schematic

4.2. Power Board

The vehicle is powered by a battery pack containing 4 AA batteries. The batteries are configured in series. The batteries shown in the figures below are Eneloop cells manufactured by Panasonic. They have a nominal voltage of 1.2 V and under full charge they sit at around ~1.4 volts.

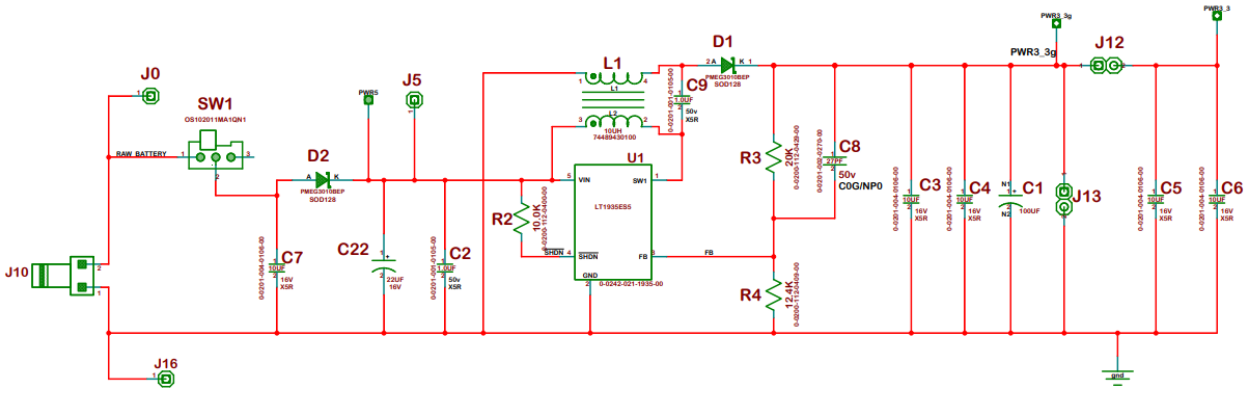

Figure 4.3: Power Board Schematic

Figure 4.4: Battery Pack

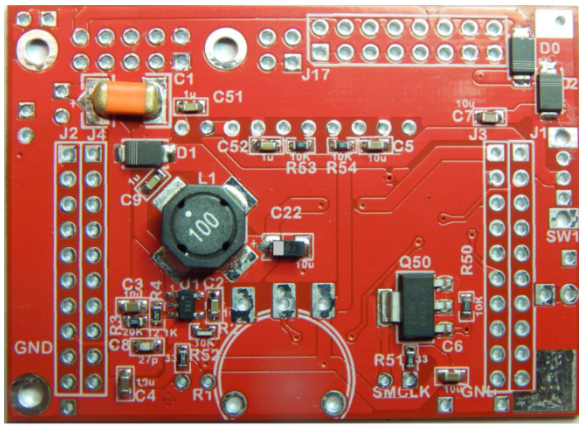

Figure 4.5: Buck Boost Converter

4.3. H-Bridge

The Full H-Bridge was added in project 05 which gives control over the motors in forward and reverse. We used nine N-type Mosfets and four P-type Mosfets. The N-type Mosfets were installed at device locations Q1, Q11, Q12, Q21, Q22, Q31, Q32, Q41, Q42. The P-type Mosfets wer installed at location Q10, Q20, Q30, and Q40. The H-bridge pcb is mounted right below the The H-bridge allows full functionality and can move perform both forward and backwards movement.

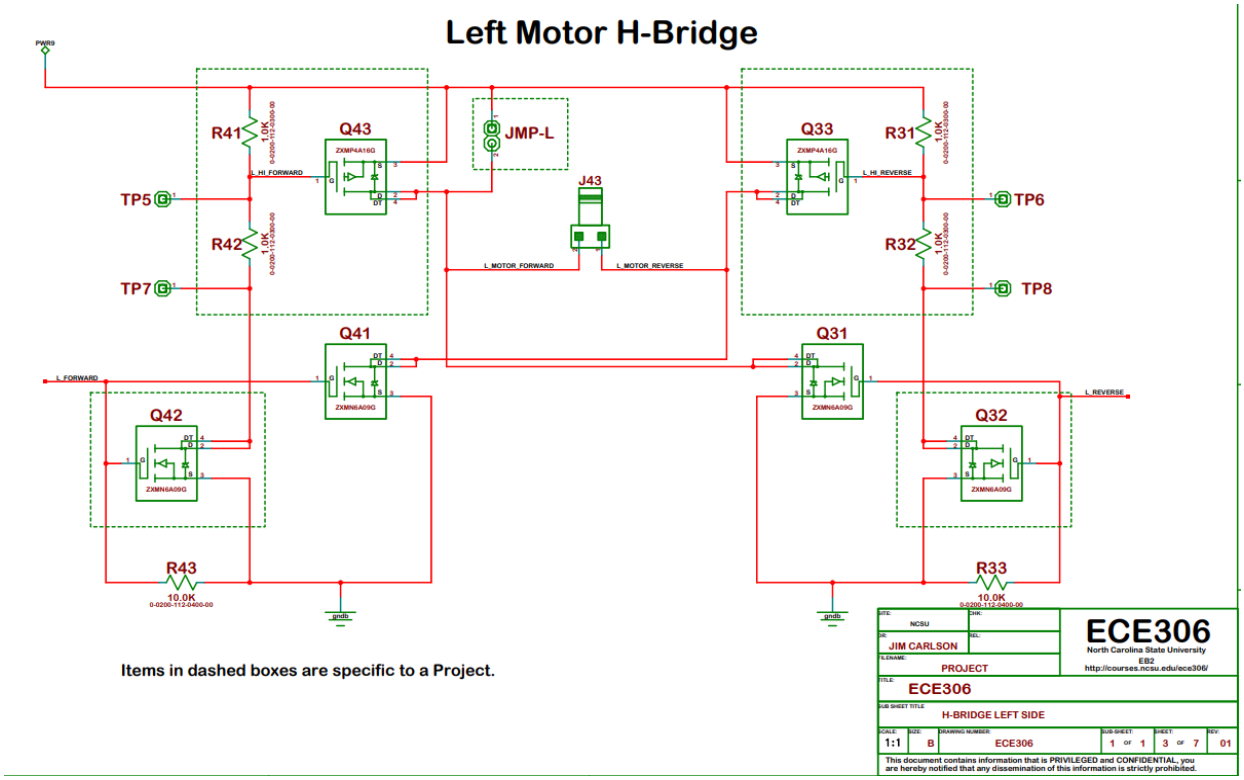

Figure 4.6: Left Motor H-Bridge Schematic

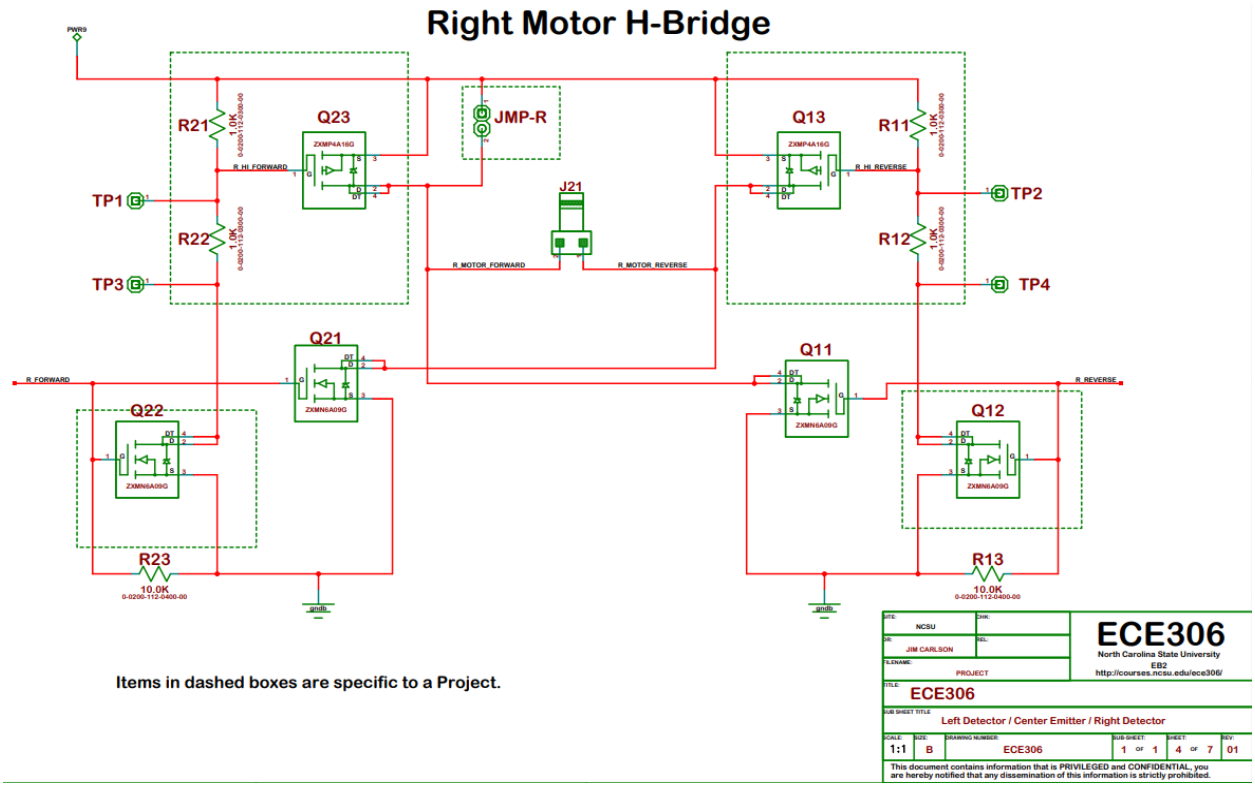

Figure 4.7: Right Motor H-Bridge Schematic

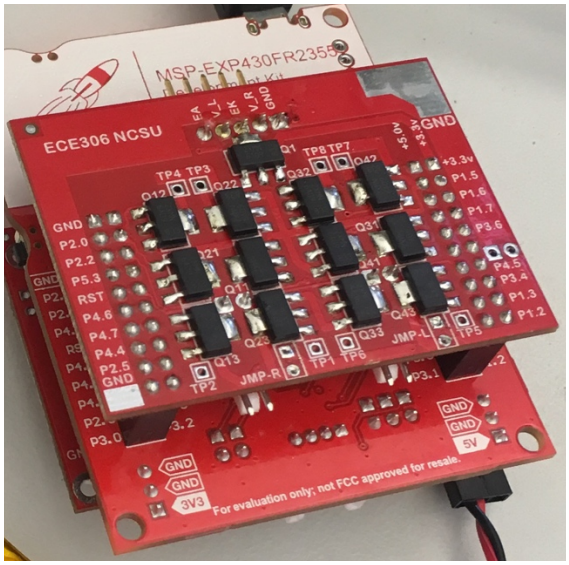

Figure 4.8: Full H-Bridge

4.4. Black Line Detector

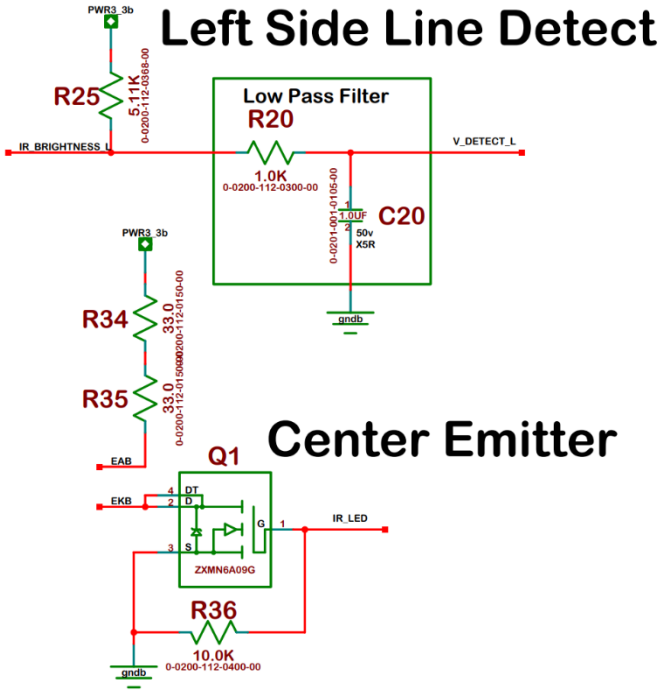

Figure 4.9: Left and Center Emitter Schematic

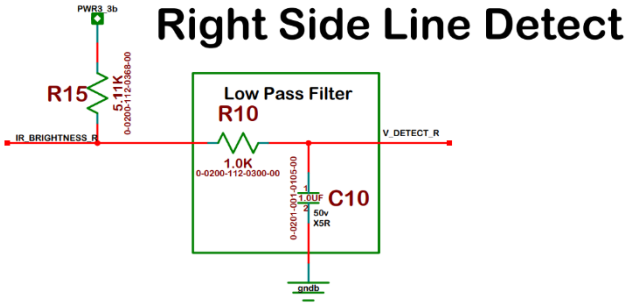

Figure 4.10 Right Side Emitter Schematic

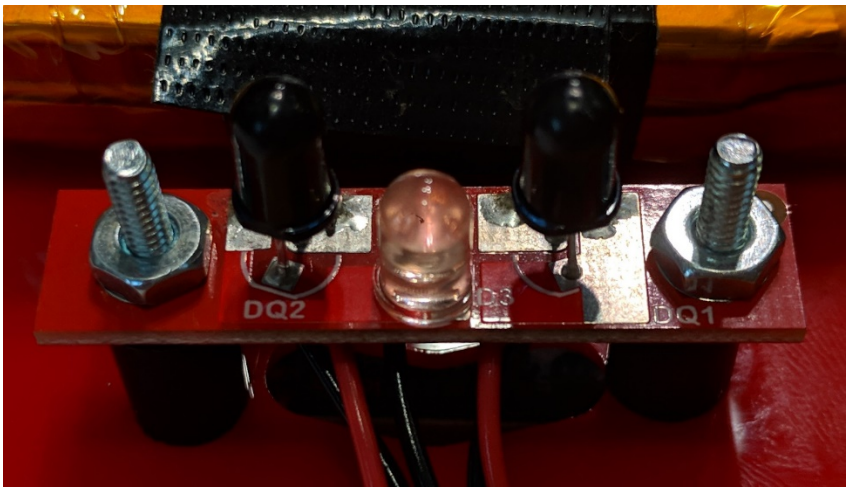

Figure 4.11: Black Line Detector Module

4.5. Serial Communications

Users communicate wirelessly with this robot using GainSpan's low power Wi-Fi connectivity module. With its own IP address, users can communicate with the robot from anywhere. This module is placed on its' own PCB which is then fixed to the top of the LCD board so that it is properly connected to the correct serial input/output ports. Being on the top of the robot also helps avoid antenna interference.

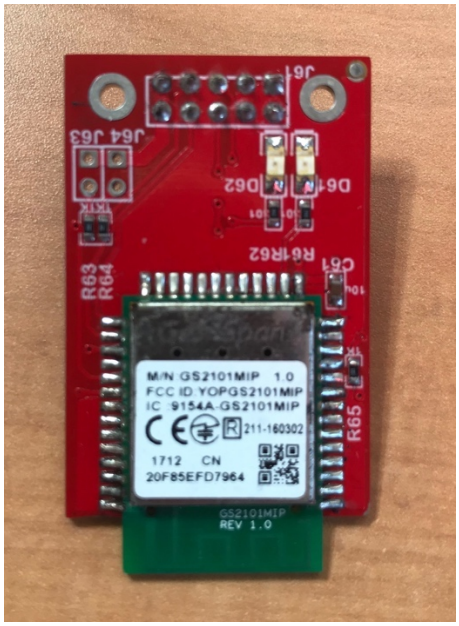

Figure 4.12: IOT Module

5. Power Analysis

The following table shows the power consumption of the major components in the car. Using a volt-amp meter the measured average current consumption is shown below.

Component	Current	Voltage	Power
FRAM	7.9 mA	3.3V	26.07 mW
LCD Backlight	68.1 mA	3.3V	224.73 mW
Infrared Emitter	29.1 mA	3.3V	96.03 mW
WiFi Module	145.1 mA	3.3V	478.83 mW
Motors	210 mA	6 V	1260 mW

The power consumption is calculated using the formula Power = Voltage * Current. The car is designed to operate at 3.3 V. The total power consumption of the car assuming that everything is running at once is 2085 mW. In order to determine the exact amount of power drawn from the batteries we also need to factor in the efficiency of conversion of the voltage regulator. We are using a voltage regulator to step up the voltage delivered to the car to be 6 volts.
Using the spec sheet provided by the manufacturer for the voltage regulator we see that it is 90% efficient. Applying the efficiency of conversion, we see that the batteries at full load see 2317 mW of power draw.

