Autonomous Line-Following Vehicle

Hands-Free Transportation on a Testable Scale

Car Pack – Hayes, Yu, Ryle, Gretsch

Team Members
Jordan Hayes
Joseph Yu
Benjamin Ryle
Rhys Gretsch

Originator: Jim Carlson
Checked: 12/05/2019 | Released: 12/05/2019
Filename: Team9_Writeup

Title: **Autonomous Line-Following Vehicle The Complete Process and Construction**

Date:	Document Number:	Rev:	Sheet:
12/05/2019	**1-4536-153-1545-15**	**4D**	**1 of 35**

Revision	Description
1A	Basic Test Process outlined, stating only tests mentioned inside of project notes. (need to add alternative diode test that was used)
1B	Added beginnings of a rough draft for the scope portion
1C	Added hardware section
1D	Added Overview section
1E	Added test verification method and notes about brownout protection circuit and rephrased scope passage
1F	Changed the order of sections to meet requirements and made formatting uniform throughout
2A	Extended Test process to include LCD screen and forward directional FET's
2B	Expanded on the Hardware section, adding the chassis and controls system, as well as adding schematics
2C	Added Abbreviations section, minor format Adjustments
2D	Inserted pictures missing from the Testing Section
3A	Updated software portion and made Port flowchart
3B	Hardware section & Timer flowchart
3C	Testing section and Main flowchart
3D	Made document reflect new submission and added interrupt flowchart
4A	Divided Software Description into subsections
4B	Software listing of ADC and Main
4C	Added Software Listing and Serial Flow chart
4D	Reformatted document for previous lost points
4A	Removed old flowcharts, created 5 flowcharts
4B	Updated date, times, removed software
4C	Comprehensive conclusion, and flow chart changes
4D	Formatting conclusion, rewriting layout

Table of Contents

Figures.

1. Scope

The contents of this document describe the construction, programming, and testing of an electronic vehicle designed to follow a path made of black electrical tape. The construction of the car involved soldering together an MSP430 control board and three custom PCBs. The programming of the car's internal logic was written and validated through the program IAR. This project currently supports battery and micro-USB power input, and an LCD screen text display.

2. Abbreviations

FRAM	Ferroelectric Random Access Memory
FET	Field-Effect Transistor
GND	Ground
H-Bridge	A circuit used to switch voltage polarity: in use in our motor
IAR	References the program "IAR Embedded Workbench" used to program the FRAM board
IR	Infrared- a type of high frequency light that isn't visible to the naked eye
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MOSFET	Metal Oxide Semi-Conductor
N-FET	Negative Field Effect Transistor
P-FET	Positive Field Effect Transistor
RISC	Reduced Instruction-set Computer
USB	Universal Serial Bus

3. Overview

The system is connected through soldered joints. The major parts that are connected include the FRAM Board, Power Board, IR board, LCD Display, H-Bridge, motors, wheels, chassis and the IAR Software. These all interact with each other to power the entire system, make the system function properly, and have the system display text for the user.

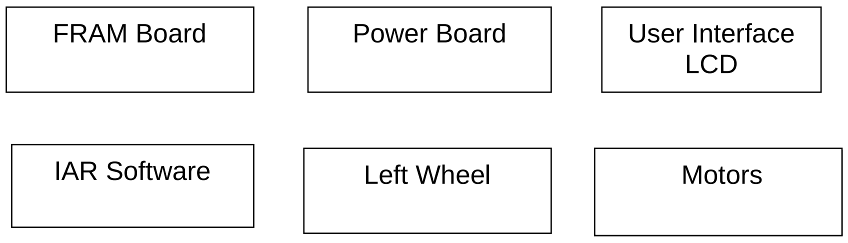


Figure 1 Block diagram

3.1. FRAM Board

The FRAM board is an MSP430FR2355. This board can be powered either through battery connection through the power board or through USB power. The power board can fully power the FRAM board through a battery pack, but the system can only operate with the available information that has been stored in the microcontroller's memory on the FRAM board. The board contains switches on each side that controls what is displayed on the user interface of the system. The FRAM board also receives information and data from the IAR software when connected to a computer via USB. The FRAM board uses this information to operate all the different functions that are currently available to the system up to this point. Some of these functions include the programming of what the switches can do to the display, and the text that is being displayed when the system is operating. The ports with pins on the FRAM board also have many different functions. These port pins can each be programmed through the IAR software to control which of these functions the ports will utilize. Up to this point, we've utilized port pins that involve powering on the motors that are connected through the H-Bridge board, and programming the switches to perform objectives such as turning on the motors per press and navigating through a set of text on the UI.

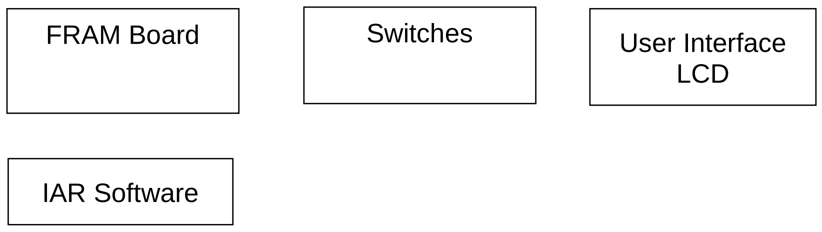


Figure 2 Block 1, FRAM Board

3.2. Power Board

The power board is used to power the entire system. The power board is connected to the FRAM board and the LCD user interface. The FRAM board is under the power board while the LCD user interface is resting on top of it. The power board can be powered by 4 to 6"AA" batteries through its connection with a mounted connector on the power board. The power board uses the voltage from the batteries and powers the motors that are connected through the H-Bridge board.

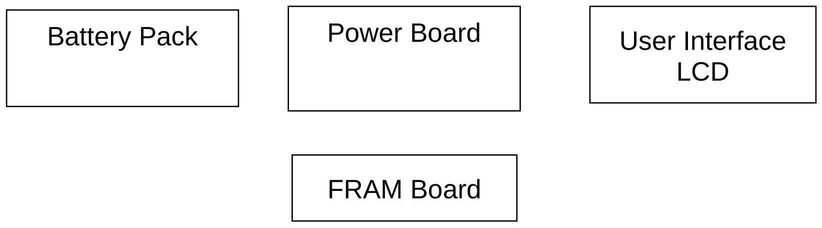
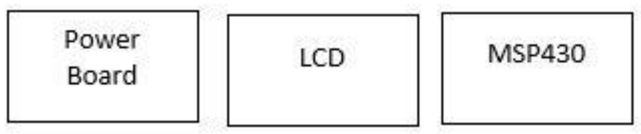


Figure 3 Block 2, Power Board

3.3. User Interface LCD

The LCD user interface is soldered on top of the power board. And runs off the data from the connected FRAM board and the power from the power board. The LCD user interface lights up and displays the IP address it's assigned from the Wi-Fi network, the word 'OK', when it's established a tcp server, a timer, and what commands it's currently following. The LCD display text can be edited through IAR Software, and follow the Menu state machine.



3.4. Control System

The H-Bridge is connected through the bottom of the FRAM board's port pins. It contains three output connectors, two of which are used to connect the motors to the entire system. The H-Bridge functions – at this point – to get forward motion working properly with the system by using N-FETs to control the power to the motors.

The motors are connected to the H-Bridge's output connectors. They are powered through the battery pack's output voltage. The motors are set to operate in forward or reverse motion, depending on the PWM values assigned in software. They can be programmed to operate on a switch command through the IAR software code set in the FRAM board. They have two points at which they need to be connected. The wires that are connected to the motors are colored red and black. Depending on the polarity of the wires set on your motor will determine which way your motors will turn.

The wheels are connected to the part of the motor that turns. These wheels can either have a tire around them or tank treads. The choice is up to the user. Treads will need an extra set of wheels to be installed on the back of the chassis. These wheels will affect how quickly the system moves and how precise the system's movements will be. The caster is only needed if your system does not have treads. The caster's purpose is to hold up the rear end of the chassis so that your system will be able to move with minimal drag. The caster will also affect how precise your system's movements will be in the future.