The battery pack that we are using puts four AA batteries in series. The exact cell used in the car is the Eneloop 1900mAh battery manufactured by Panasonic. Since there are a total of 4 batteries, each battery provides a quarter of the power necessary.

Having 4 batteries multiplies the voltage by 4 but keeps the current constant.

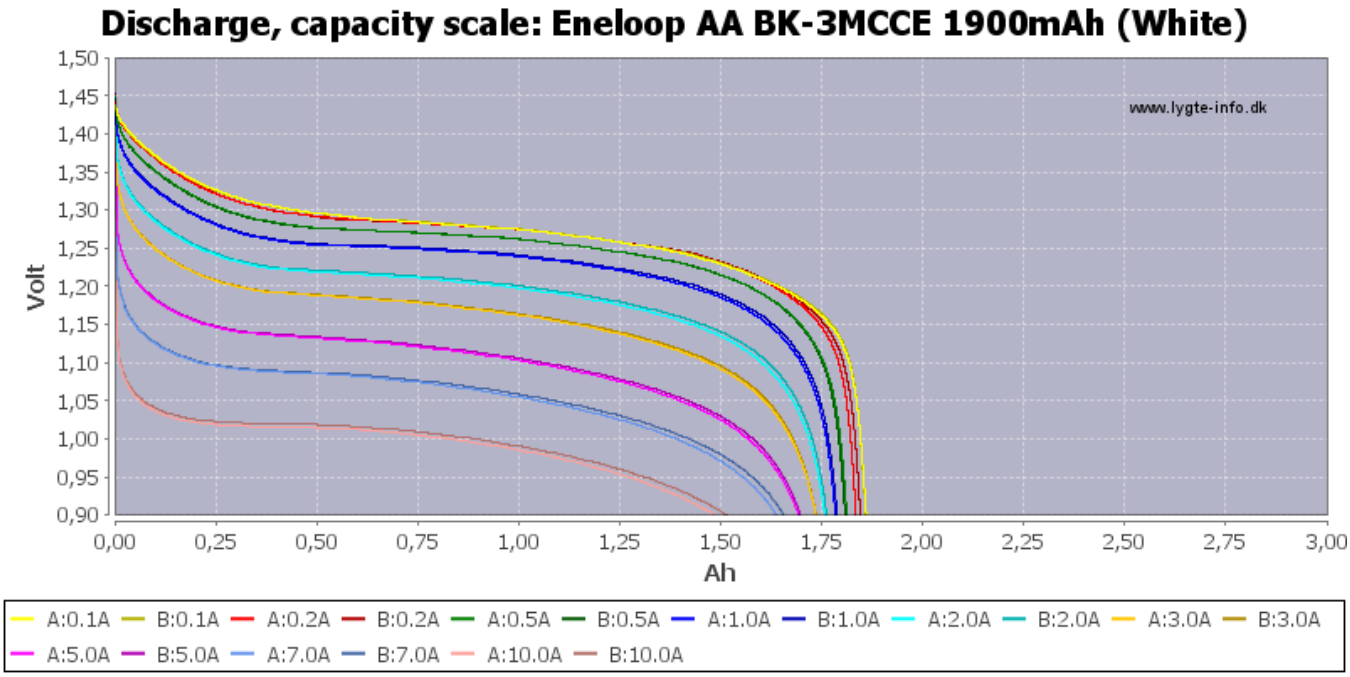

Figure 5.1: Eneloop AA Batteries Discharge and Capacity Graph

6. Test Process

The power board was the first piece of hardware to be tested after the construction of the control board. The power board is designed to be able to output a constant 3.3v with a varying DC input. A power supply and oscilloscope were set up in the lab to read the output. The power supply was set to 5 V and .1A output and the positive probe was connected to the J0 port and the negative probe was connected to ground. The Oscilloscope was connected to J12 and ground and the reading was ~3.3v as expected.

The power board and H-bridge were carefully inspected under a magnification lens to check for any faulty solder joints. Faulty solder joints are hard to find because appear to have been set properly but under magnification a hairline crack can be seen at the C7 capacitor, this can be seen in figure 6.1. For this example, the component would have to have the solder reflowed.

After the Battery pack was connected the voltage was measured at port J0 to ensure connectivity. A multimeter was used, and the voltage displayed was 6.3 V. This is displayed in figure 6.2

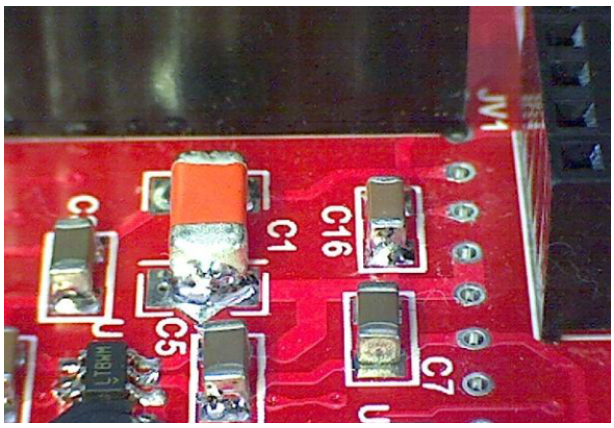

Figure 6.1: Faulty Solder Connection

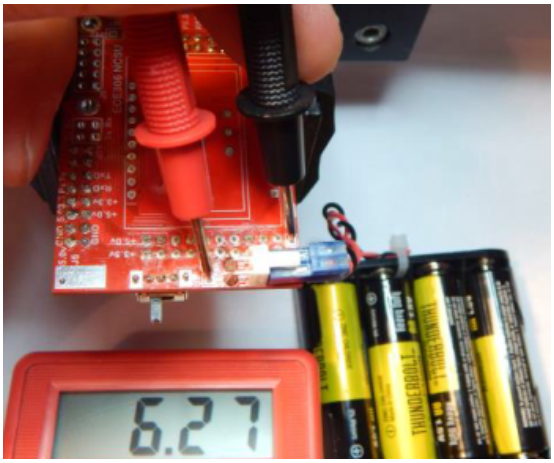

Figure 6.2: Test Voltage of Battery Pack

The LCD and Backlight were tested by connecting the battery pack and turning the switch on. The preloaded strings should be displayed on the LCD. Using the two interface buttons S1, and S2 the user is able to toggle between the two different messages displayed on the LCD. The battery pack connected to the display is seen in figure 6.3. The different displays the buttons can switch between are shown in figures 6.3 and 6.4

6.1. Full H-Bridge

The assembly of the car involved attaching the H-bridge to control movement of the wheels. The H-bridge was carefully inspected under the lab microscope to check for proper solder joints on all components on the reflowed side.

2X10 Connectors are mounted to the back of the MSP430 to allow for connection to the H-Bridge PCB. The H-bridge is soldered onto the connecters and then visually inspected to check for a solid connection on each pin. N-FETS are installed at device designation Q21 and Q41 while jumpers are installed at location JMP-R and JMP-L.

After the final assembly of the half H-bridge PCB the motors were connected and both forward motors were initialized to output high in the project code. Testing of the H-Bridge is done by observing the movement of the left and right motors.

6.2. Line Follow

The assembly of the car involved attaching the Line Detection Module. This consists of one Infrared Line Emitter and two Infrared Line Detectors. These components had their continuity checked with a multimeter to ensure strong and reliable connection. Furthermore, once this module was attached to the car, the Infrared Line Emitter's functionally was checked with a camera because infrared is undetectable by the human eye.

Once attached, the car is hovered above the line and slowly moved back and forth across the black line. This triggers the wheels to move. If the motors are not responding the way they should, the IR LED values and/or PWM.

6.3. Serial Communication

One of the major components to serial communication is making sure the receiver (Rx) and transmitter (Tx) ports share the same baud rate. On our car, this is performed with a push of a switch. Every time the "write" and "read" pointer does not match, the character that the "read" pointer is pointing at is saved into an array and the array index is incremented. Often times, messages start with the same ASCII character. This character is caught by our system, which resents the buffer array so that the string is read in the correct order.

To test this, the receiver and transmitter functionalities are repeated tested and checked for accuracy against multiple different valued signals.

7. Software

7.1. Main

As the name suggests, the "main" file utilizes all the code from the attached files to actually run the system. Main is responsible for all the motor movements, display features, and on-board LED changes.

7.2. Ports

Ports are initialized at the beginning of the code. Throughout the process of running your program, certain ports can be changed. These include the output of a port, changing the port from output to input or vice versa, and changing a port from a GPIO to a function pin. There are six ports on our processor and each port ranges from having five to eight pins. Port 1 contains the pins relating to using the ADC and the red LED. Port 2 contains pins relating to the SW2. Port 3 contains pins controlling the small SMCLK and the IOT Link. Port 4 contains pins used for SW1 and the LCD. Port 5 contains pins for the IR_LED used with the ADC. Finally, port 6 contains the pins that control all of the motor functions, as well as the LCD backlight and green LED. These are what the port pins are initialized to at the start of the program, but as certain processes happen throughout the program the values of the pins can change

7.3. Interrupts and ADC

An interrupt service routine is something that is executed in response to an interrupt signal that is thrown by hardware. The sequencing for the interrupt begins with the trigger which is received from hardware. It immediately stops running the main code and runs the interrupt service routine. Once the ISR finished the MCU continues right back at the line of code that the interrupt was triggered at.

The below code shows the Interrupt Service routine that runs when a switch is pressed. The line #pragma vector = PORT4_VECTOR tells the CPU to run the interrupt service routine that is stored at the address defined by that macro. The following lines of code are the additional code that needs to be run defined by us. For this specific example, we are telling it to check if the interrupt flag has been set to high. If so, then switch debouncing will begin. We will enable the B1 timer which is an interrupt that runs every 100 milliseconds. Within the B1 timer we will wait for 5 iterations which means that the switch will not be able to receive any additional input during this time period. After the B1 interrupt service routine is finished it will clear all the flags that were set to high in the switch ISR.

7.4. Timers

To create a timer, you must first initialize and setup how often you want the timer to happen. When initializing the timer, you must clear the control and index registers then select the clock you wish to source. You must then select how you wish to read the clock cycle which may be leading edge, falling edge, or continuous, and then you may divide the clock by setting the clock input divider and index divider expansion registers. The goal of the input dividers is to reduce the clock cycle number to be small enough so that we can set the interrupt interval as an integer value below 65535. After that, the timer clear register must be set to 1 then you must set the interrupt interval. This is gotten by calculating the clock over the input divider over the input divider expansion over 1 over the time wanted in seconds. An example of this would be if the clock is SMCLK with value (8,000,000), the index divider is 2, the index divider expansion is 8, and we want the timer to happen every 50 msec. This would get us 8,000,000 / 2 / 8 / (1 / .05)) = 25000. We would set the interrupt interval to 25000 to get a timer for 50 msec. Lastly, we must enable the interrupt to allow it to happen. Once it has been enabled, an interrupt associated with the vector for the timer can be created where actions which one wants to perform by the timer can occur.

7.5. Serial Communications

The first thing that must be done is to properly set up the pertinent ports. Once these are correctly configured, the EUSCI_A0_VECTOR and EUSCI_A1_VECTOR must be added to Interrupt_Ports.c. The important part of these interrupt is to store data coming in from UCA1RXBUF and UCA0RXBUF into a rotating buffer (resetting array). Once this is set up, the code must create a system that allows it to properly read commands. The way we do this is by creating a “command character” such as “$” that resets the array to index zero. Be resetting the array, we know the exact location of any commands that follows this command character and can then set up reactions in our main() based off of each specific following character.

8. Flow Chart

The following is the flow chart for our code.

8.1. Main Blocks (Nolan)

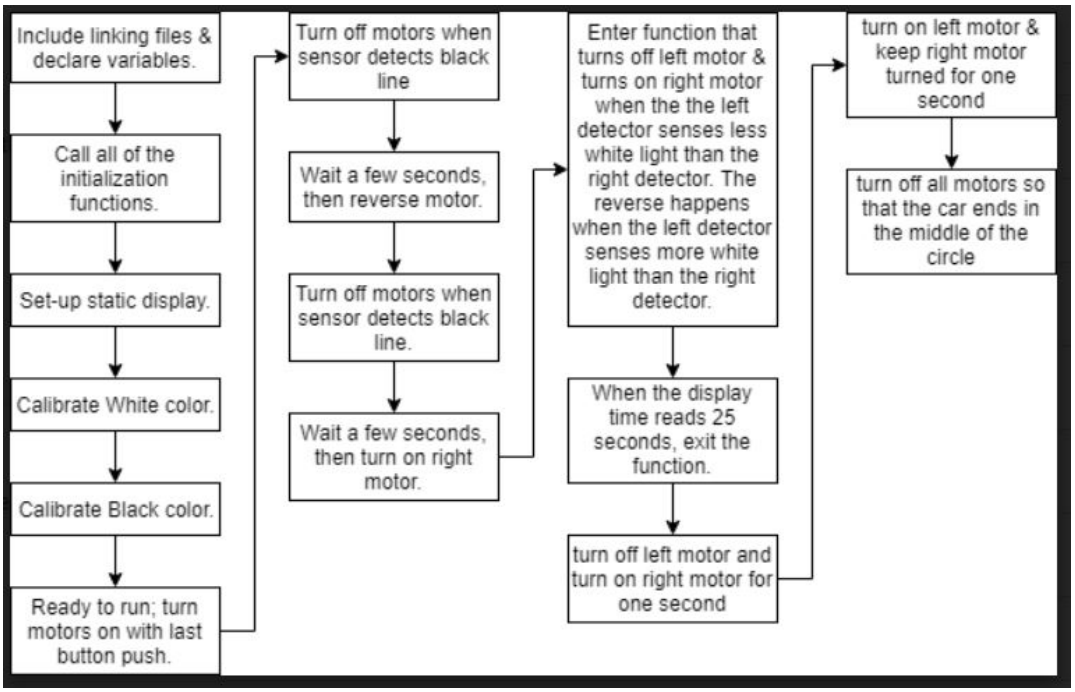

Figure 8.1: Main Flowchart (Nolan)

8.2. Main Blocks (Bennett)

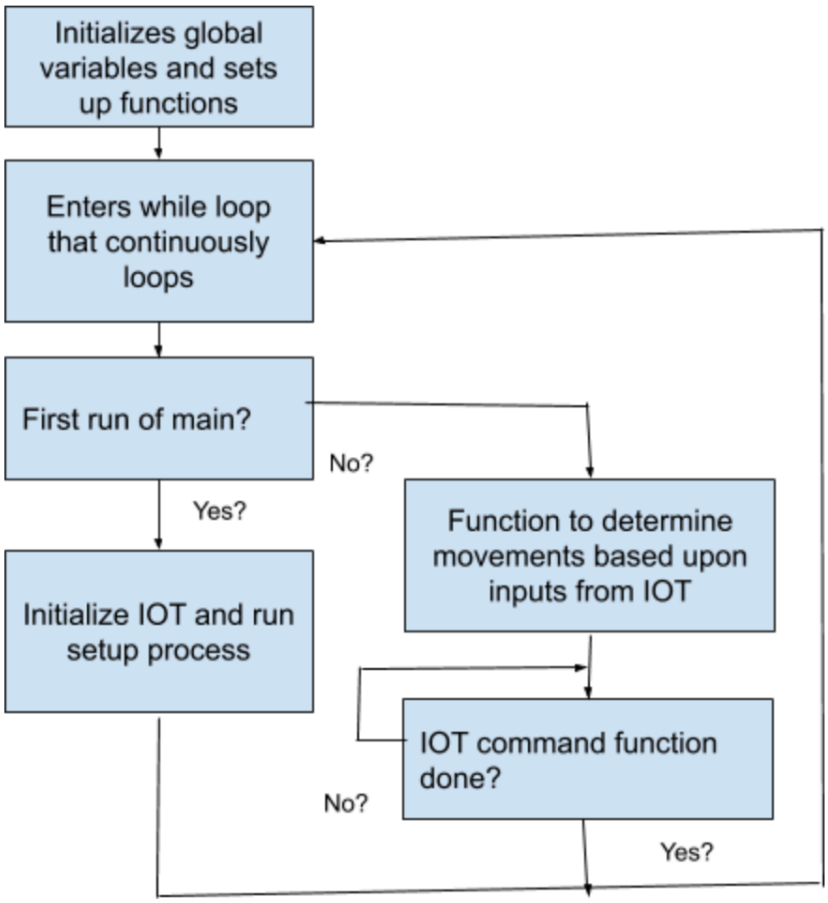

Figure 8.2: Main Flowchart (Bennett)

8.3. Main Block (Alec)

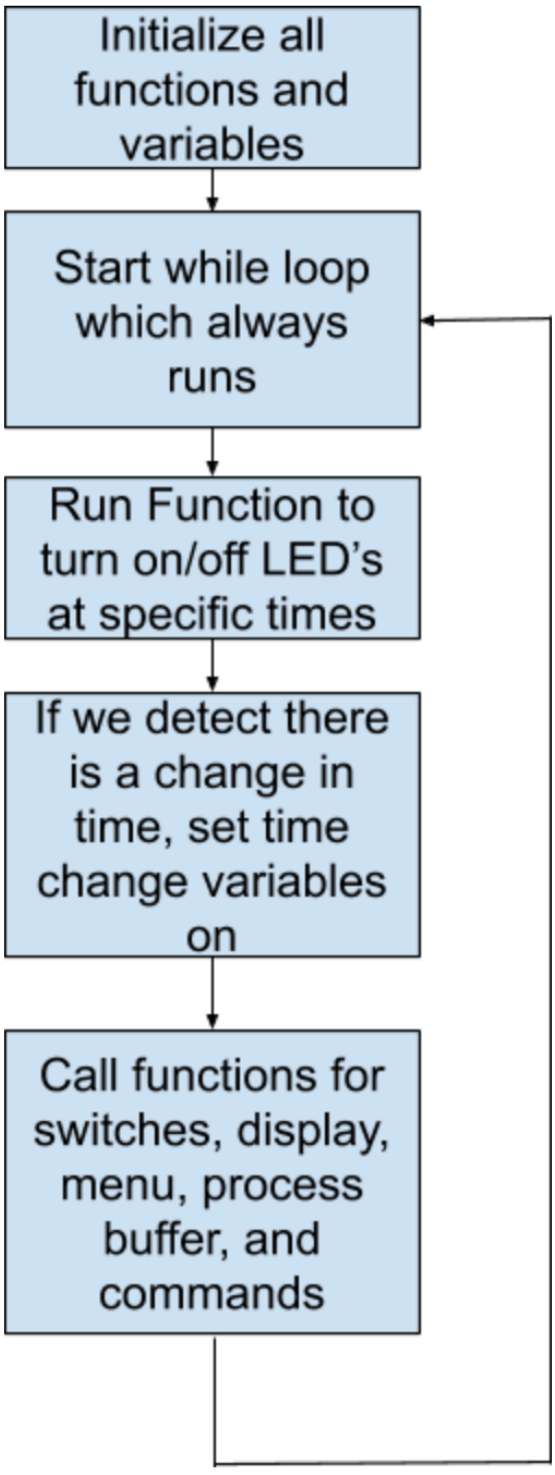

Figure 8.3 Main Flowchart (Alec)

8.4. Main Blocks (Steven)

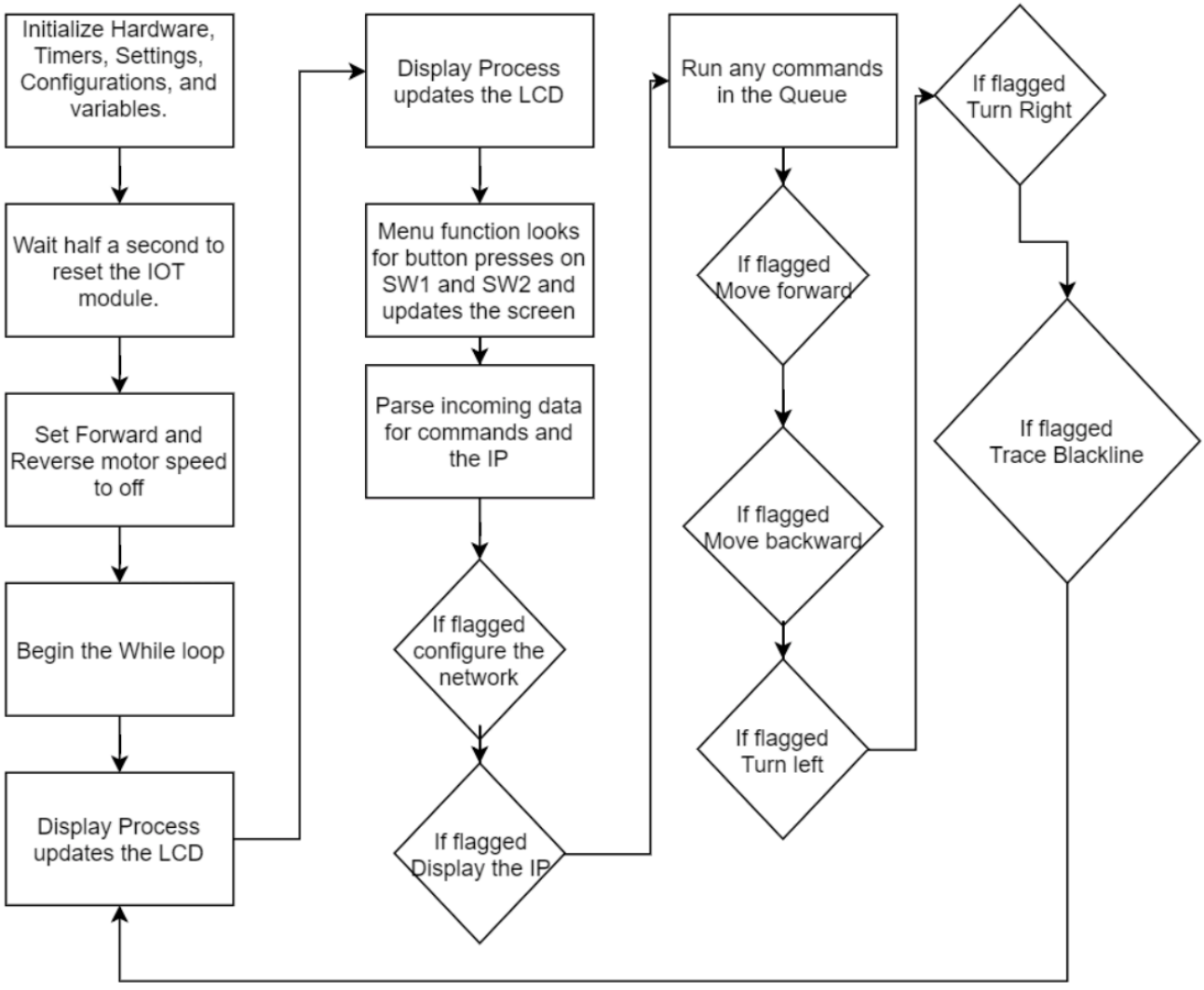

Figure 8.4 Main Flowchart (Steven)