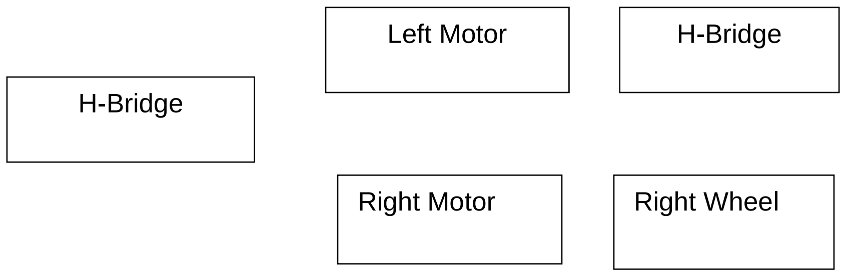
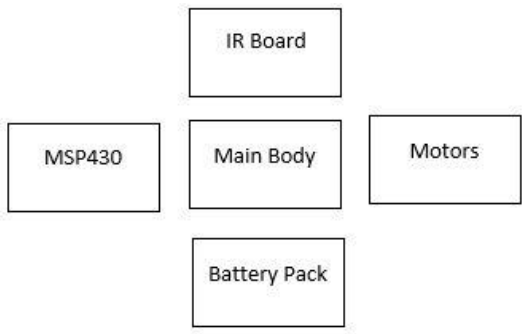


Figure 4 Block diagram

3.5. Chassis

The purpose of the chassis is to hold all the parts of the system up and function as a car frame. The chassis can come in many different sizes offered by the course or even made entirely on the user's own. The chassis is important because it will affect the movement of how the system moves as a user progresses through the course.



3.7 IR Board

The IR Board's job is to receive and transmit IR light and send whatever it receives to the FET board, and from there, to the FRAM board to be interpreted to determine the existence of the black line, and the directionality of the car chassis as a whole.

4. Hardware

4.1. FRAM Board

This is the MSP430FR2355 LaunchPad^{TM} Development Kit, which contains the MSP430FR2355 microcontroller along with a 40-pin plug-in module connector and a couple LEDs and switches. The MSP430 microcontroller is a 16-bit RISC architecture with 32KB of Program FRAM and 4KB of RAM. It is also connected to a MOSFET Onboard Debug Probe, which allows for easy connection to a computer for programming and debugging of the system. The schematic for what each pin of the MSP430 is being used for is shown below:

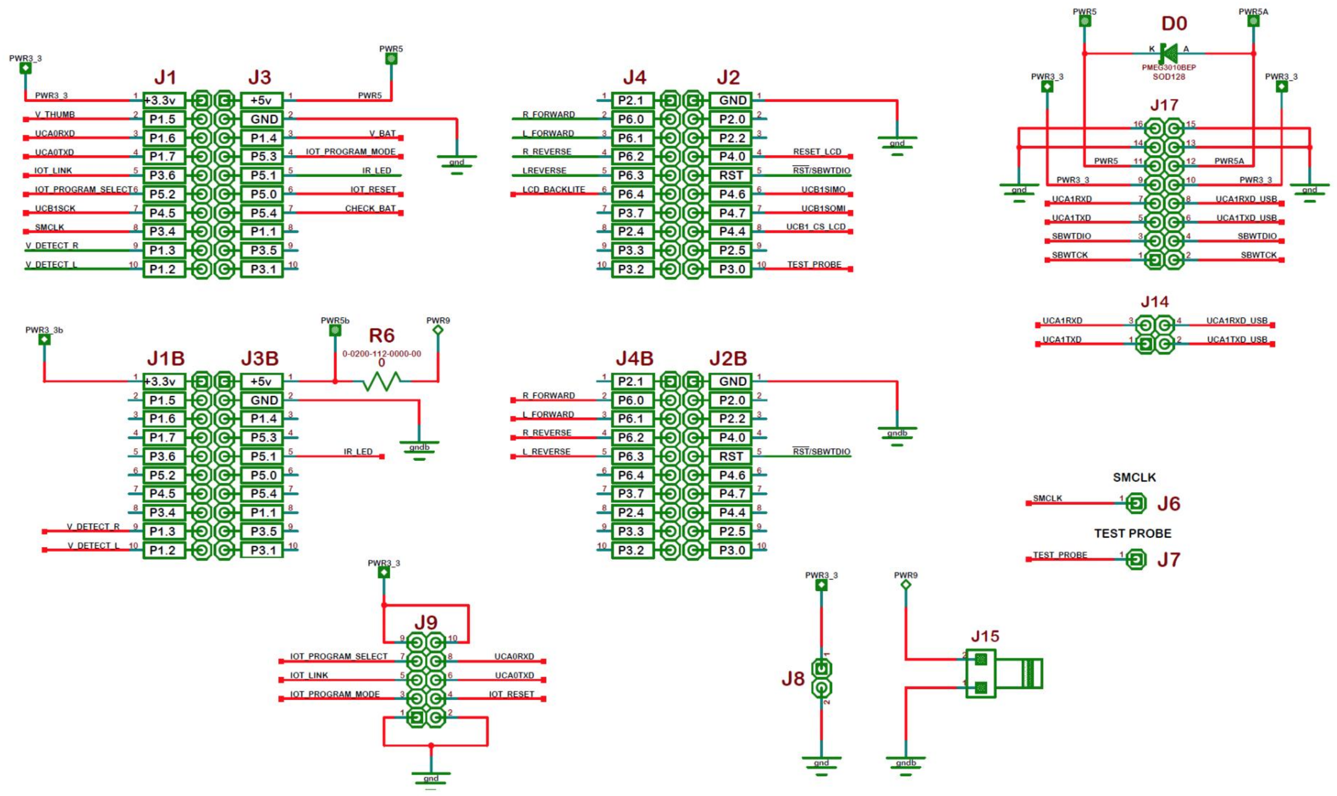


Figure 5 FRAM Board

4.2. Power Board

The power board was created to step down the input voltage to roughly 3.3 volts, and to distribute power to the devices, such as the MSP430 microcontroller and the LCD screen. The power board was created using the following circuit:

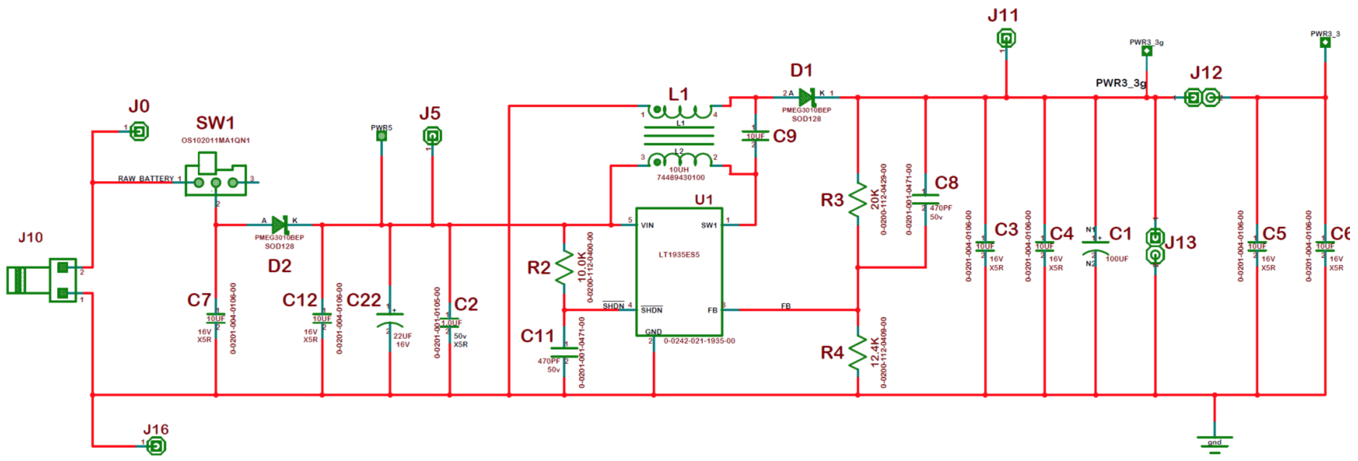


Figure 6 Power Board

4.3. LCD Display

The LCD Display is an EA DOGS104-A, and is soldered directly on top of the power board with 4 connections at the bottom of the board and 10 connections at the top of the board. The screen runs on 3.3V with a typical power consumption of 440mA. The schematic for the LCD is below:

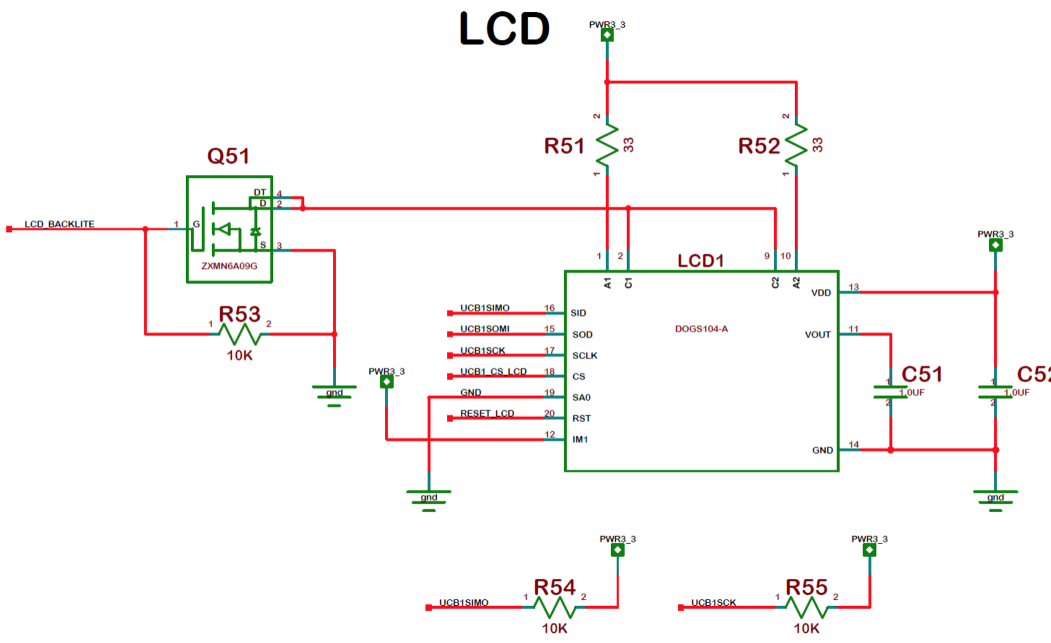


Figure 7 LCD Display

4.4. Control System

The control system is made up of three parts, the H-bridge, the motors and the wheels. The H-bridge currently has two N-FETs installed that control the motors, one for each motor. When the N-FET is on, current flows directly from the batteries, which causes the motors to turn. This in turn moves the wheels, which causes the car to run. The MOSFET is turned on and off rapidly to adjust the speed of the motors and wheels, allowing the car to turn. The schematic for what the control system will be when the car is completed is shown below:

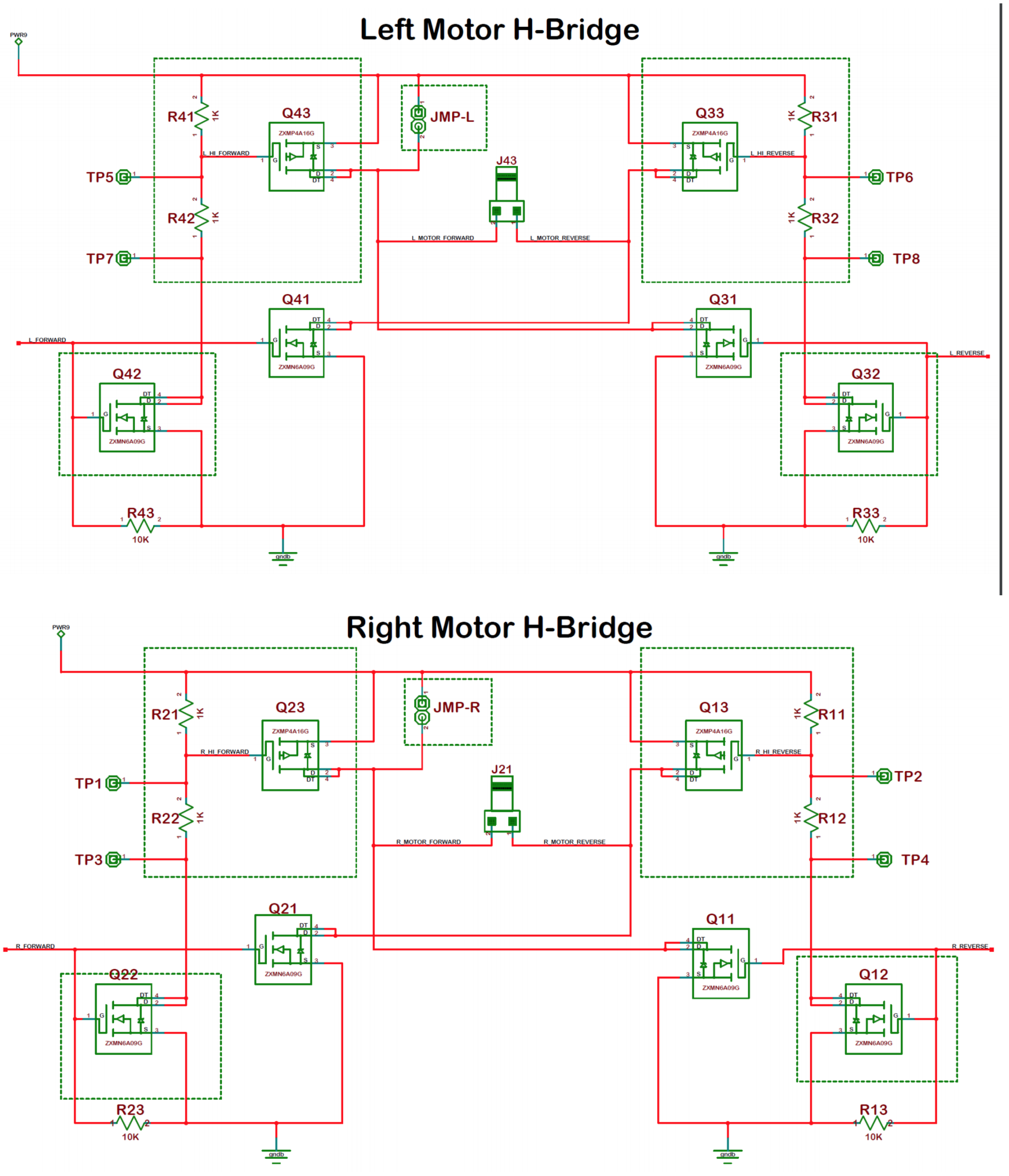


Figure 8 H-Bridge Circuits

Note, these two circuits appear on the same PCB board.

4.5. Chassis

The chassis is a flat block of material where all of the other hardware is mounted. The motors are mounted at the front, and the PCB boards are mounted in the middle. The battery pack is also mounted near the PCB boards. At the back of the chassis is some sort of object that keeps the car off of the ground that follows the front two wheels. Typically, this is either a caster wheel or a ping pong ball. Below is an image of the chassis:

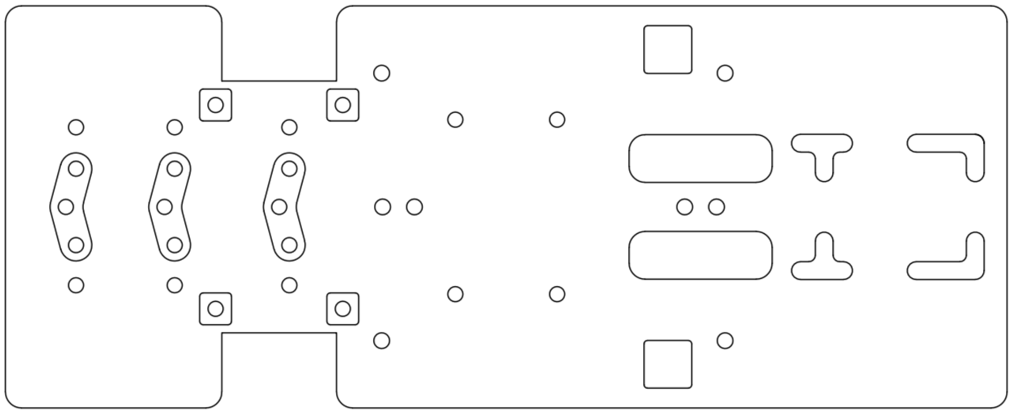
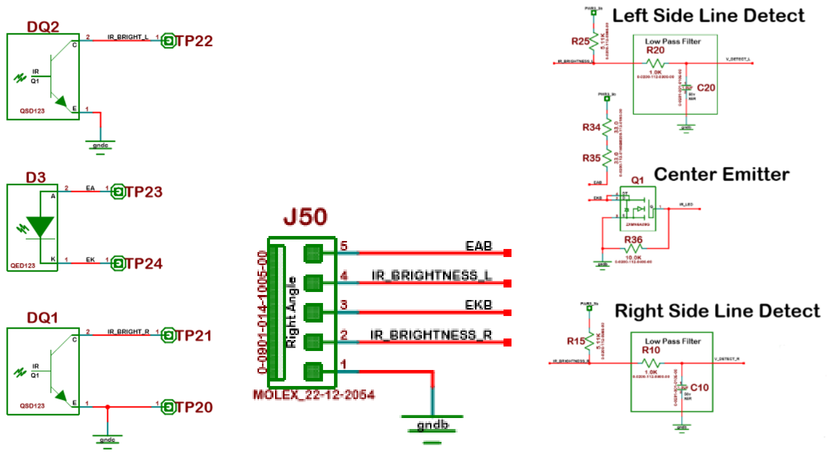


Figure 9 Vehicle Chassis

4.6 IR Board

The IR detectors consist of three parts: two detectors straddling an IR emitter. When the emitter is turned on, IR light strikes the floor underneath and reflects. The left and right detectors then measure how much light being reflected beneath them. When the detectors are over the color white, they reflect a low value; when the detectors are over the color black, they reflect a high value. The schematics for the left and right detectors and the IR emitter can be found below.