8.5. Ports

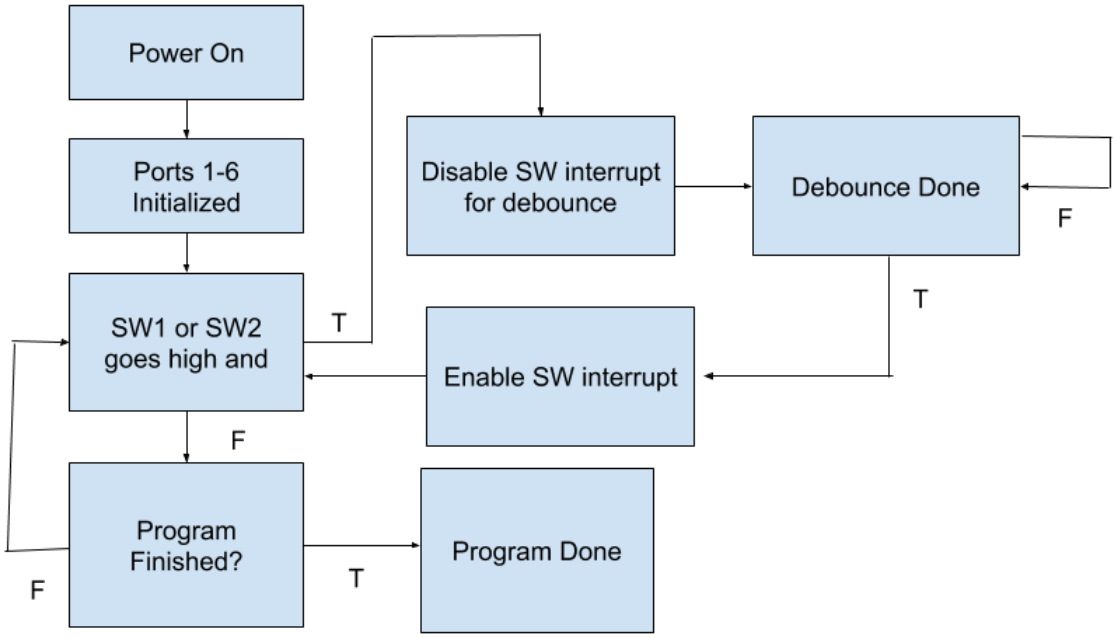

Figure 8.5: Ports Flowchart

8.6. Interrupts and ADC

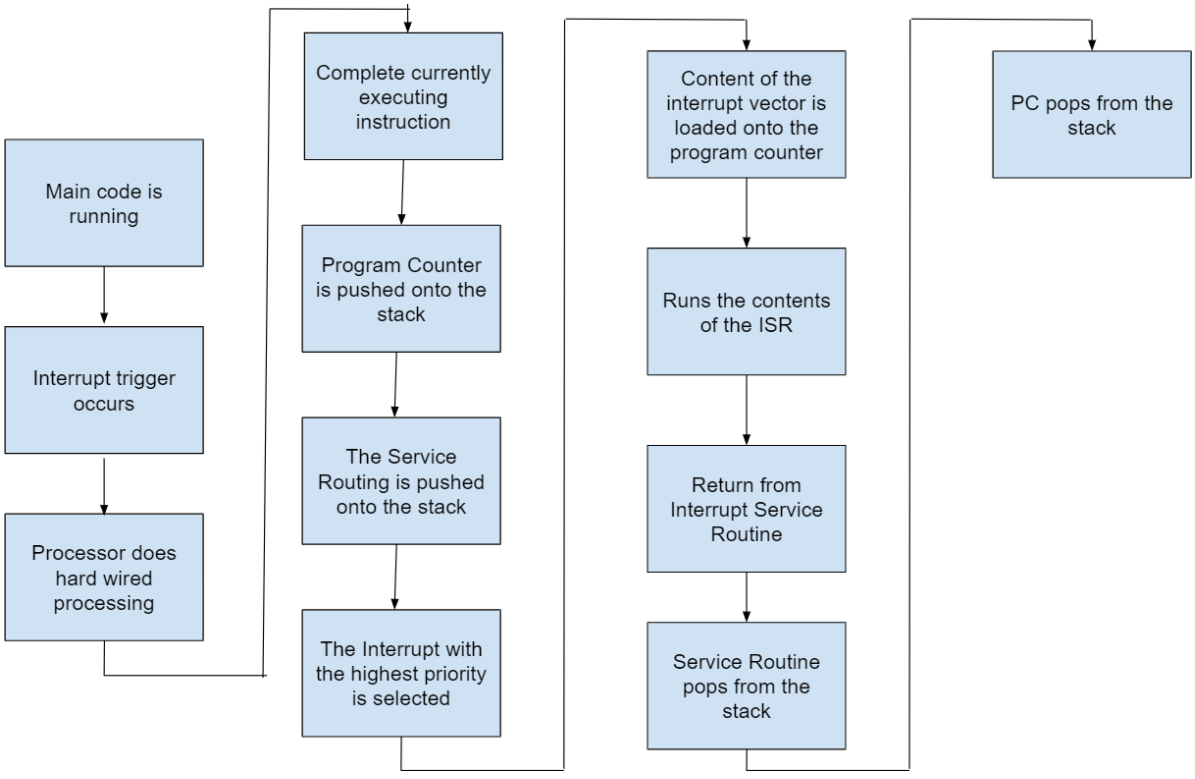

Figure 8.6: Interrupts Flowchart

8.7. Timers

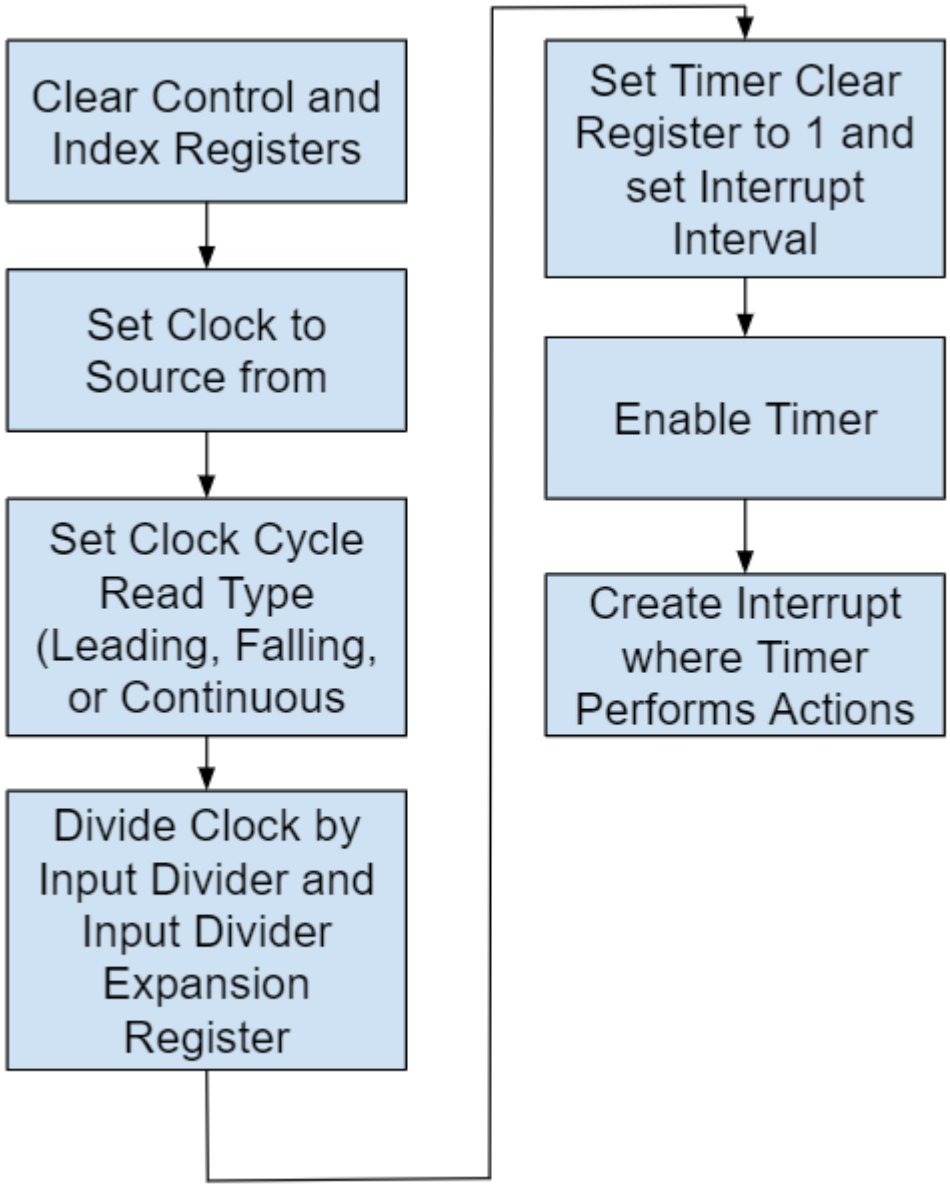

Figure 8.7: Timers Flowchart

8.8. Serial RX and TX Processing

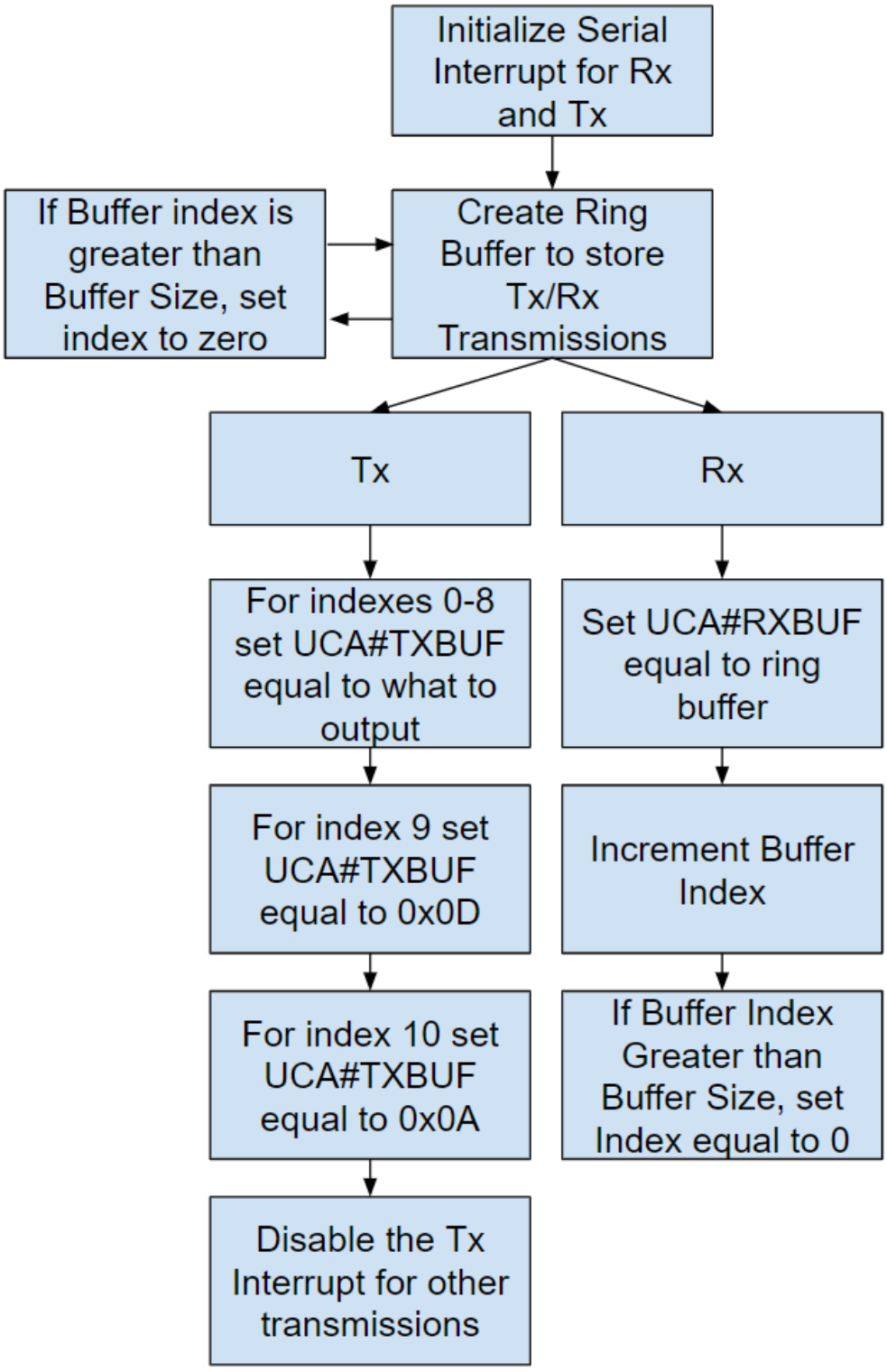

Figure 8.8: Serial RX and TX Flowchart

8.9. Display.c Flowchart

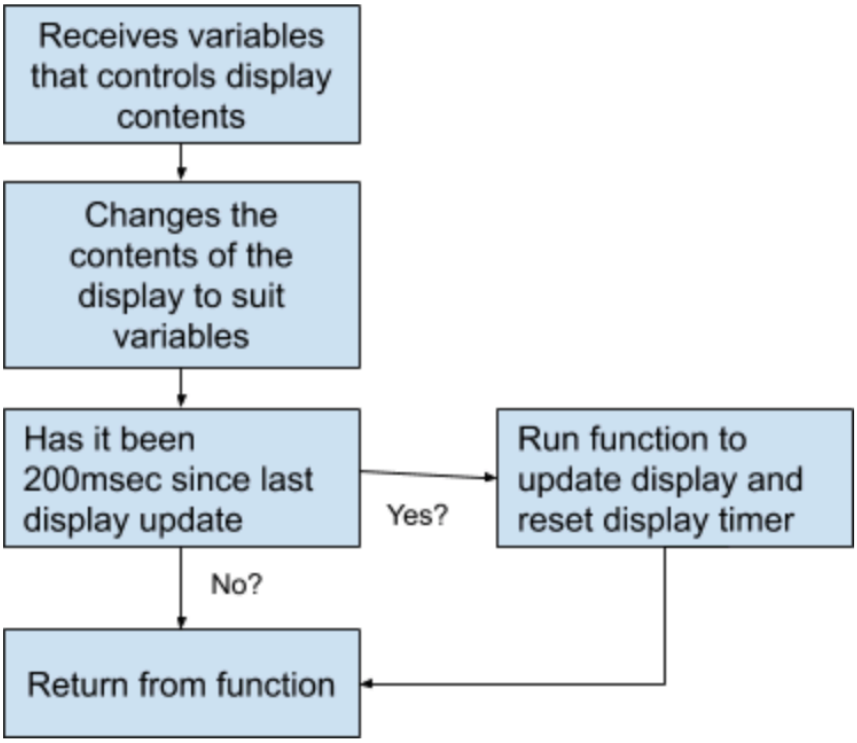

Figure 8.9 Display Flowchart

8.10. Movements.c Flowchart

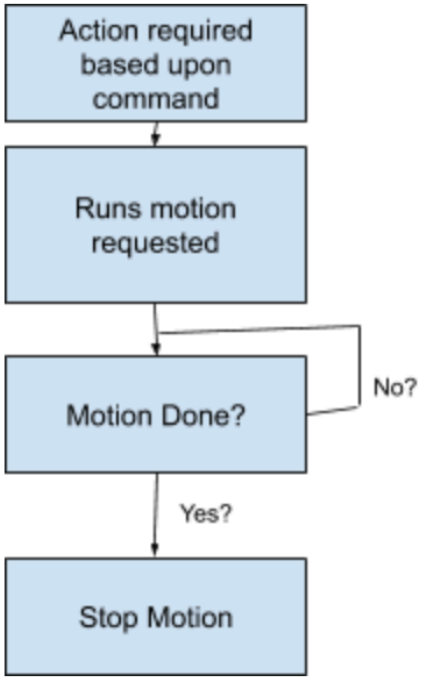

Figure 8.10 Movements Flowchart

8.11. A0_processes.c

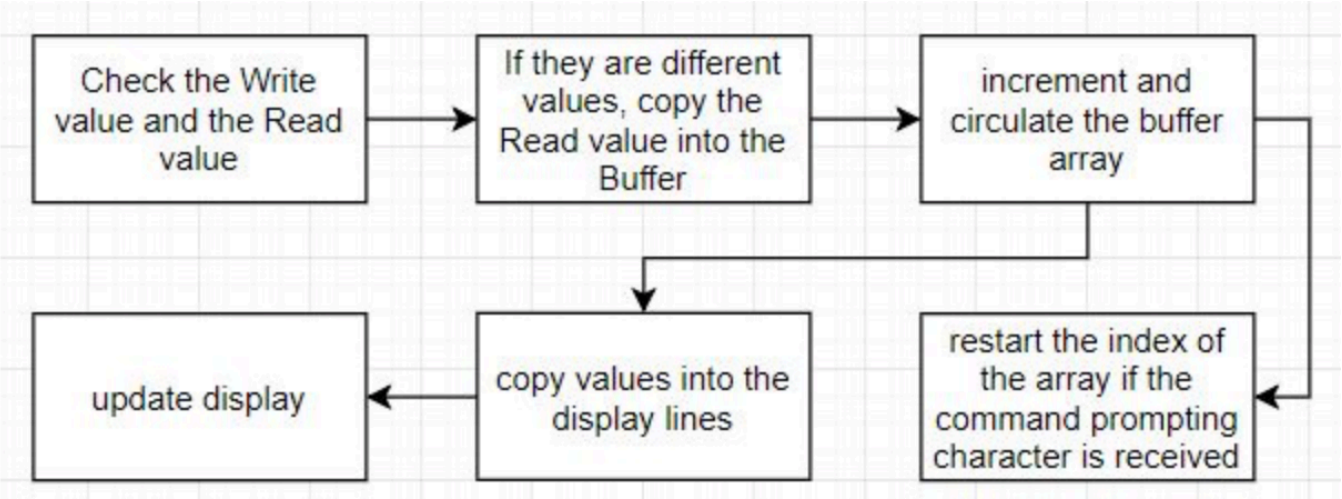

Figure 8.11 A0_processes Flowchart

8.12. Forward.c

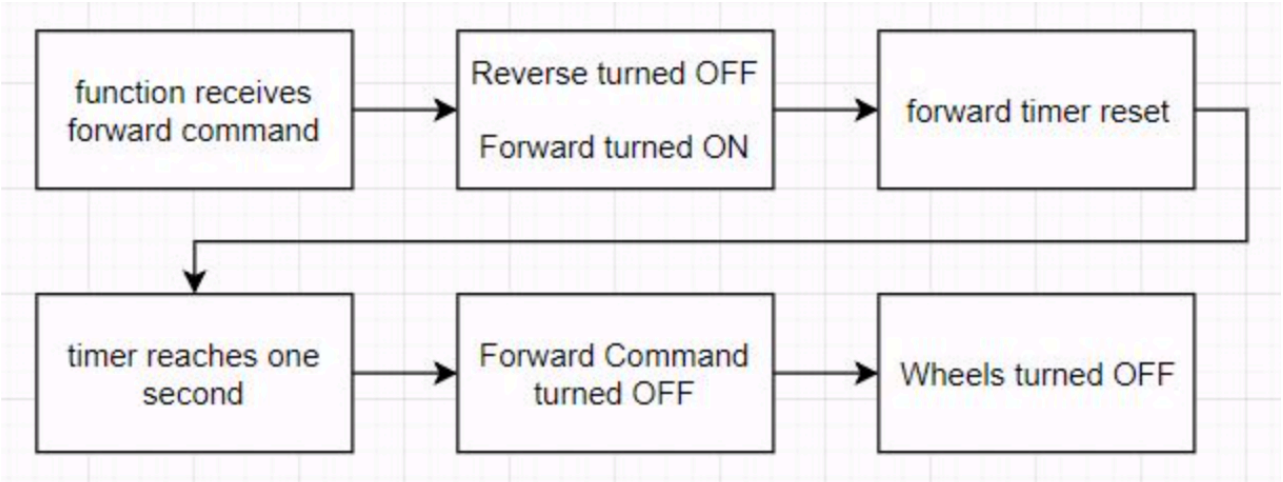

Figure 8.12 Forward Flowchart

8.13. menu.c

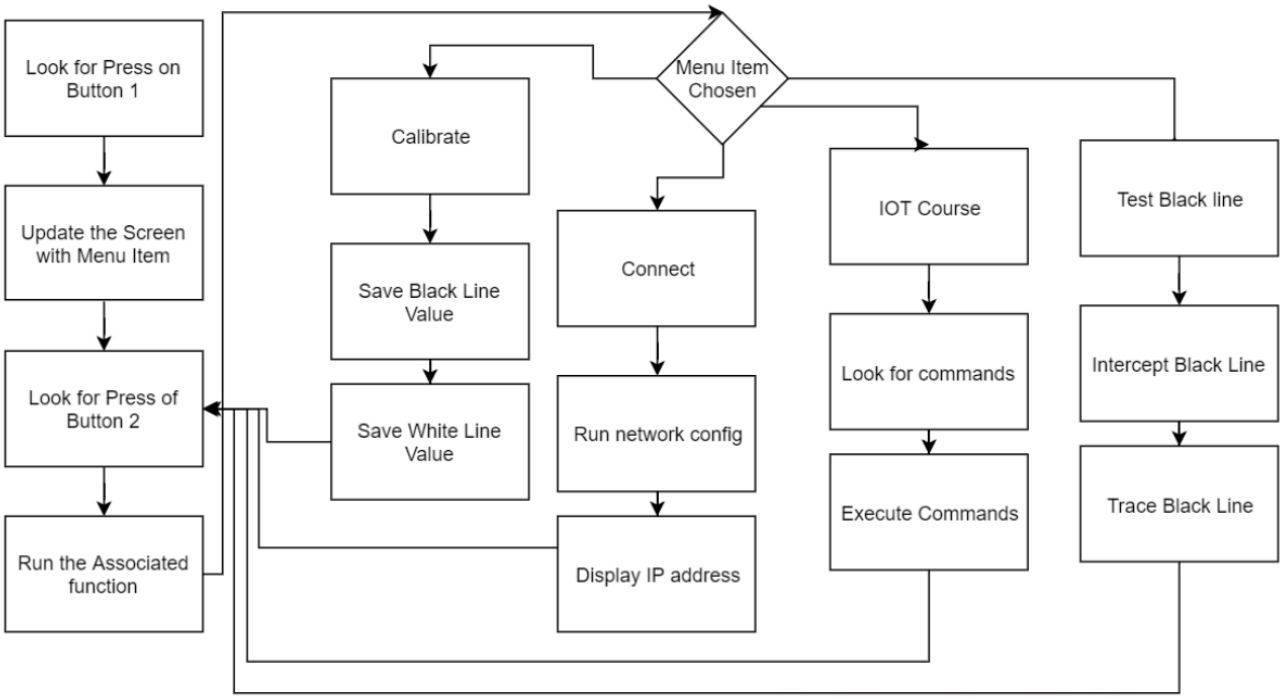

Figure 8.13 Menu Flowchart

8.14. networkconfig.c

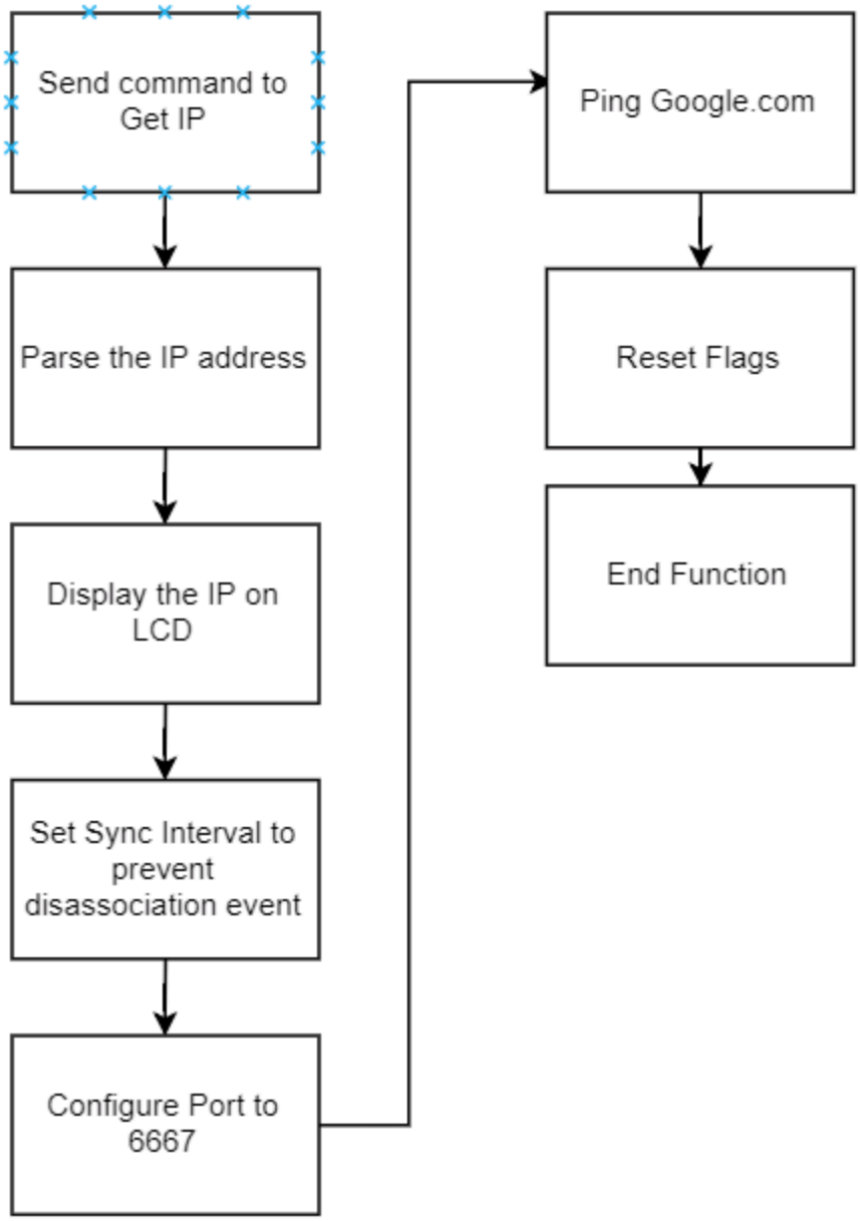

Figure 8.14 Network Configuration Flowchart

8.15. irconfig.c

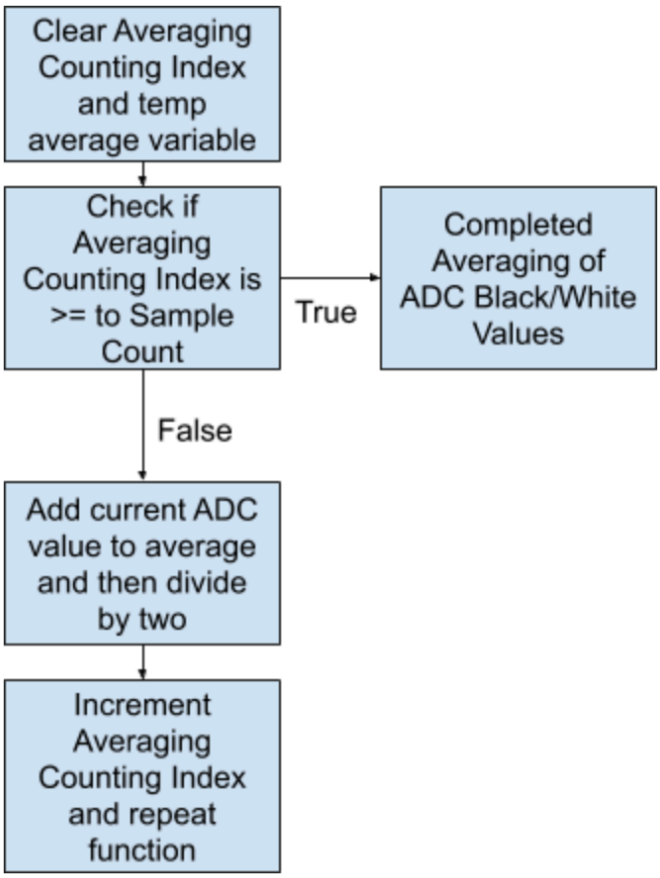

Figure 8.15 irconfig Flowchart

8.16. switches_proceses.c

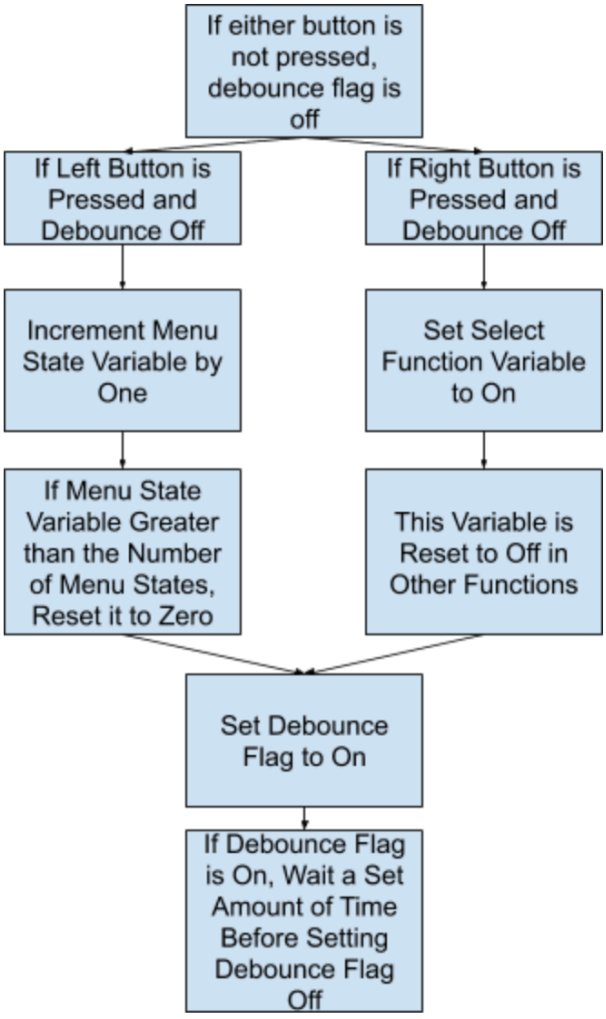

Figure 8.16 switch_processes Flowchart

9. Software Listing

9.1. Main.c (Nolan)

```
//-------------------------------------------------------------------------------
//  Description: This file contains the Main Routine - "While" Operating System
//  James Nolan Joyce
//  Feb 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[4][11];
extern char *display[4];
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;
unsigned int test_value;
char chosen_direction;
char change;

unsigned int Last_Time_Sequence;
unsigned int cycle_time;
unsigned int time_change;
unsigned int event;

unsigned int IR_led_Status;

extern volatile unsigned int SW1clicked;       // Set a variable to identify the switch has been pressed.
extern volatile unsigned int SW2clicked;       // Set a variable to identify the switch has been pressed.

extern unsigned int leftSW1_ctr;

extern volatile unsigned int DETECT_L;
extern volatile unsigned int DETECT_R;
unsigned int temp_L;
unsigned int temp_R;

unsigned int initial_pivot;
extern unsigned int Display_Time;

extern unsigned int Calibrate_Black;
extern unsigned int Calibrate_White;
unsigned int Switch2_Clicked;
extern unsigned char adc_char[FIVE];
extern unsigned int PlayDis;

extern volatile unsigned int SW1clicked;       // Set a variable to identify the switch has been pressed.
extern volatile unsigned int SW1debounce;      // Set a variable to identify the switch is being
debounced.
extern volatile unsigned int SW2clicked;       // Set a variable to identify the switch has been pressed.
extern volatile unsigned int SW2debounce;      // Set a variable to identify the switch is being
debounced.
```

```
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();
// Place the contents of what you want on the display, in between the quotes
// Limited to 10 characters per line
  display_line[ONE][FIVE] = 'T';
  update_string(display_line[ONE], ONE);
  display_changed = ONE;
  display_line[TWO][FIVE] = 'L';
  update_string(display_line[TWO], TWO);
  display_changed = ONE;
  display_line[THREE][FIVE] = 'R';
  update_string(display_line[THREE], THREE);
  display_changed = ONE;

  IR_led_Status = OFF;
  initial_pivot = 1;
  Switch2_Clicked = 0;
  PlayDis = 0;

//  P6OUT |= R_FORWARD;
//  P6OUT |= L_FORWARD;

// Begining of the "While" Operating System
//-------------------------------------------------------------------------
  while(ALWAYS) {                 // Can the Operating system run
    if(Last_Time_Sequence != Time_Sequence){
      Last_Time_Sequence = Time_Sequence;
      cycle_time++;
      time_change = ONE;
    }
//--//--//--//--//--//--//--//--//
  if(IR_led_Status){
    display_line[ZERO][SEVEN] = ' ';
    display_line[ZERO][EIGHT] = 'O';
```

```
    display_line[ZERO][NINE] = 'N';
    update_string(display_line[ZERO], ZERO);
    display_changed = ONE;
   }
   else{
    display_line[ZERO][SEVEN] = 'O';
    display_line[ZERO][EIGHT] = 'F';
    display_line[ZERO][NINE] = 'F';
    update_string(display_line[ZERO], ZERO);
    display_changed = ONE;
   }
//--//--//--//--//--//--//--//--//
   if(PlayDis == 0){
    display_line[THREE][ZERO] = 'C';
    display_line[THREE][ONE] = 'a';
    display_line[THREE][TWO] = 'l';
    display_line[THREE][THREE] = 'W';
    display_line[THREE][FOUR] = ' ';
    update_string(display_line[THREE], THREE);
    display_changed = ONE;
   }
   if(PlayDis == 1){
    display_line[THREE][ZERO] = 'C';
    display_line[THREE][ONE] = 'a';
    display_line[THREE][TWO] = 'l';
    display_line[THREE][THREE] = 'B';
    display_line[THREE][FOUR] = ' ';
    update_string(display_line[THREE], THREE);
    display_changed = ONE;
   }
   if(PlayDis == 2){
    display_line[THREE][ZERO] = 'R';
    display_line[THREE][ONE] = 'e';
    display_line[THREE][TWO] = 'a';
    display_line[THREE][THREE] = 'd';
    display_line[THREE][FOUR] = 'y';
    update_string(display_line[THREE], THREE);
    display_changed = ONE;
   }
   if(PlayDis == 3){
    display_line[THREE][ZERO] = 'R';
    display_line[THREE][ONE] = 'u';
    display_line[THREE][TWO] = 'n';
    display_line[THREE][THREE] = ':';
    display_line[THREE][FOUR] = ')';
    update_string(display_line[THREE], THREE);
    display_changed = ONE;
```

```
  }
//--//--//--//--//--//--//--//--//
  temp_R = DETECT_R;
  temp_L = DETECT_L;

  if(PlayDis == 3){
    P6OUT |= R_FORWARD;
    P6OUT |= L_FORWARD;
    if (temp_R > 100){
      PlayDis++;
      initial_pivot++;
    }
  }
if (Display_Time >= 2){
  switch(initial_pivot){
  case 1:
    break;
  case 2:
      P6OUT &= ~R_FORWARD;
      P6OUT &= ~L_FORWARD;
      if (Display_Time >= 6){ initial_pivot++; }
    break;
  case 3:
      P6OUT |= R_REVERSE;
      P6OUT |= L_REVERSE;
      if (temp_R > 100){ initial_pivot++; }
    break;
  case 4:
      P6OUT &= ~R_REVERSE;
      P6OUT &= ~L_REVERSE;
      if (Display_Time >= 10){ initial_pivot++; }
      break;
  case 5:
      P6OUT |= R_FORWARD;