5. Conclusion

This project was outstanding, and taught me about concepts like operating systems, timers, interrupts, infrared sensors, LED's, buck boost converters, pulse width modulation, analog to digital converters, reading data sheets, creating testing procedures for hardware, and much more. There were times where the end scope of the project being more clear from the get-go would have helped considerably, like how I installed my batteries where my infrared sensors would later need to go, and had to rebuild my chassis from scratch, or when I would make a module that would do something like drive in a circle using a not flushed out motor function, that later proved to be useless for the rest of the project. But, overall I think I learned a lot about creating embedded systems, and managing my own projects. This is to the point where I'm able to pick up a few more projects relating to embedded systems that will come to fruition after I finish my next 2 personal projects.

6. Test Processes

The Following describes the testing process for the construction of the Power board, LCD, and motors of our Autonomous line following vehicle

6.1 Power Board Verification

Before beginning the process of installing the Control Board, verification of the power board must begin. The first step is to visually verify that every capacitor on your power board is connected correctly. A disconnected capacitor will show light, or darkness between the pad and the capacitor. If visual verification can be confirmed, you will be able to provide 5 volts of power, current limited to .1 Amp, to the center of the 5 ports between J0 and J5. You can then link the ground wire of your voltage source, and voltmeter to either of the two ports marked. If you're able to verify that there are no loose solder joints, and that every joint is correctly receiving power, you may move on to verification of the Control Board.

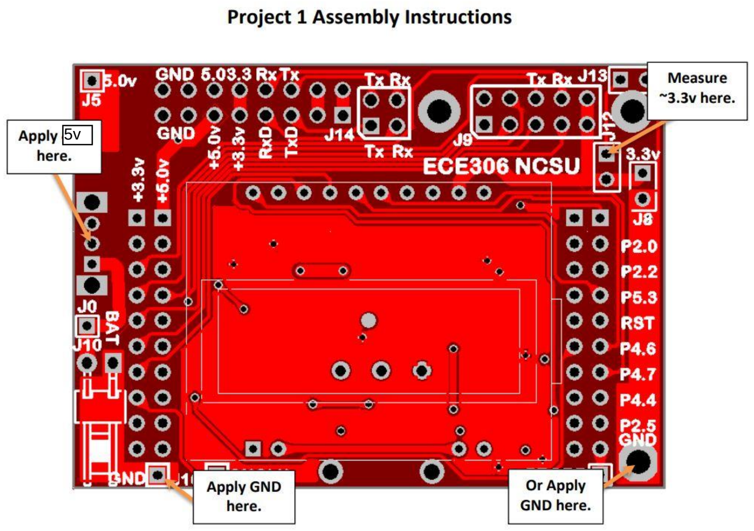


Figure 30 Power Board PCB

6.2 LCD Screen

During project 2, one must visually maintain that all soldering joints are correct, but supposing one does have visually correct soldering joints: after installing the switch, but before installing the jumper to j12, one can they can apply battery voltage, and measure the voltage at j0 being about 6.27 volts. They can then measure the voltage between j12/j13 and ground. This should measure about 3.42 volts. These two tests should show that power would be correctly sent to the LCD screen. Finally, after installing the LCD screen, being able to physically power it on can show your power system is currently set up correctly. Take note though: the board as it stands has a brownout protection circuit that means it may take one to even five power cycles before the LCD screen displays correctly. If you are having trouble, you can try powering the board through micro usb instead of through battery power. When all is said and done, the following should display on your board.

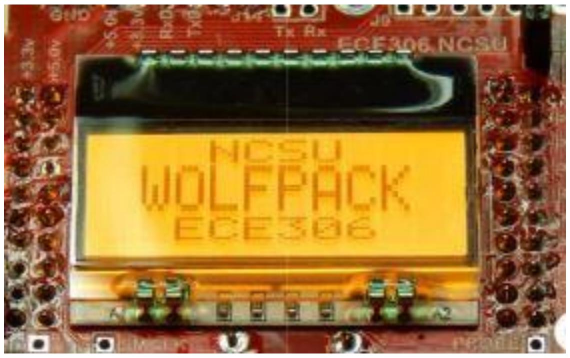


Figure 41 Active LCD Display

6.3 Forward Direction FET's

After installing the FET's required to implement forward motor control, you are encouraged to reflow the board regardless of how confident you are in your visual solder joints. You can then turn your motor on, and once again monitor the 3.3v source leading into the processor. Until you program a capacitor charge delay, you will need to use a technique dubbed "jumping your board", where you plug your board into a micro-usb port, turn on your battery, then unplug your usb port to successfully power your motors. If your motors successfully turn on, than you can download the forward-reverse test code, jump your board, and ensure your forward motors are operational. If they are operational, you should ensure your motors are successfully strain relieved as below. And, if you have strain relieved your motor, and are still not confident, you are encouraged to both glue and strain relieve your motor.

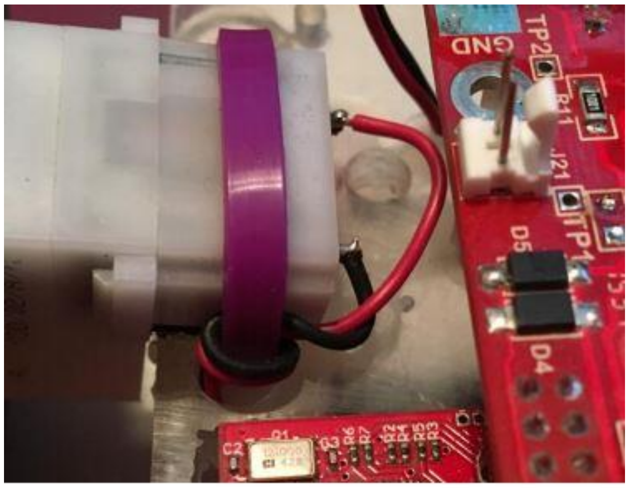


Figure 52 Motor Strain Relief

6.4 IR Emitter and Detectors

In order to verify working condition of your IR LED's, and IR Emitter, your encouraged to solder the wires to the board first, and touch a red LED to each of the ports that you will be soldering your IR LED's to, to ensure they are in working order. If they are, you will also want to press the IR LED to the pad, and check if it turns on with a non IR-blocking smart phone camera. If all of these spell out success, you can solder your IR LED's onto your LED board.

7. Software

The software portion of our entire system works with many interrelated files of code that each serve a purpose that allows our system to operate the way that we want it to. All of the files that are required to allow the system to work properly are put in the same directory. In this case, every variable that is created can be shared throughout each file in the directory as long as it is defined properly. For this project it was important to utilize files called main.c, ports.c, interrupt_ports.c, interrupt_timers.c, timers.c, serial.c and serial_display.c files. The names of the files don't essentially matter, but as long as each file has the properly assigned code, the system should work the way it needs to. Each interrupt works independently from each other and the same goes for the while loop in main, but the data and variables that are shared between the interrupts and main file are all affected by the code in the files. The functions in each of the files are also dependent on where the user calls them.

7.1. Main

In the main file, there is a while loop that is always being run while the system is operating and a list of initialization functions for the ports, clocks, conditions, timers, LCD, ADC, and Serial UCA0. In main, there should be code that functions when your system has officially received something through serial communication. This code should go through the received information and display that received information on your LCD. This information should also be ready to be transmitted back to where you received from.

7.2. Serial

The serial file will contain the initialization function for serial code as well as your serial interrupt. In the initial function for serial code, you will be setting your system's initial baud rate. You will also be setting up a loop based on your ring size. In the interrupt code, there is code that is set up to receive, and there is also code that is set up to transmit a set of at most 16 characters in your ring buffer. A flag should be set when your system has received something. This flag will trigger the code explained in main.

7.3. Timers

The interrupt timers file contains interrupts that are triggered by a set interval created by the user. Whatever is run through the interrupt is also written by the user. The timers file contains the initialization and configuration of the interrupt timers and also the initialization for the Pulse Width Modulation code that is used for controlling how fast the motors operate.

7.4. Ports

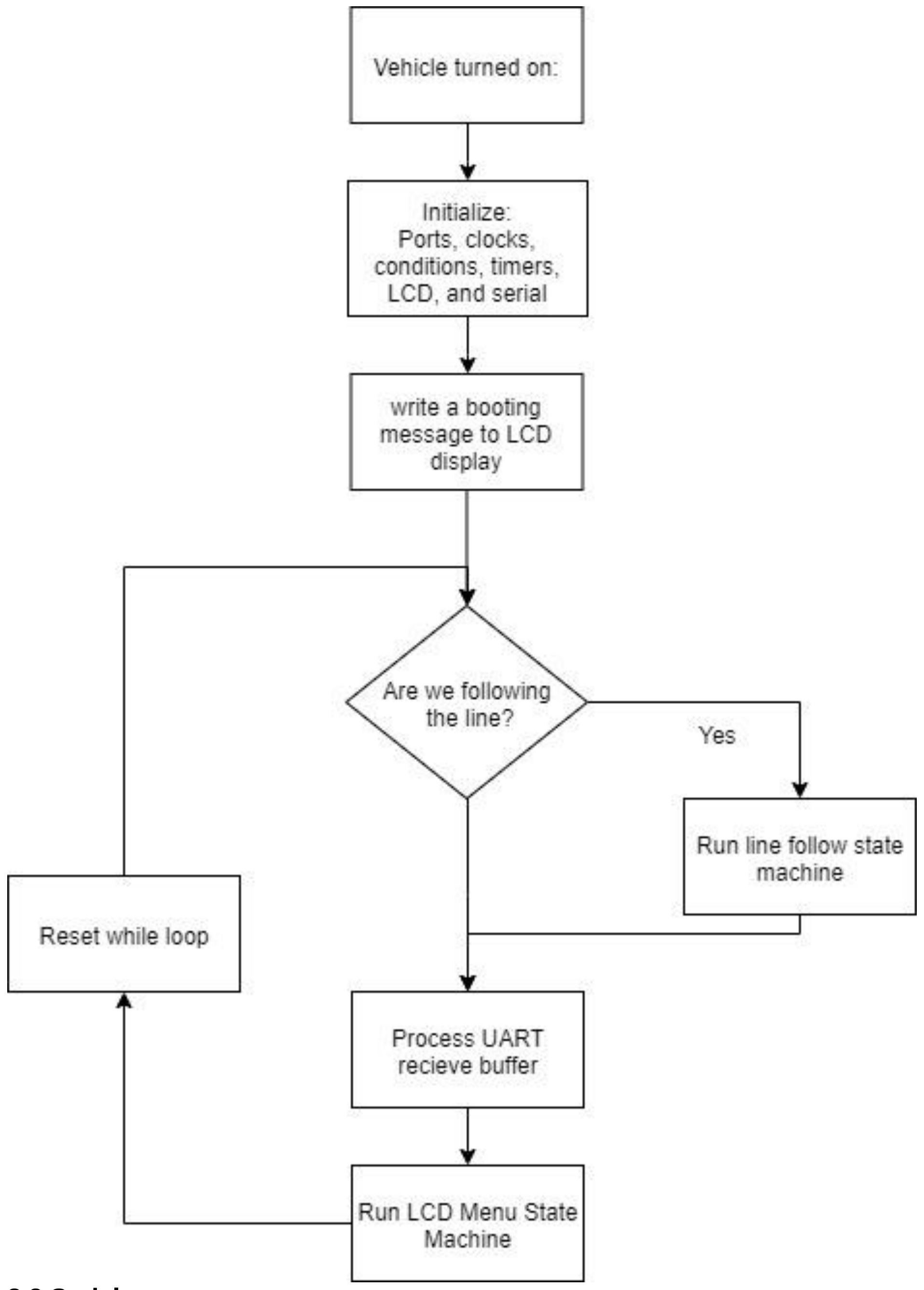
The ports file contains all the different port pin initializations. Each initialization can be set through this file to whatever function the user needs it to be for their system. For this project, ports are only needed for switch interrupt configurations. The rest can be set to their initial configurations. The interrupt ports file contains code that will cause an interrupt to be triggered whenever one of the FRAM board switches is pressed.

7.5. Display File

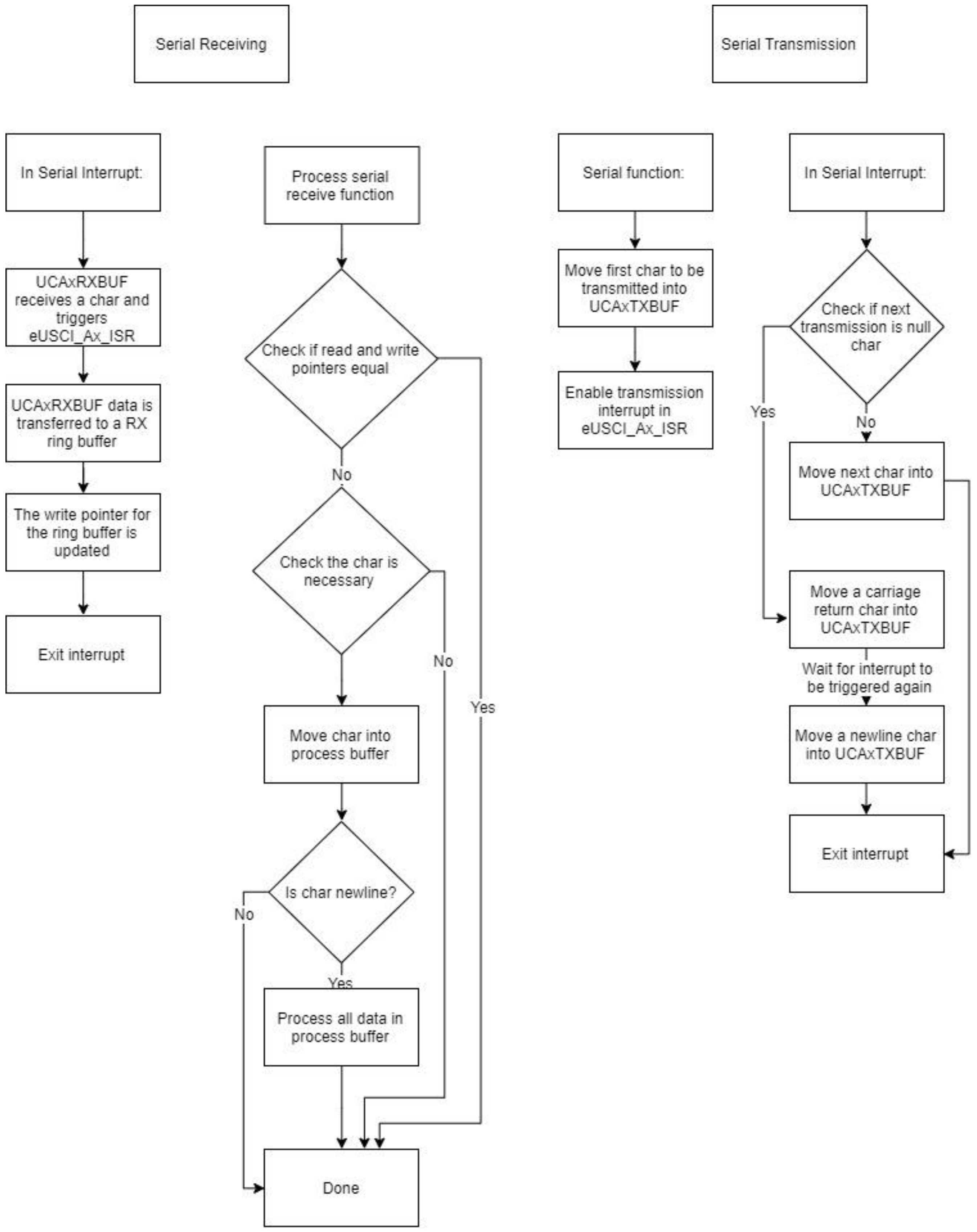
The serial display file will contain what your display will look like based on the different baud rates you set, and what you're transmitting or receiving.