      if (Display_Time >= 11){ initial_pivot++; }
      break;
  case 6:
    if (temp_L > temp_R){
      P6OUT &= ~R_FORWARD;
      P6OUT |= L_FORWARD;
    }
    else if((temp_L < temp_R)){
      P6OUT |= R_FORWARD;
      P6OUT &= ~L_FORWARD;
    }
    else{
      P6OUT |= R_FORWARD;
```

```
    P6OUT |= L_FORWARD;
   }
   if(Display_Time >= 22){ initial_pivot++; }
   break;
  case 7:
   P6OUT |= R_FORWARD;
   P6OUT &= ~L_FORWARD;
   if(Display_Time >= 23){ initial_pivot++; }
   break;
  case 8:
   P6OUT |= R_FORWARD;
   P6OUT |= L_FORWARD;
   if(Display_Time >= 24){ initial_pivot++; }
  break;
  case 9:
   P6OUT &= ~R_FORWARD;
   P6OUT &= ~L_FORWARD;
   P6OUT &= ~R_REVERSE;
   P6OUT &= ~L_REVERSE;
  default:
   break;
  }
}
Display_Process();
  }
}
```

9.2. Main.c (Bennett)

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

// Global Variables
volatile char slow_input_down;
extern char display_line[4][11];
extern char *display[4];
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;

unsigned int once = HIGH;
extern unsigned int read_index;

extern unsigned int display_IP;
```

```
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();                     // Initializes ADC
 Init_Serial_UCA0();
 Init_Serial_UCA1();

 strcpy(display_line[0], "          ");
 update_string(display_line[0], 0);
 strcpy(display_line[1], "          ");
 update_string(display_line[1], 1);
 strcpy(display_line[2], "          ");
 update_string(display_line[2], 2);
 strcpy(display_line[3], "          ");
 update_string(display_line[3], 3);
 display_changed = 1;
 lcd_4line();
//-------------------------------------------------------------------------
// Beginning of the "While" Operating System
//-------------------------------------------------------------------------
 while(ALWAYS) {                // Can the Operating system run

  if(once == HIGH){
   initial_setup();}
  Display_Process();          // Display update timer
  Update_Display();           // Updates the display on timer (200msec)

  if(once == LOW){
  clears();
  commands();
  transmit_receive();
  }
}
```

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

9.3. Main.c (Alec)

Description: The Main function initializes all functions and their values then starts a while loop that always runs. Inside of that while loop a function runs which turns the red and green LED lights on and off, first turning on the green and off the red, then turning on the red and off the green, then turning both LED's on before turning both LED's off. Also in the while loop, a function runs which detects if there has been a change in time. If there has been, time change variables are turned on. Finally, the functions for switches, display, menu, process buffer, and commands are called for.

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

// Global Variables
volatile char slow_input_down;
extern char display_line[DISPCOLUMN][DISPCHAR];
extern char *display[DISPCOLUMN];
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;
extern unsigned int pindef;
volatile char one_time;
unsigned int test_value;
extern unsigned int cycle_time = RESET_STATE;
extern unsigned int time_change = RESET_STATE;
unsigned int time_motor = RESET_STATE;
unsigned int time_travel = RESET_STATE;
extern char event = NONE;
unsigned int Old_Time_Sequence;
extern char state = END;
extern unsigned int ADC_Thumb;
extern unsigned int ADC_Det_L;
extern unsigned int ADC_Det_R;

extern volatile char IOT_Char_Rx[];
extern volatile unsigned int iot_rx_ring_wr;
extern unsigned int iot_rx_ring_rd;
extern volatile char USB_Char_Rx[];
extern volatile unsigned int usb_rx_ring_wr;
extern unsigned int usb_rx_ring_rd;
```

```
extern unsigned int RF_PERCENT;
extern unsigned int LF_PERCENT;

//Global HEX/BCD Values
unsigned int run_time = RESET_STATE;

void main(void){
 // 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_Timer_B0();               // Initialize Timers
 Init_Timer_B3();
 Init_LCD();                    // Initialize LCD
 Init_ADC();                    //Initialize ADC
 Init_Serial_UCA0();            //Initialize Serial Port (115200 Baud default)
 Init_Serial_UCA1();
 // Place the contents of what you want on the display, in between the quotes
 // Limited to 10 characters per line
 //

 //-------------------------------------------------------------------------
 // Begining of the "While" Operating System
 //-------------------------------------------------------------------------
 while(ALWAYS) {                // Can the Operating system run
  switch(Time_Sequence){
  case TWOFIFTY_MS:             //
   if(one_time){
    Init_LEDs();
    display_changed = TURNON;
    one_time = RESET_STATE;
   }
   Time_Sequence = RESET_STATE;          //
   break;
  case TWOHUNDRED_MS:           //
   if(one_time){
    GREEN_LED_ON;          // Change State of LED 5
    one_time = RESET_STATE;
   }
   break;
  case ONEFIFTY_MS:             //
   if(one_time){
    RED_LED_ON;            // Change State of LED 4
    GREEN_LED_OFF;         // Change State of LED 5
```

```
      one_time = RESET_STATE;
    }
    break;
  case ONEHUNDRED_MS:                    //
    if(one_time){
      lcd_4line();
      GREEN_LED_ON;          // Change State of LED 5
      display_changed = TURNON;
      one_time = RESET_STATE;
    }
    break;
  case  FIFTY_MS:                    //
    if(one_time){
      RED_LED_OFF;          // Change State of LED 4
      GREEN_LED_OFF;          // Change State of LED 5
      one_time = RESET_STATE;
    }
    break;                    //
  default: break;
  }


  if(Time_Sequence != Old_Time_Sequence)          //Time Sequence Changes every 50msec
  {
    cycle_time++;
    run_time++;
    Old_Time_Sequence = Time_Sequence;
    time_change = TURNON;
    time_motor = TURNON;
    time_travel = TURNON;
  }

  Switches_Process();        // Check for switch state change
  Display_Process();
  Menu_Process();
  Process_Buffer_Dump();
  Command_Proccessing();
  }
 }
 //---------------------------------------------------------------------------
}
```