8. Flow Charts

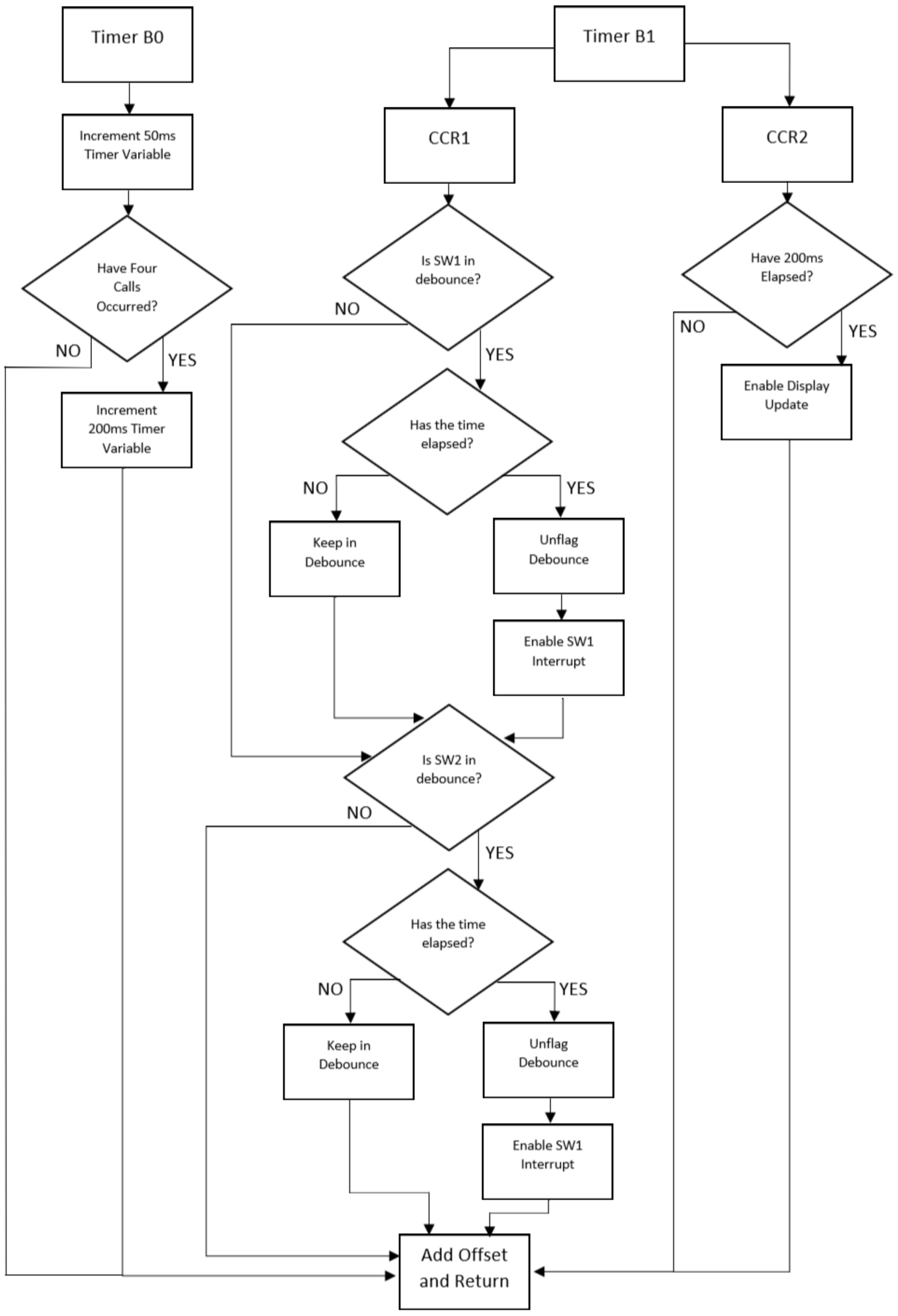
8.1. Main Blocks


Vehicle turned on:
Initialize: Ports, clocks, conditions, timers, LCD, and serial
write a booting message to LCD display
Are we following the line?
Yes
Run line follow state machine
Reset while loop
Process UART recieve buffer
Run LCD Menu State Machine

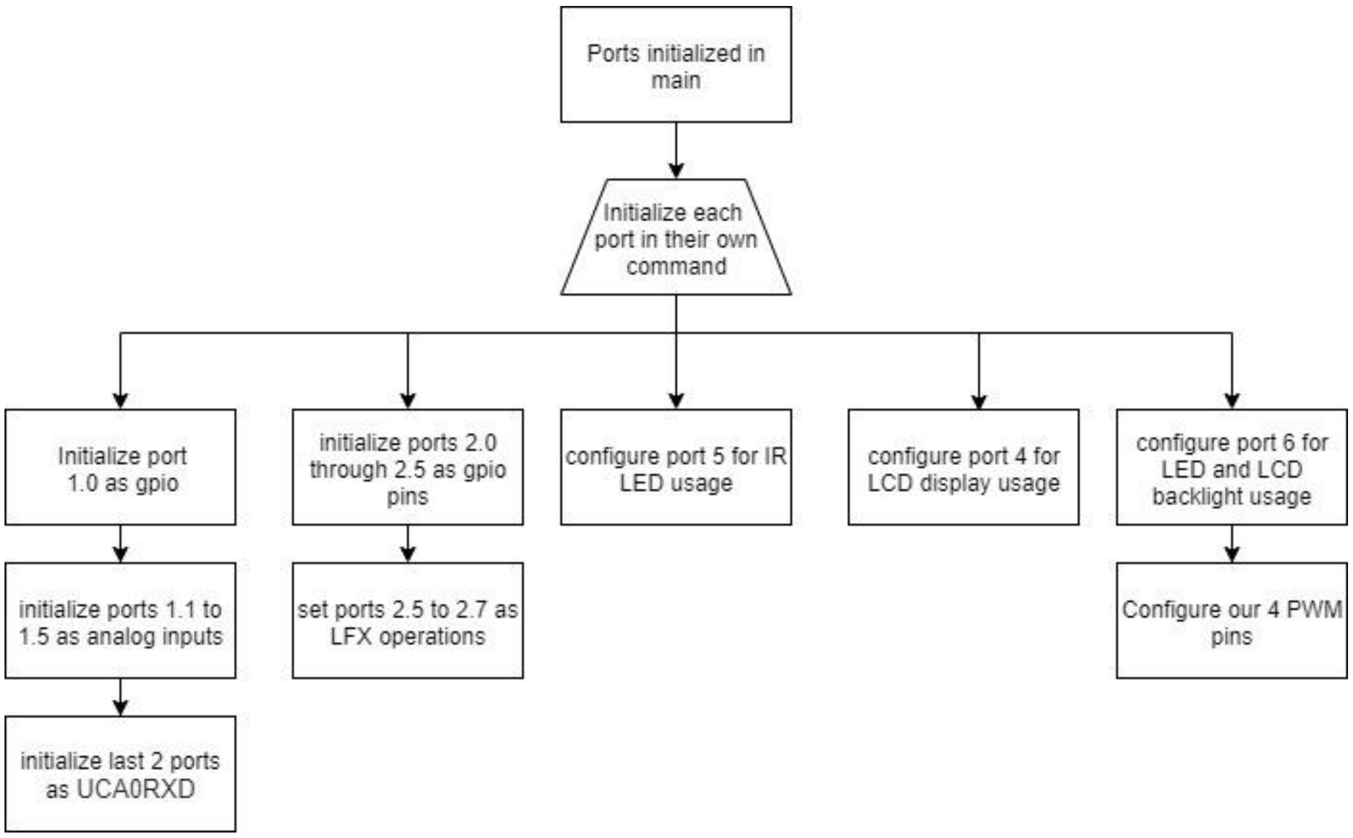
8.2 Serial


Serial Receiving
Serial Transmission
In Serial Interrupt:
UCAxRXBUF receives a char and triggers eUSCI_Ax_ISR
UCAxRXBUF data is transferred to a RX ring buffer
The write pointer for the ring buffer is updated
Exit interrupt
Process serial receive function
Check if read and write pointers equal
No
Yes
Check the char is necessary
No
Move char into process buffer
Is char newline?
No
Yes
Process all data in process buffer
Done
Serial function:
Move first char to be transmitted into UCAxTXBUF
Enable transmission interrupt in eUSCI_Ax_ISR
In Serial Interrupt:
Check if next transmission is null char
Yes
No
Move next char into UCAxTXBUF
Move a carriage return char into UCAxTXBUF
Wait for interrupt to be triggered again
Move a newline char into UCAxTXBUF
Exit interrupt