9.4. Main.c (Steven)

Description: The main function in my code is responsible for initializing all the input/output pins, timers, settings configurations and variables that are going to be using while my car is running. After it is finished with these initializations it enters a while loop. My car runs off a foreground/background operating system. This means that is continually runs through all the functions in main and services any devices when necessary. There is a limitation to running functions like this Because the system might get slow and then overall run times will take longer. This problem is solved with interrupt routines, the interrupt routines handle the most urgent work by setting flags to request processing by the main loop.

```
//-------------------------------------------------------------------------
//
//  Description: This file contains the Main Routine - "While" Operating System
//
//
//  Steven Yan
//  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"



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

 // Global Variables
volatile char slow_input_down;
extern char display_line[DISPLAYFOUR][DISPLAYCOLUMNS];
extern char *display[DISPLAYFOUR];
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;

char event;
```

```
volatile unsigned int cycle_time;
volatile unsigned int ADC_Thumb;
volatile unsigned int V_detect_r;
volatile unsigned int V_detect_l;

char buttonOne;
char buttonTwo;

//variables to handle switch debouncing
unsigned int debouncer = RESET_STATE;
char debounce_TF = RESET_STATE;
unsigned int menu_item;

char adc_char[5];

//project 7 variables

volatile unsigned int ones;
volatile unsigned int tens;
volatile unsigned int hundreds;
volatile unsigned int milliseconds;
unsigned int first_time;
unsigned int timer_enable = RESET_STATE;

char lineEvent;
char currentState;
unsigned int blackLine;
unsigned int whiteSpace;
unsigned int displayValues;
unsigned int displayToggle;
unsigned int batteryFull;

//homework 8 variables
unsigned int UCA0_index;
unsigned int baud1 = BR115;
unsigned int mctlw1 = UCFX115;
unsigned int baud2;
char * outputstring = "NCSU  #1";
char * getIP = "AT+NSTAT=?\n";
extern unsigned int ipsearchstate;
unsigned int forwardCommand;
unsigned int reverseCommand;
unsigned int leftCommand;
unsigned int rightCommand;
unsigned char IOTcourse = '0';
unsigned int nwflag;
```

```
unsigned int broadcastcommand;
char startIOTflag;
char startblackline;
//-------------------------------------------------------------------------
// 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


void main(void){

 PM5CTL0 &= ~LOCKLPM5;
 Init_Ports(USE_GPIO);              // 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(BR115, UCFX115);
 Init_Serial_UCA1(BR115, UCFX115);
 cycle_time = RESET_STATE;
 while(cycle_time < 10);
 P5OUT |= IOT_RESET;                //set the IOT module

 //turn off all motors
 RIGHT_FORWARD_SPEED = WHEEL_OFF;
 LEFT_FORWARD_SPEED = WHEEL_OFF;
 RIGHT_REVERSE_SPEED = WHEEL_OFF;
 LEFT_REVERSE_SPEED = WHEEL_OFF;
//    P5OUT |= IR_LED;                //Start the IR LED On
// P6OUT |= LCD_BACKLITE;             //Start the LCD backlight on

 //ipsearchstate = 1;


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

   Display_Process();               //initialize display update stuff
 //  if(!batteryFull) { splashScreen(); }   //load the splash screen first
 //  if(batteryFull) { menu(); }            //load the menu
   menu();                          //load the menu
```

```
  if(broadcastcommand) { networkconfig(); }       //parse for the IP and set port
   blacklinedetection();                    //State machine to follow black line
  if(broadcastcommand) { showIP(); }            //display the IP on the LCD
  processInput();                           //parse the incoming data for commands
  executeCommand();                            //run commands received from PC
  if (startIOTflag) {  IOTscreen(); }      //IOT course screen
  if (forwardCommand) {  forward_2secs(); }  //drive forward
  else if(reverseCommand) {  backwards_1sec(); } //move backwards
  else if(leftCommand) { turn_left45(); }        //turn the car left
  else if(rightCommand) { turn_right90(); }       //turn the car right
}
}
```

9.5. Ports.c

```
//-------------------------------------------------------------------------------
// File Name : ports.c
//
// Description: This file contains functions for initializing and defining each
//                  individual port pin of the MSP430 and a function to call the port
//                  initializations individually.
//
// Bennett James
// 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>

void Init_Ports(void){
//-------------------------------------------------------------------------------
//Call functions for initializing port pins

 Init_Port1();
 Init_Port2();
 Init_Port3();
 Init_Port4();
 Init_Port5();
 Init_Port6();
  //-------------------------------------------------------------------------------
}
void Init_Port1(void){
//-------------------------------------------------------------------------------
//Configure PORT 1, PINS 0-7
```

```
 P1DIR = 0xFF;
 P1OUT = 0x00;

 P1SEL0 &= ~RED_LED;
 P1SEL1 &= ~RED_LED;
 P1DIR |= RED_LED;
 P1OUT |= RED_LED;

 P1SEL0 |= A1_SEEED;
 P1SEL1 |= A1_SEEED;

 P1SEL0 |= V_DETECT_L;
 P1SEL1 |= V_DETECT_L;

 P1SEL0 |= V_DETECT_R;
 P1SEL1 |= V_DETECT_R;

 P1SEL0 |= A4_SEEED;
 P1SEL1 |= A4_SEEED;

 P1SEL0 |= V_THUMB;
 P1SEL1 |= V_THUMB;

 P1SEL0 |= UCA0TXD;
 P1SEL1 &= ~UCA0TXD;

 P1SEL0 |= UCA0RXD;
 P1SEL1 &= ~UCA0RXD;

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

void Init_Port2(void){
//-------------------------------------------------------------------------
//Configure PORT 2. PINS 0-7

 P2DIR = 0xFF; // Set P2 direction to output
 P2OUT = 0x00; // P2 set Low

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

 P2SEL0 &= ~P2_1; // P2_1 GPIO operation
 P2SEL1 &= ~P2_1; // P2_1 GPIO operation
 P2DIR &= ~P2_1; // Direction = input
```

```
 P2SEL0 &= ~P2_2; // P2_2 GPIO operation
 P2SEL1 &= ~P2_2; // P2_2 GPIO operation
 P2DIR &= ~P2_2; // Direction = input

 P2SEL0 &= ~SW2; // SW2 Operation
 P2SEL1 &= ~SW2; // SW2 Operation
 P2DIR &= ~SW2; // Direction = input

 P2PUD |= SW2; // Configure pullup resistor
 P2REN |= SW2; // Enable pullup resistor

 P2IES |= SW2; // P2.0 Hi/Lo edge interrupt
 P2IFG &= ~SW2; // Clear all P2.6 interrupt flags
 P2IE |= SW2; // P2.6 interrupt enabled

 P2SEL0 &= ~P2_4; // P2_4 GPIO operation
 P2SEL1 &= ~P2_4; // P2_4 GPIO operation
 P2DIR &= ~P2_4; // Direction = input

 P2SEL0 &= ~P2_5; // P2_5 GPIO operation
 P2SEL1 &= ~P2_5; // P2_5 GPIO operation
 P2DIR &= ~P2_5; // Direction = input

 P2SEL0 &= ~LFXOUT; // LFXOUT Clock operation
 P2SEL1 |= LFXOUT; // LFXOUT Clock operation

 P2SEL0 &= ~LFXIN; // LFXIN Clock operation
 P2SEL1 |= LFXIN; // LFXIN Clock operation
//------------------------------------------------------------------------------
}

void Init_Port3(void){
//------------------------------------------------------------------------------
//Configure PORT 3, PINS 0-7
 P3SEL0 = 0x00;
 P3SEL1 = 0x00;
 P3DIR = 0x00; // Set P1 direction to output
 P3OUT = 0x00; // P1 set Low

 P3SEL0 &= ~TEST_PROBE;
 P3SEL1 &= ~TEST_PROBE;
 P3DIR |= TEST_PROBE;
 P3OUT &= ~TEST_PROBE;

 P3SEL0 |= OA20;
 P3SEL1 |= OA20;
```

```
 P3SEL0 |= OA2N;
 P3SEL1 |= OA2N;

 P3SEL0 |= OA2P;
 P3SEL1 |= OA2P;

                        // Makes SMCLK_OUT a functon
 P3SEL0 |= SMCLK_OUT;          // Select 0 equals 1
 P3SEL1 &= ~SMCLK_OUT;         // Select 1 equals 0
 P3DIR  |= SMCLK_OUT;          // Direction set to output

 P3SEL0 |= OA3O;
 P3SEL1 |= OA3O;

 P3SEL0 |= OA2N;
 P3SEL1 |= OA2N;

 P3SEL0 &= ~IOT_LINK;
 P3SEL1 &= ~IOT_LINK;
 P3DIR &= ~IOT_LINK;

 P3SEL0 &= ~P3_7;
 P3SEL1 &= ~P3_7;
 P3DIR &= ~P3_7;

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

void Init_Port4(void){
//-------------------------------------------------------------------------
// Configure PORT 4, PINS 0-7

 P4DIR = 0xFF; // Set P1 direction to output
 P4OUT = 0x00; // P1 set Low
 P4SEL0 &= ~RESET_LCD; // RESET_LCD GPIO operation
 P4SEL1 &= ~RESET_LCD; // RESET_LCD GPIO operation
 P4DIR |= RESET_LCD; // Set RESET_LCD direction to output
 P4OUT |= RESET_LCD; // Set RESET_LCD Off [High]
 P4SEL0 &= ~SW1; // SW1 GPIO operation
 P4SEL1 &= ~SW1; // SW1 GPIO operation
 P4DIR &= ~SW1; // Direction = input
 P4PUD |= SW1; // Configure pullup resistor
 P4REN |= SW1; // Enable pullup resistor
 P4IES |= SW1; // P2.0 Hi/Lo edge interrupt
 P4IFG &= ~SW1; // Clear all P2.6 interrupt flags
 P4IE |= SW1; // P2.6 interrupt enabled
```

```
 P4SEL0 |= UCA1TXD; // USCI_A1 UART operation
 P4SEL1 &= ~UCA1TXD; // USCI_A1 UART operation
 P4SEL0 |= UCA1RXD; // USCI_A1 UART operation
 P4SEL1 &= ~UCA1RXD; // USCI_A1 UART operation
 P4SEL0 &= ~UCB1_CS_LCD; // UCB1_CS_LCD GPIO operation
 P4SEL1 &= ~UCB1_CS_LCD; // UCB1_CS_LCD GPIO operation
 P4DIR |= UCB1_CS_LCD; // Set SPI_CS_LCD direction to output
 P4OUT |= UCB1_CS_LCD; // Set SPI_CS_LCD Off [High]
 P4SEL0 |= UCB1CLK; // UCB1CLK SPI BUS operation
 P4SEL1 &= ~UCB1CLK; // UCB1CLK SPI BUS operation
 P4SEL0 |= UCB1SIMO; // UCB1SIMO SPI BUS operation
 P4SEL1 &= ~UCB1SIMO; // UCB1SIMO SPI BUS operation
 P4SEL0 |= UCB1SOMI; // UCB1SOMI SPI BUS operation
 P4SEL1 &= ~UCB1SOMI; // UCB1SOMI SPI BUS operation
 //-------------------------------------------------------------------------
}

void Init_Port5(void){
//-------------------------------------------------------------------------
// Configure PORT 5, PINS 0-4

 P5DIR = 0xFF; // Set P1 direction to output
 P5OUT = 0x00; // P1 set Low

 P5SEL0 |= IOT_RESET;
 P5SEL1 |= IOT_RESET;

 P5SEL0 |= P5_1;
 P5SEL1 |= P5_1;

 P5SEL0 |= IOT_PROG_SEL;
 P5SEL1 |= IOT_PROG_SEL;

 P5SEL0 |= IOT_PROG_MODE;
 P5SEL1 |= IOT_PROG_MODE;

 P5SEL0 &= ~IR_LED;
 P5SEL1 &= ~IR_LED;
 P5DIR |= IR_LED;
 P5OUT |= IR_LED;
 //-------------------------------------------------------------------------
}

void Init_Port6(void){
//-------------------------------------------------------------------------
// Configure PORT 6, PINS 0-7
```

```
 P6DIR = 0xFF; // Set P1 direction to output
 P6OUT = 0x00; // P1 set Low

 P6SEL0 |= R_FORWARD;
 P6SEL1 &= ~R_FORWARD;
 P6DIR |= R_FORWARD;

 P6SEL0 |= L_FORWARD;
 P6SEL1 &= ~L_FORWARD;
 P6DIR |= L_FORWARD;

 P6SEL0 |= R_REVERSE;
 P6SEL1 &= ~R_REVERSE;
 P6DIR |= R_REVERSE;

 P6SEL0 |= L_REVERSE;
 P6SEL1 &= ~L_REVERSE;
 P6DIR |= L_REVERSE;

 P6SEL0 &= ~LCD_BACKLITE;
 P6SEL1 &= ~LCD_BACKLITE;
 P6DIR |= LCD_BACKLITE;
 P6OUT |= LCD_BACKLITE;

 P6SEL0 &= ~P6_5;
 P6SEL1 &= ~P6_5;
 P6DIR &= ~P6_5;

 P6SEL0 &= ~GRN_LED;
 P6SEL1 &= ~GRN_LED;
 P6DIR |= GRN_LED;
 P6OUT &= ~GRN_LED;

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

9.6. Interrupt.c

```
//-------------------------------------------------------------------------
//
//  Description: This file contains the interrupt vector definitions for switch
// debouncing. SW1 and SW2 are sent to the ISR definitions in timers.c
// Added the state machine for ADC.
//
//  Steven Yan
//  Feb 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.1)
```

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


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

extern char display_line[DISPLAYFOUR][DISPLAYCOLUMNS];
extern char *display[DISPLAYFOUR];
extern volatile unsigned char display_changed;
volatile unsigned int debounce_count;
extern volatile char debounceSW1_TF;
extern volatile char debounceSW2_TF;
extern char buttonOne;
extern char buttonTwo;
extern unsigned int channel;
extern volatile unsigned int ADC_Thumb;
extern volatile unsigned int V_detect_r;
extern volatile unsigned int V_detect_l;

#pragma vector = PORT4_VECTOR          //debouncing switch 1
__interrupt void switchOne_interrupt(void) {
  if(P4IFG & SW1) {                  //if the bit for SW1 is flagged
    debounceSW1_TF = TRUE;           //indicate that the switch has been pressed
    buttonOne = TRUE;                //set global variable high for use in other functions
    P4IE &= ~SW1;                    //disable the switch interrupt enable
    TB0CCTL1 |= CCIE;                //enable the switch timer

  }
}


#pragma vector = PORT2_VECTOR
__interrupt void switchTwo_interrupt(void) {
  if(P2IFG & SW2) {
    debounceSW1_TF = TRUE;                  //start debouncing
    buttonTwo = TRUE;                       //set the global variable high
    TB0CCTL1 |= CCIE;                       //enable the switch timer
    P2IE &= ~SW2;                           //disable the switch interrupt enable
  }
}


#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void){
  switch(__even_in_range(ADCIV,ADCIV_ADCIFG)){
  case ADCIV_NONE:
```

```
    break;
  case ADCIV_ADCOVIFG: // When a conversion result is written to the ADCMEM0
    // before its previous conversion result was read.
    break;
  case ADCIV_ADCTOVIFG: // ADC conversion-time overflow
    break;
  case ADCIV_ADCHIIFG: // Window comparator interrupt flags
    break;
  case ADCIV_ADCLOIFG: // Window comparator interrupt flag
    break;
  case ADCIV_ADCINIFG: // Window comparator interrupt flag
    break;
  case ADCIV_ADCIFG: // ADCMEM0 memory register with the conversion result
   ADCCTL0 &= ~ADCENC;
    switch(channel++) {
      case THUMB:
        ADC_Thumb = ADCMEM0;    //save conversion result in mem register
        ADCMCTL0 &= ~ADCINCH_5; //turn off V_thumb
        ADCMCTL0 |= ADCINCH_2;  //L_detector
        break;

    case L_DET:
      V_detect_l = ADCMEM0;     //save conversion result in mem register
      ADCMCTL0 &= ~ADCINCH_2;   //turn off L_detector
      ADCMCTL0 |= ADCINCH_3;    //R_detector
      break;

    case R_DET:
      V_detect_r = ADCMEM0;     //save conversion result in mem register
      ADCMCTL0 &= ~ADCINCH_3;  //turn off R_detect
      ADCMCTL0 |= ADCINCH_5;   //V_thumb
      channel = THUMB;         //reset case to thumbwheel
      break;
    }
    ADCCTL0 |= ADCENC; //ADC enable conversion
    ADCCTL0 |= ADCSC; // Start next sample
    break;
  default:
    break;
   }
}

unsigned int channel;
void Init_ADC(void){


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

```
  // V_DETECT_L (0x04) // Pin 2 A2
  // V_DETECT_R (0x08) // Pin 3 A3
  // V_THUMB (0x20) // Pin 5 A5
  //-------------------------------------------------------------------------
  // ADCCTL0 Register
  ADCCTL0 = RESET_STATE; // Reset
  ADCCTL0 |= ADCSHT_2; // 16 ADC clocks
  ADCCTL0 |= ADCMSC; // MSC
  ADCCTL0 |= ADCON; // ADC ON
  // ADCCTL1 Register
  ADCCTL2 = RESET_STATE; // Reset
  ADCCTL1 |= ADCSHS_0; // 00b = ADCSC bit
  ADCCTL1 |= ADCSHP; // ADC sample-and-hold SAMPCON signal from sampling timer.
  ADCCTL1 &= ~ADCISSH;          // ADC invert signal sample-and-hold.
  ADCCTL1 |= ADCDIV_0;         // ADC clock divider - 000b = Divide by 1
  ADCCTL1 |= ADCSSEL_0;        // ADC clock MODCLK
  ADCCTL1 |= ADCCONSEQ_0;      // ADC conversion sequence 00b = Single-channel single-
conversion
  // ADCCTL1 & ADCBUSY identifies a conversion is in process
  // ADCCTL2 Register
  ADCCTL2 = RESET_STATE; // Reset
  ADCCTL2 |= ADCPDIV0; // ADC pre-divider 00b = Pre-divide by 1
  ADCCTL2 |= ADCRES_2; // ADC resolution 10b = 12 bit (14 clock cycle conversion time)
  ADCCTL2 &= ~ADCDF; // ADC data read-back format 0b = Binary unsigned.
  ADCCTL2 &= ~ADCSR; // ADC sampling rate 0b = ADC buffer supports up to 200 ksps
  // ADCMCTL0 Register
  ADCMCTL0 |= ADCSREF_0; // VREF - 000b = {VR+ = AVCC and VR– = AVSS }
  ADCMCTL0 |= ADCINCH_5; // V_THUMB (0x20) Pin 5 A5
  ADCIE |= ADCIE0; // Enable ADC conv complete interrupt
  ADCCTL0 |= ADCENC; // ADC enable conversion.
  ADCCTL0 |= ADCSC; // ADC start conversion.