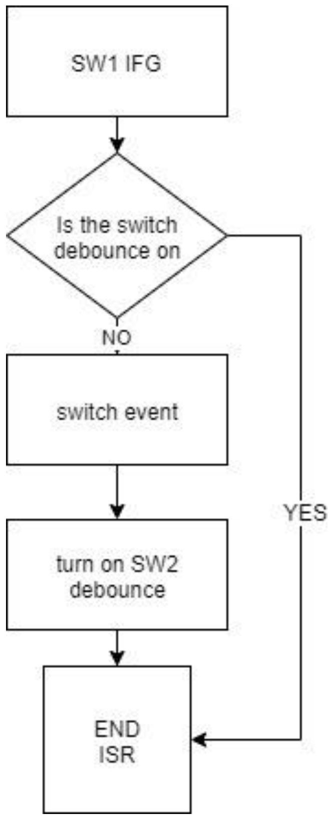
8.3 Timers


Timer B0
Increment 50ms Timer Variable
Have Four Calls Occurred?
NO
YES
Increment 200ms Timer Variable
Timer B1
CCR1
Is SW1 in debounce?
NO
YES
Has the time elapsed?
NO
YES
Keep in Debounce
Unflag Debounce
Enable SW1 Interrupt
Is SW2 in debounce?
NO
YES
Has the time elapsed?
NO
YES
Keep in Debounce
Unflag Debounce
Enable SW1 Interrupt
CCR2
Have 200ms Elapsed?
NO
YES
Enable Display Update
Add Offset and Return

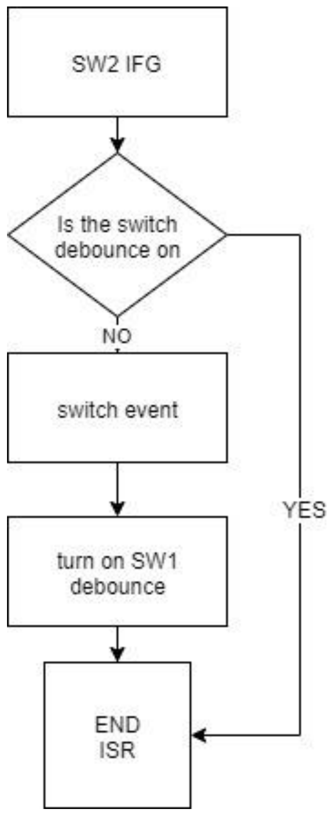
8.4 Ports


Ports initialized in main
Initialize each port in their own command
Initialize port 1.0 as gpio
initialize ports 1.1 to 1.5 as analog inputs
initialize last 2 ports as UCA0RXD
initialize ports 2.0 through 2.5 as gpio pins
set ports 2.5 to 2.7 as LFX operations
configure port 5 for IR LED usage
configure port 4 for LCD display usage
configure port 6 for LED and LCD backlight usage
Configure our 4 PWM pins

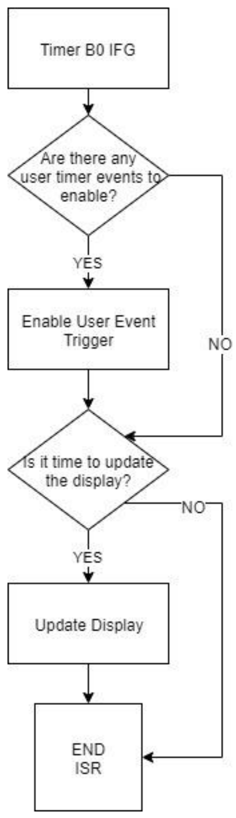
8.5 SW1 IFG


SW1 IFG
Is the switch debounce on
NO
YES
switch event
turn on SW2 debounce
END ISR

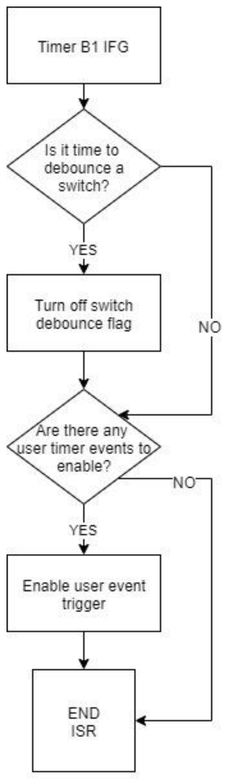
8.6 SW2 IFG



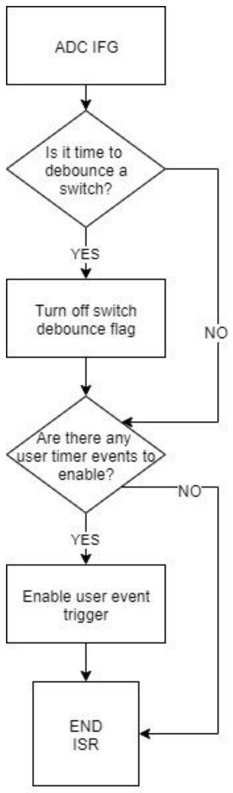
8.7 Timer B0 IFG


Timer B0 IFG
Are there any user timer events to enable?
YES
NO
Enable User Event Trigger
Is it time to update the display?
NO
YES
Update Display
END ISR

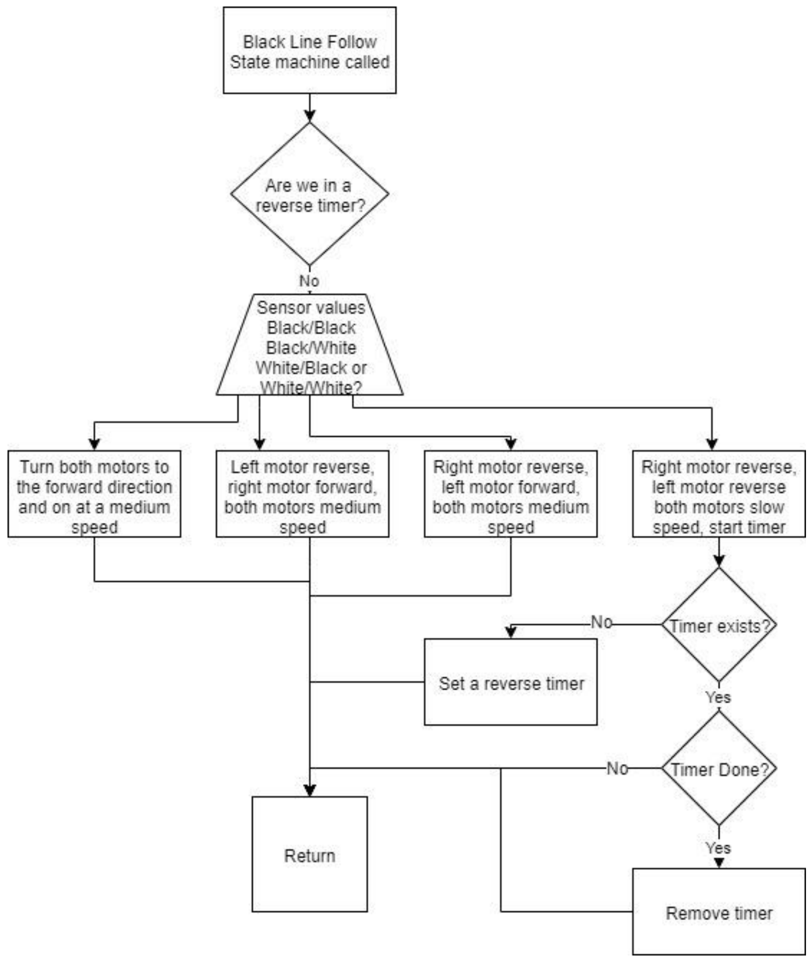
8.8 Timer B1 IFG


Timer B1 IFG
Is it time to debounce a switch?
YES
NO
Turn off switch debounce flag
Are there any user timer events to enable?
YES
NO
Enable user event trigger
END ISR