}
```

9.7. Timers.c

```
void Init_Timer_B0(void)
{
  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, Interrupt happens every 50 msec
  TB0CCTL0 |= CCIE;                    // CCR0 enable interrupt

  // Capture Compare 1,2, Overflow
  // #pragma vector = TIMER0_B1_VECTOR

  // Capture compare 1
  TB0CCR1 = TB0CCR1_INTERVAL;       // CCR1 NOTE: THIS VALUE IS NOT IMPLEMENTED
CORRECTLY AT PRESENT
  TB0CCTL1 &= ~CCIE;                // CCR1 enable interrupt

  // Capture compare 2
  TB0CCR2 = TB0CCR2_INTERVAL;       // CCR2 NOTE: THIS VALUE IS NOT IMPLEMENTED
CORRECTLY AT PRESENT
  TB0CCTL2 &= ~CCIE;                // CCR2 enable interrupt

  // Overflow
  TB0CTL &= ~TBIE;                  // Disable Overflow Interrupt
  TB0CTL &= ~TBIFG;                  // Clear Overflow Interrupt flag
}
```

9.8. Serial_Com.c

Description: This file below contains the interrupts for both ports UCA0 and UCA1. When values are received in the UCA0 or UCA1, they are stored within the array USB_Char_Rx array from here they are acted upon within main and once the transmit interrupt is pulled for either UCA0 or UCA1 the data is transmitted out of the respective pins.

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

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

extern volatile unsigned int usb_rx_ring_wr;
extern volatile unsigned int usb_rx_ring_rd;
extern volatile char USB_Char_Rx[SMALL_RING_SIZE];
extern volatile unsigned int usb_tx_ring_wr;
extern volatile unsigned int usb_tx_ring_rd;
extern volatile char USB_Char_Tx[SMALL_RING_SIZE];
char test_command[16];
unsigned int UCA0_index;
unsigned int UCA1_index;
//-------------------------------------------------------------------------
```

```
#pragma vector=EUSCI_A0_VECTOR
__interrupt void eUSCI_A0_ISR(void){
unsigned int temp;
switch(__even_in_range(UCA0IV,0x08)){
case 0: // Vector 0 - no interrupt
break;
case 2: // Vector 2 – RXIFG
// code for Receive
  temp = usb_rx_ring_wr++;
  USB_Char_Rx[temp] = UCA0RXBUF; // RX -> USB_Char_Rx character

  if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
    usb_rx_ring_wr = BEGINNING; // Circular buffer back to beginning
  }
break;
case 4: // Vector 4 – TXIFG
// Code for Transmit
  switch(UCA0_index++){
    case 0: //
    case 1: //
    case 2: //
    case 3: //
    case 4: //
    case 5: //
    case 6: //
    case 7: //
    case 8: //
    UCA0TXBUF = test_command[UCA0_index];
      break;
    case 9: //
      UCA0TXBUF = 0x0D;
      break;
    case 10: // Vector 0 - no interrupt
      UCA0TXBUF = 0x0A;
      break;
    default:
      UCA0IE &= ~UCTXIE; // Disable TX interrupt
      break;
    }
  break;
default: break;
}
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
#pragma vector=EUSCI_A1_VECTOR
```

```
__interrupt void eUSCI_A1_ISR(void){
unsigned int temp;
switch(__even_in_range(UCA1IV,0x08)){
case 0: // Vector 0 - no interrupt
break;
case 2: // Vector 2 – RXIFG
// code for Receive
  temp = usb_rx_ring_wr++;
  USB_Char_Rx[temp] = UCA1RXBUF; // RX -> USB_Char_Rx character

  if (usb_rx_ring_wr >= (sizeof(USB_Char_Rx))){
    usb_rx_ring_wr = BEGINNING; // Circular buffer back to beginning
  }
break;
case 4: // Vector 4 – TXIFG
// Code for Transmit
  switch(UCA1_index++){
    case 0: //
    case 1: //
    case 2: //
    case 3: //
    case 4: //
    case 5: //
    case 6: //
    case 7: //
    case 8: //
    UCA1TXBUF = test_command[UCA1_index];
      break;
    case 9: //
      UCA1TXBUF = 0x0D;
      break;
    case 10: // Vector 0 - no interrupt
      UCA1TXBUF = 0x0A;
      break;
    default:
      UCA1IE &= ~UCTXIE; // Disable TX interrupt
      break;
    }
  break;
default: break;
}
}
//---------------------------------------------------------------------------
```

Description

This file contains the initialization of the serial communication ports UCA0 and UCA1. Both of these ports are initialized to have a baud rate of 460,800. Once these functions have been performed serial communication receiving and transmitting with either UCA0 or UCA1 can be performed.

```
//-------------------------------------------------------------------------
void Init_Serial_UCA0(void){
  int i;
  for(i=0; i< LARGE_RING_SIZE; i++){
  USB_Char_Rx[i] = 0x00; // USB Rx Buffer
  }
  usb_rx_ring_wr = BEGINNING;
  usb_rx_ring_rd = BEGINNING;

  for(i=0; i < SMALL_RING_SIZE; i++){ // May not use this
  USB_Char_Tx[i] = 0x00; // USB Tx Buffer
  }

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

  UCA0BRW = 1; // 460,800 Baud
  UCA0MCTLW = 0x4911 ;

UCA0CTLW0 &= ~ UCSWRST; // Set Software reset enable
UCA0IE |= UCRXIE; // Enable RX interrupt
}

void Init_Serial_UCA1(void){
  int i;
  for(i=0; i< LARGE_RING_SIZE; i++){
  USB_Char_Rx[i] = 0x00; // USB Rx Buffer
  }
  usb_rx_ring_wr = BEGINNING;
  usb_rx_ring_rd = BEGINNING;

  for(i=0; i < SMALL_RING_SIZE; i++){ // May not use this
  USB_Char_Tx[i] = 0x00; // USB Tx Buffer
  }

  usb_tx_ring_wr = BEGINNING;
  usb_tx_ring_rd = BEGINNING;
  // Configure UART 0
  UCA1CTLW0 = 0; // Use word register
  UCA1CTLW0 |= UCSWRST; // Set Software reset enable
```

```
  UCA1CTLW0 |= UCSSEL__SMCLK; // Set SMCLK as fBRCLK

  UCA1BRW = 1; // 460,800 Baud
  UCA1MCTLW = 0x4911 ;

UCA1CTLW0 &= ~ UCSWRST; // Set Software reset enable
UCA1IE |= UCRXIE; // Enable RX interrupt
}
```

9.9. Display.c

```
/-------------------------------------------------------------------------
Description: This function runs the display of my device. Flags are raised and positions are changed
throughout the running of my main function some of these changes are reflected on to  what the display
shows. The display is updated every 200 millisecs.
/-------------------------------------------------------------------------
unsigned char state;

extern char display_line[4][11];
extern char *display[4];
extern unsigned char display_mode;
extern volatile unsigned char display_changed;
extern volatile unsigned char update_display;

extern unsigned int timer_count;
extern unsigned int timer_update;
unsigned char position;

unsigned int movement_type;
unsigned int display_type;
unsigned int position_type;

extern char IP_address1[THREE];
extern char IP_address2[THREE];
extern char IP_address3[THREE];
extern char IP_address4[THREE];

void Update_Display(){
    strcpy(display_line[POS0], "          ");
    update_string(display_line[POS0], ZERO);
    strcpy(display_line[POS1], "          ");
    update_string(display_line[POS1], ONE);
    strcpy(display_line[POS2], "          ");
    update_string(display_line[POS2], TWO);
    strcpy(display_line[POS3], "          ");
    update_string(display_line[POS3], THREE);

      display_line[POS1][POS1] = IP_address1[POS0];
```

```
    display_line[POS1][POS2] = IP_address1[POS1];
    display_line[POS1][POS3] = IP_address1[POS2];

    display_line[POS1][POS5] = IP_address2[POS0];
    display_line[POS1][POS6] = IP_address2[POS1];
    display_line[POS1][POS7] = IP_address2[POS2];

    display_line[POS2][POS1] = IP_address3[POS0];
    display_line[POS2][POS2] = IP_address3[POS1];
    display_line[POS2][POS3] = IP_address3[POS2];

    display_line[POS2][POS5] = IP_address4[POS0];
    display_line[POS2][POS6] = IP_address4[POS1];
    display_line[POS2][POS7] = IP_address4[POS2];

  switch(display_type){
  case IP:
    strcpy(display_line[POS0], "Wait4Input");
    update_string(display_line[POS0], ZERO);
    break;
  case MOVEMENT:
    switch(movement_type){
    case FORWARD:
      strcpy(display_line[POS0], " Forward  ");
      update_string(display_line[POS0], ZERO);
      break;
    case BACKWARD:
      strcpy(display_line[POS0], " Backward ");
      update_string(display_line[POS0], ZERO);
      break;
    case LEFT_45:
      strcpy(display_line[POS0], " Left 45  ");
      update_string(display_line[POS0], ZERO);
      break;
    case LEFT_90:
      strcpy(display_line[POS0], " Left 90  ");
      update_string(display_line[POS0], ZERO);
      break;
    case RIGHT_45:
      strcpy(display_line[POS0], " Right 45 ");
      update_string(display_line[POS0], ZERO);
      break;
    case RIGHT_90:
      strcpy(display_line[POS0], " Right 90 ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_START:
```

```
      strcpy(display_line[POS0], " BL Start ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_TURN:
      strcpy(display_line[POS0], "Intercept ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_TRAVEL:
      strcpy(display_line[POS0], "BL Travel ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_CIRCLE:
      strcpy(display_line[POS0], "BL Circle ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_EXIT:
      strcpy(display_line[POS0], " BL Exit  ");
      update_string(display_line[POS0], ZERO);
      break;
    case BL_STOP:
      strcpy(display_line[POS0], " BL Stop  ");
      update_string(display_line[POS0], ZERO);
      break;
    }
    break;
  case POSITION:
    lcd_4line();
    switch(position_type){
    case POSITION_1:
      strcpy(display_line[POS0], "Position 1");
      update_string(display_line[POS0], ZERO);
      break;
    case POSITION_2:
      strcpy(display_line[POS0], "Position 2");
      update_string(display_line[POS0], ZERO);
      break;
    case POSITION_3:
      strcpy(display_line[POS0], "Position 3");
      update_string(display_line[POS0], ZERO);
      break;
    case POSITION_4:
      strcpy(display_line[POS0], "Position 4");
      update_string(display_line[POS0], ZERO);
      break;
    case POSITION_5:
      strcpy(display_line[POS0], "Position 5");
      update_string(display_line[POS0], ZERO);
```

```
        break;
      case POSITION_6:
        strcpy(display_line[POS0], "Position 6");
        update_string(display_line[POS0], ZERO);
        break;
      case POSITION_7:
        strcpy(display_line[POS0], "Position 7");
        update_string(display_line[POS0], ZERO);
        break;
      case POSITION_8:
        strcpy(display_line[POS0], "Position 8");
        update_string(display_line[POS0], ZERO);
        break;

      }
      break;
    case WAITING:
      strcpy(display_line[POS0], " Waiting  ");
      update_string(display_line[POS0], ZERO);
      break;
    }
    Display_Time();             // Displays run time on screen

    if(timer_count >= DISPLAY_UPDATE_WAIT){         // Update display every half sec or 500 msec
      update_display = ONE;
      timer_count = RESET;
      timer_update = RESET;
  }
     else{
      timer_update = 1;
      display_changed = 1;
     }
}
```

9.10. Movements.c

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

Description: This function controls all the movements for my vehicle. These functions are referenced throughout my code based upon the input gathered from my IOT. The main movement commands I used were forward, reverse, clockwise spin, and counterclockwise spin.

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

void Run_Forward(void){
  TB3CCTL1 = OUTMOD_7;
  TB3CCTL2 = OUTMOD_7;
```

```
  TB3CCTL3 = OUTMOD_7;
  TB3CCTL4 = OUTMOD_7;

  RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM OFF
  RIGHT_FORWARD_SPEED = RIGHT_FORWARD_AMOUNT; // P6.0 Right Forward PWM ON
amount

  LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM OFF
  LEFT_FORWARD_SPEED = LEFT_FORWARD_AMOUNT; // P6.1 Left Forward PWM ON
amount

  RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM OFF
  RIGHT_REVERSE_SPEED = [DESIRED ON AMOUNT]; // P6.2 Right Reverse PWM ON amount

  LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM OFF
  LEFT_REVERSE_SPEED = [DESIRED ON AMOUNT]; // P6.3 Left Reverse PWM ON amount
}

void Run_Reverse(void){
  TB3CCTL1 = OUTMOD_7;
  TB3CCTL2 = OUTMOD_7;
  TB3CCTL3 = OUTMOD_7;
  TB3CCTL4 = OUTMOD_7;

  RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM OFF

  LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM OFF

  RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM OFF
  RIGHT_REVERSE_SPEED = RIGHT_REVERSE_AMOUNT; // P6.2 Right Reverse PWM ON
amount

  LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM OFF
  LEFT_REVERSE_SPEED = LEFT_REVERSE_AMOUNT; // P6.3 Left Reverse PWM ON amount
}

void Run_SpinCW(void){
  TB3CCTL1 = OUTMOD_7;
  TB3CCTL2 = OUTMOD_7;
  TB3CCTL3 = OUTMOD_7;
  TB3CCTL4 = OUTMOD_7;

  RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM OFF

  LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM OFF
  LEFT_FORWARD_SPEED = LEFT_SPIN_CW_AMOUNT; // P6.1 Left Forward PWM ON amount
```

```
  RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM OFF
  RIGHT_REVERSE_SPEED = RIGHT_SPIN_CW_AMOUNT; // P6.2 Right Reverse PWM ON
amount

  LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM OFF
}

void Run_SpinCCW(void){
  TB3CCTL1 = OUTMOD_7;
  TB3CCTL2 = OUTMOD_7;
  TB3CCTL3 = OUTMOD_7;
  TB3CCTL4 = OUTMOD_7;

  RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM OFF
  RIGHT_FORWARD_SPEED = RIGHT_SPIN_CCW_AMOUNT ; // P6.0 Right Forward PWM ON
amount

  LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM OFF

  RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM OFF

  LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM OFF
  LEFT_REVERSE_SPEED = LEFT_SPIN_CCW_AMOUNT ; // P6.3 Left Reverse PWM ON amount

}

void Run_Pause(void){
  TB3CCTL1 = OUTMOD_7;
  TB3CCTL2 = OUTMOD_7;
  TB3CCTL3 = OUTMOD_7;
  TB3CCTL4 = OUTMOD_7;

  RIGHT_FORWARD_SPEED = WHEEL_OFF; // P6.0 Right Forward PWM OFF
  LEFT_FORWARD_SPEED = WHEEL_OFF; // P6.1 Left Forward PWM OFF
  RIGHT_REVERSE_SPEED = WHEEL_OFF; // P6.2 Right Reverse PWM OFF
  LEFT_REVERSE_SPEED = WHEEL_OFF; // P6.3 Left Reverse PWM OFF
}
```

9.11. A0_processes.c

Description:

This function takes in the data coming from the UC A0 Rx Buffer and iterates over an array to hold the data sequentially as it comes into the device. When the A0 pin receives the character ‘ ` ’, the array is set to zero. This is useful because the first 10 indexes of the array are displayed on the third line of the LCD display. Furthermore, it allows you to always know which index the following commands will be.

```
void A0_processes(void){
unsigned int snapshotwr_A0=usb_rx_ring_wr_A0;
 while(usb_rx_ring_rd_A0 != snapshotwr_A0){
  if(USB_Char_Rx_A0[usb_rx_ring_rd_A0]=='`'){
    UCA0_index = ZERO;
  }
  A0_ProcessBuffer[UCA0_index] = USB_Char_Rx_A0[usb_rx_ring_rd_A0];
  TransRec = TWO;
  UCA0_index++;
  usb_rx_ring_rd_A0++;
  display_line[TWO][ZERO] =  A0_ProcessBuffer[ZERO];
  display_line[TWO][ONE] =   A0_ProcessBuffer[ONE];
  display_line[TWO][TWO] =   A0_ProcessBuffer[TWO];
  display_line[TWO][THREE] = A0_ProcessBuffer[THREE];
  display_line[TWO][FOUR] =  A0_ProcessBuffer[FOUR];
  display_line[TWO][FIVE] =  A0_ProcessBuffer[FIVE];
  display_line[TWO][SIX] =   A0_ProcessBuffer[SIX];
  display_line[TWO][SEVEN] = A0_ProcessBuffer[SEVEN];
  display_line[TWO][EIGHT] = A0_ProcessBuffer[EIGHT];
  display_line[TWO][NINE] =  A0_ProcessBuffer[NINE];
  update_string(display_line[TWO], TWO);
  display_changed = ONE;
  if(UCA0_index >= (sizeof(A0_ProcessBuffer))){
    UCA0_index = ZERO;
  }
  if (usb_rx_ring_rd_A0 >= (sizeof(USB_Char_Rx_A0))){
    usb_rx_ring_rd_A0 = ZERO;        //Circulate buffer to beginning
  }
 }
}
```

9.12. Forward.c

Description:

This one of the primary functions of the robot. This function reads the input command. Once it receives the “F” command, standing for “forward”, the function ensures that the wheels’ reverse functionality is turned off, and the forward functionality is set to 100% speed using pulse width modulation, and that the Forward Timer gets reset to zero. Once this timer reaches 1 second, the function deletes the “F” command and turns off all of the wheels.

```
void Forward(void){
 if(A0_ProcessBuffer[1] == 'F'){
  ForwardTimer = 0;
  RIGHT_REVERSE_SPEED = WHEEL_OFF;
  LEFT_REVERSE_SPEED = WHEEL_OFF;
  RIGHT_FORWARD_SPEED = 40000;
  LEFT_FORWARD_SPEED = 40000;
  if(Initial == 0){
    ForwardTimer = 0;
```

```
    Initial = 1;
   }
   ForwardCommandSent = 1;
  }
  if((ForwardTimer >= 1) && (ForwardCommandSent == 1)){
   A0_ProcessBuffer[1] = '~';
   RIGHT_FORWARD_SPEED = WHEEL_OFF;
   LEFT_FORWARD_SPEED = WHEEL_OFF;
   RIGHT_REVERSE_SPEED = WHEEL_OFF;
   LEFT_REVERSE_SPEED = WHEEL_OFF;
   A0_ProcessBuffer[1] = '~';
   Initial = 0;
   ForwardCommandSent = 0;
  }
}
```

9.13. menu.c

Description: The menu function in my domain works by polling for button presses on switch one. After it receives a button press it increments a variable called menu_iitem and enters a switch statement that prints the corresponding menu item onto the screen. The user has the option to choose calibrate values, network configuration, IOT course, or test black line. When the user decides on the option that they want, the code will flag a variable allowing the sub function to run. Pressing the second button will select the onscreen option. The calibrate screen displays the ADC values for the left and right detector. Pressing button two again will save the onscreen values to be used for the black line trace.

Selecting star IOT will update the display with relevant information the car will immediately begin processing input from the PC and opening the TCP client on the computer will allow you to send commands to your car. Just Porsche in the car will specifically look for the secret key followed by the forward reverse right or left commands these will be directly translated into motion for the car. At the top of the display the car will show the corresponding IOT pad that it has arrived at on the next two lines it will show the IP address on the final line it will show the current receive command. After successful navigation of the IOT course pressing the black line command key will begin the black line intercept.

One extra command that I put on my menu was black line test this allowed me to test my black line function without actually connecting my IOT module I found this extremely helpful during the testing portion of my car because I did not need to configure the network settings each time I needed to test.

```
//-------------------------------------------------------------------------
//
// If button one is pressed. Update the menu screen. Check for overflow and if
// so then reset menu item. The menu function just sets flags to true so that
// other functions will start to run.
//
//
//-------------------------------------------------------------------------
```

```
void menu() {

 if(buttonOne) {
  ++menu_item;
  menuprint = TRUE;
  // Clear Display
  displayLine(DISPLAYTHREE,"         ",LEFTALIGN);
  displayLine(DISPLAYONE,"         ",LEFTALIGN);
  displayLine(DISPLAYTHREE,"         ",LEFTALIGN);
  displayLine(DISPLAYTWO,"         ",LEFTALIGN);
  displayLine(DISPLAYTHREE,"         ",LEFTALIGN);
  buttonOne = RESET_STATE;
  //reset the menu if overflow
  if(menu_item > BLACKLINETESTOVERFLOW) { menu_item = RESET_STATE; }
 }

 switch(menu_item) {
 case CALIBRATE:

  if(menuprint) {
   displayLine(DISPLAYZERO," CALIBRATE",LEFTALIGN); //print text
   menuprint = FALSE;
  }
  detectorValues();
  break;

 case CONNECT:
  if(menuprint) {
   displayLine(DISPLAYZERO," NETWORK  ",LEFTALIGN); //print text
   menuprint = FALSE;
  }
  break;

 case IOTCOURSE:
  if(menuprint) {
   displayLine(DISPLAYZERO,"START IOT ",LEFTALIGN);  //print text
   menuprint = FALSE;
  }
  break;
   case BLACKLINETRACE:
      if(menuprint) {
    displayLine(DISPLAYZERO," BLACKLINE", LEFTALIGN); //print text
   menuprint = FALSE;
      }
    break;
 default: break;
 }
```

```
unsigned int tmp1 = V_detect_r;
unsigned int tmp2 = V_detect_l;

//    if(Time_Sequence >= TOGGLEPERIOD) {
//      displayValues = RESET_STATE;
//      displayToggle = RESET_STATE;
//     displayLine(DISPLAYTWO,"          ",LEFTALIGN);
//      displayLine(DISPLAYTHREE,"          ",LEFTALIGN);
//      P6OUT &= ~LCD_BACKLITE;
//    }

if(buttonTwo) {
  Time_Sequence = RESET_STATE;
  buttonTwo = FALSE;
  switch(menu_item) {
  case CALIBRATE:
    displayLine(DISPLAYZERO,"-CALIBRATE",LEFTALIGN); //print text

    if(!blackLine) {
      blackLine = (tmp1 + tmp2) >> DIVIDE;      //save value for black
      displayLine(DISPLAYTHREE,"B:",LEFTALIGN); //print text
      displayLine(DISPLAYTHREE,HEXtoBCD(blackLine),SHIFTTWO); //print text
    }

        else {
          whiteSpace = (tmp1 + tmp2) >> DIVIDE;   //save whitespace value
          displayLine(DISPLAYTHREE,"W:",LEFTALIGN); //print text
          displayLine(DISPLAYTHREE,HEXtoBCD(whiteSpace),SHIFTTWO); //print text

        }
    break;


  case CONNECT:
    broadcastcommand = TRUE;   // set flag to true
    break;

  case IOTCOURSE:
   displayLine(DISPLAYZERO," WAITING  ", LEFTALIGN); //print waiting to start
   startIOTflag = TRUE;       //set flag to true
    break;


  case BLACKLINETRACE:
     displayLine(DISPLAYZERO," BL START ", LEFTALIGN);   //print to screen
     interception = BLACKLINETEST;       //start tracing black line
```

```
      cycle_time = RESET_STATE;           //Reset Timer
     break;


   default: break;
   }
 }
}
```

9.14. networkconfig.c

Description: The Network configuration gets the IP address that the IOT module sends to the FRAM when a connection is established. The FRAM will then send three back to back commands setting up the port number, syncinterval, and ping google.

```
void networkconfig(void) {
  switch(broadcastcommand) {
 case PARSEIP:
  if(!commandsent) {
   TX_string0(getIP);       //parse the IP
   commandsent = TRUE;
  }
  if(cycle_time > COMMANDTIME ) {
   broadcastcommand = SYNCSTATE;   //move to next state
   cycle_time = RESET_STATE;
    commandsent = RESET_STATE;

  }
   break;

 case SYNCINTERVAL:
    if(!commandsent) {
  TX_string0("AT+WSYNCINTRL=65535\n");  //set syncinterval
  commandsent = TRUE;
    }
  if(cycle_time > COMMANDTIME ) {   //if over wait time
     broadcastcommand = PORTSTATE;   //move to next state
     cycle_time = RESET_STATE;
     commandsent = RESET_STATE;
  }
  break;
   case PORTCONFIG:
   if(!commandsent) {
  TX_string0("AT+NSTCP=6667,1\n");      //command to set port to 6667
   commandsent = TRUE;
   }
  if(cycle_time > WAITFORCONFIG ) {  //if over wait time
     broadcastcommand = PINGSTATE;    //move to next state
```

```
    cycle_time = RESET_STATE;
    commandsent = RESET_STATE;
   }

   break;

 case PINGGOOGLE:
    if(!commandsent) {
   TX_string0("AT+PING=google.com,3\n");   //command to ping google
    commandsent = TRUE;
    }
    if(cycle_time > COMMANDTIME ) {  //if over wait time
     broadcastcommand = FALSE;   //move to next state
     cycle_time = RESET_STATE;
     commandsent = RESET_STATE;
   }
   break;

 default: break;
 }
}
```

9.15. irconfig.c

Description: This function contains the functions which average the values for a white IR value and a black IR value. Both of these functions work the exact same, however one saves its value for a white line value and the other saves its value for a black line value. To get this average, while our amount of samples is lower than the desired amount already set, we continually add the current ADC left/right value to its respective left/right value, then divide by two. Our amount of samples index is increased by one after this is done then this function is repeated again. Once this has finished, there will be a left and right average value for the respective black or white value.

```
extern char display_line[DISPCOLUMN][DISPCHAR];

extern unsigned int ADC_Det_L;
extern unsigned int ADC_Det_R;
extern char state;

unsigned int IR_Ambient_L = RESET_REGISTER;
unsigned int IR_Ambient_R = RESET_REGISTER;
unsigned int IR_White_L = RESET_REGISTER;
unsigned int IR_White_R = RESET_REGISTER;
unsigned int IR_Black_L = RESET_REGISTER;
unsigned int IR_Black_R = RESET_REGISTER;
unsigned int average = RESET_REGISTER;
unsigned int Avg_Count = RESET_REGISTER;
```

```
//This function takes an average reading of both IR sensors when the IR LED is on and over white
void white_ir_config(void)
{
 P5OUT |= IR_LED;
   //*********MUST TURN ON IR LED BEFORE CALLING THIS
FUNCTION**********************************
 Avg_Count = RESET_REGISTER;                              //Reset any previous count
 average = RESET_REGISTER;
 while(Avg_Count < IR_DETECTION_SAMPLE_RATE)              //While our average count is
lower than our set sample rate, average both detectors then add them to the current average of white
varaible.
 {
  if(Avg_Count != RESET_REGISTER)
  {
   IR_White_L = (IR_White_L + ADC_Det_L) / HALF_AVG;
   IR_White_R = (IR_White_R + ADC_Det_R) / HALF_AVG;
   Avg_Count++;
  }
  else
  {
   IR_White_L = ADC_Det_L;                                //Set the default white variable to the first
average so that we do not divide by two for our first detections
   IR_White_R = ADC_Det_R;
   Avg_Count++;
  }
 }
}

//This function takes an average reading of both IR sensors when the IR LED is on and over black
void black_ir_config(void)
{
 P5OUT |= IR_LED;
   //*********MUST TURN ON IR LED BEFORE CALLING THIS
FUNCTION**********************************
 Avg_Count = RESET_REGISTER;                              //Reset any previous count
 average = RESET_REGISTER;
 while(Avg_Count < IR_DETECTION_SAMPLE_RATE)              //While our average count is
lower than our set sample rate, average both detectors then add them to the current average of black
varaible.
 {
  if(Avg_Count != RESET_REGISTER)
  {
   IR_Black_L = (IR_Black_L + ADC_Det_L) / HALF_AVG;
   IR_Black_R = (IR_Black_R + ADC_Det_R) / HALF_AVG;
   Avg_Count++;
  }
  else
```

```
  {
   IR_Black_L = ADC_Det_L;                          //Set the default black variable to the first
average so that we do not divide by two for our first detections
   IR_Black_R = ADC_Det_R;
   Avg_Count++;
  }
 }
}
```

9.16. switch_processes.c

Description: This function establishes what happens when the buttons on the processor are pressed. The left button controls the menu that is displayed and the functions that can be accessed while the button on the right works as a selection button. When either button is not pressed, the debounce flag variable is set to off. When the right button is pressed and the debounce flag is off, we set the select_function variable to on. This variable is set off in the function that it turns on. When the left button is pressed, we increment the menu state variable. If the menu state variable is greater than the max number of menu states, we reset this variable to zero. When either the left or the right button is pressed we set the debounce flag on. When both the time has changed and the debounce flag is on, we wait for a set amount of time before we allow the buttons to be pressed again.

```
extern char display_line[NUMOFLINES][MAXCHRBREAK];
extern volatile unsigned char display_changed;
extern unsigned int pindef;
extern char state;
extern unsigned int debounce_flag;
extern unsigned int time_change;
int delay_button = RESET_STATE;
unsigned int menu_state = RESET_STATE;
char select_function = BUTTONNOTPRESSED;

void Switches_Process(void)
{
 if((P4IN & SW1) && (P2IN & SW2))
 {
  debounce_flag = RESET_STATE;
 }

 if(time_change && debounce_flag == RESET_STATE)                    //if we detect that there has
been an increase in time and we need to debounce
 {
  time_change = RESET_STATE;                //reset our detection variable
  if(delay_button++ >= WAITING4BUTTON)        //if we detect that there has been a increase in time
larger than our set waiting time
  {
   delay_button = RESET_STATE;              //reset our long term set waiting variable
   debounce_flag = TURNON;                    //set our button timer back to 1
```

```
    select_function = BUTTONNOTPRESSED;
  }
 }

 if(!(P4IN & SW1) && debounce_flag != RESET_STATE)
 {
  debounce_flag = RESET_STATE;
  delay_button = RESET_STATE;
  select_function = BUTTONPRESSED;
 }

 if(!(P2IN & SW2) && debounce_flag != RESET_STATE)
 {
  debounce_flag = RESET_STATE;
  delay_button = RESET_STATE;
  if(menu_state >= MAXMENUSTATE-1)
  {
   menu_state = RESET_STATE;
  }
  else
  {
   menu_state++;
  }
 }
}
```

10. Conclusion

Embedded systems were one of the most impactful classes to date that I have taken in my college career. I really got to learn in depth how software interacts with hardware and all the different input and output devices that you can give to a microprocessor.

We began the class with the PCB, and I learned about surface mount devices and the reflow process. I got a lot of practice with through hole soldering and doing point to point tests. This knowledge will be very useful for us later on in our careers.

One of the coolest concepts I learned in this class was pulse width modulation. This is how you turn a device on for a percentage of the given time period.
Throughout the class, I learned how to make my operating system more efficient and how to turn devices off after not using them.

Knowing what to do with the output from the devices attached to our car was also interesting to learn. The black line detector circuit was the first device that we got to work with that gave data back to our car. Making the car follow a black line seemed like a tremendous task at first but then after thinking about the process at a very low level it was actually super easy.

Serial Communication and the IOT project were considered to be the most difficult by our team. Getting our board to communicate with another device seemed to be simple at first but many of us were having a hard time processing the data because our code was running too slowly. Many of us had to revisit older functions in our code that were slowing the overall system down and causing data loss. Eventually everyone got serial communications and their IOT module to work properly.

We finished off the course with a project that involved tying together all of the previous projects. We got the opportunity to work with communicating with our car through the internet. Some of us found it easier to write a menu that our cars could interact with. Overall Embedded Systems was a really great class.

An early issue that could have been avoided if we were more careful was shorting the mosfets on the H-Bridge. This occurred because in our code we were writing functions that were making the car go forward and reverse at the same time. This essentially shorts the H-Bridge and causes the mosfets to heat up. The amount of damage varied between cars some of us were able to continue without replacing anything. Other members had to replace some mosfets because they were no longer working.

Another issue that our group had was IOT modules shorting themselves. The mounting pins that the IOT module has was drilling into the ground plane in the PCB. Two members in our group experienced this problem and the fix was to lift the IOT module up and cover the PCB with insulating tape.

One thing that went wrong this semester was the power system for the car. The issue revolved around the car shutting off the moment the motors of the car switched on. My group spent many hours trying to figure out a solution to the problem. At first, we thought that the motors where pulling too much current that the battery pack could not support it which in turn caused the processor to brownout. Some members of our group implemented a separate power supply for the motors to try to isolate the issue, but this did not fix the issue.