8.9 ADC IFG


ADC IFG
Is it time to debounce a switch?
YES
NO
Turn off switch debounce flag
Are there any user timer events to enable?
YES
NO
Enable user event trigger
END ISR

8.10 Follow Line


Black Line Follow State machine called
Are we in a reverse timer?
No
Sensor values Black/Black Black/White White/Black or White/White?
Turn both motors to the forward direction and on at a medium speed
Left motor reverse, right motor forward, both motors medium speed
Right motor reverse, left motor forward, both motors medium speed
Right motor reverse, left motor reverse both motors slow speed, start timer
Timer exists?
No
Set a reverse timer
Yes
Timer Done?
No
Yes
Return
Remove timer

9. Software Listing

9.1. Main

```
//------------------------------------------------------------------------------------------------
//
//  Description: This file contains the Main Routine - "While" Operating System
//
//
//  Benjamin Ryle
//  Mar 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------------------------

#include  "functions.h"              // lists function names for all pre-compiled files
#include  "msp430.h"                 // provides names for every port
#include  <string.h>                 // string functions from std library
#include  "macros.h"                 // global definitions file
#include  "motors.h"                 // motor definitions file

// Function Prototypes
void Menu(void);                     // State machine to process what goes in each lcd line
void LCD_Write(void);                // Refreshes LCD screen
void StateMachine(void);             // Autonomous Line following state machine
void uart_statemachine(void);       // turn uart commands passed from ISR into variable changes
void Init_Serial_UCA0(void);         // Sets up ports to run
void Init_Serial_UCA1(void);         // Sets up ports to run USB
void Init_Display(void);             // Sets up ports to run display
```

```
// Global Variables
unsigned int followtheline=0;       // don't run the line follow state machine by default

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.
  //
  //------------------------------------------------------------------------------------------------

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_UCA0();                    // Initialize Wifi module Serial
  Init_Serial_UCA1();                    // Initialize USB Serial
  Init_Display();                // Initialize LCD Menu abstraction variables

  //------------------------------------------------------------------------------------------------
  // Begining of the "While" Operating System
  //------------------------------------------------------------------------------------------------
  while(ALWAYS) {
    if (followtheline)           //if i'm currently following the line...
      StateMachine();                    //run the autonomous line following state machine.

    uart_statemachine();                 //look for and process any incoming or outgoing UART communications
    Menu();                              //refresh all menu aspects
    LCD_Write();                 //write to the LCD Screen.
  }

}
```

9.2. Serial

```
/////////////////////////////////////////////////////////////////////////////
//Description: Contains the uart state machine
//  Benjamin Ryle
//  Nov 2019
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//
/////////////////////////////////////////////////////////////////////////////

//include files
#include  "functions.h"             // lists function names for all pre-compiled files
#include  "msp430.h"                // provides names for every port
#include  <string.h>                // string functions from std library
#include  "macros.h"                // global definitions file
#include  "motors.h"                // motor definitions file
#include  "ports.h"                 // port definition translation file
#include  "StateMachine.h"          // definitions for the autonomous white line following state machine
// global variables

extern char LCD_LINE1 [LCD_LEN];            // String outputting to line 1 of the lcd
extern char LCD_LINE2 [LCD_LEN];            // String outputting to line 2 of the lcd
extern char LCD_LINE3 [LCD_LEN];            // String outputting to line 3 of the lcd
extern char LCD_LINE4 [LCD_LEN];            // String outputting to line 4 of the lcd

//usb ring buffer
extern volatile char USB_Char_Rx[SMALL_RING_SIZE] ;        // usb  receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr;               // usb  receive  ring buffer write position
extern unsigned int usb_rx_ring_rd;                        // usb  recieve  ring buffer read  position
extern volatile char USB_Char_Tx[LARGE_RING_SIZE] ;        // usb  transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr;                // usb  transmit ring buffer write position
extern unsigned int usb_tx_ring_rd;                         // usb  transmit ring buffer read  position

//wifi ring buffer
extern volatile char USB_Char_Rx1[SMALL_RING_SIZE] ;        // wifi receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr1;               // wifi recieve  ring buffer write position
extern unsigned int usb_rx_ring_rd1;                        // wifi recieve  ring buffer read  position
extern volatile char USB_Char_Tx1[LARGE_RING_SIZE] ;        // wifi transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr1;               // wifi transmit ring buffer write position
extern unsigned int usb_tx_ring_rd1;                        // wifi transmit ring buffer read  position

//function calls
void out_string(void);                                      // outputs the contents of the circular buffer to UCA0
void motor_set_directions(int left, int right);             // safely sets the direction of both motors
void motors_on(int speed);                                  // turns on the motors when it's next safe to do so
void right_motor_on(int speed);                             // turns on the right motor when it's safe to do so
void left_motor_on(int speed);                              // turns on the left  motor when it's safe to do so
void motors_off(void);                                      // turns off all motors

extern unsigned int disp_time;        // count of how many seconds have passed to print a timer to the lcd display
extern unsigned int state;           // state of autonomous line following state machine
extern unsigned int battery_level;    // pwm value of line following state machine
extern volatile int cmd_state;        // whether there is an IOT or rc car command to process or not
extern unsigned int followtheline;    // whether we are following the line or not


char command_string [lcd_col] = "     "; // stores commmands respective string to be printed by menu state machine
char comand [comand_len];               // stores commmands recieved from wifi module
int Motor_timer=timer_disabled;         // auto motor cutoff on timer
volatile unsigned int pos_count=0        // what part of the obstacale course the car is at.

void uart_statemachine(void){
  if (cmd_state==process_iot_command){
    lcd_4line();
    while (usb_rx_ring_wr1!=usb_rx_ring_rd1){
      char temp=USB_Char_Rx1[usb_rx_ring_rd1];
      USB_Char_Tx[usb_tx_ring_wr]=temp;
      usb_rx_ring_rd1++;
      usb_tx_ring_wr++;
      if (usb_tx_ring_wr>LARGE_RING_SIZE)
        usb_tx_ring_wr=BEGINNING;
      if (usb_rx_ring_rd1>SMALL_RING_SIZE)
        usb_tx_ring_rd1=BEGINNING;
      cmd_state=hold;
    }
    out_string();
  }
  //void motor_set_directions(int left, int right);
  //void motors_on(int speed);
  //void motors_off(void);
  extern int lcd_state;
  if (cmd_state==process_fram_command){
    lcd_state=show_command;
    lcd_BIG_mid();
    //ensure they have the right key
    cmd_state=hold;

    switch (comand[parse_direction]){
    case 'L':   //turn left
      Motor_timer= time_constant*(comand[parse_time]-'0');
      motor_set_directions(reverse, forward);
      motors_on(med);

      command_string[2]='L';
      command_string[3]='E';
      command_string[4]='F';
      command_string[5]='T';
      command_string[6]=' ';
      command_string[7]=' ';
      command_string[8]=' ';
      right_motor_on(med);
      left_motor_on(med);
      break;
    case 'R':          // turn right
      Motor_timer=time_constant*(comand[parse_time]-'0');
      motor_set_directions(forward, reverse);
      motors_on(med);

      command_string[2]='R';
      command_string[3]='I';
      command_string[4]='G';
      command_string[5]='H';
      command_string[6]='T';
      command_string[7]=' ';
      command_string[8]=' ';
      right_motor_on(med);
      left_motor_on(med);
      break;
    case 'B':   //go backwards
      Motor_timer= time_constant*(comand[parse_time]-'0');
      motor_set_directions(reverse, reverse);
      motors_on(med);

      command_string[2]='B';
      command_string[3]='A';
      command_string[4]='C';
      command_string[5]='K';
      command_string[6]=' ';
      command_string[7]=' ';
      command_string[8]=' ';
      right_motor_on(fast);
      left_motor_on(med);

      break;
    case 'F': //go forwards
      Motor_timer= time_constant*(comand[parse_time]-'0');
      motor_set_directions(forward, forward);
      motors_on(fast);

      command_string[2]='F';
      command_string[3]='O';
      command_string[4]='R';
      command_string[5]='W';
      command_string[6]='A';
      command_string[7]='R';
      command_string[8]='D';
      motors_on(fast);
      break;
    case 'H': //HALT
      motors_off();
      Motor_timer= -1;
      command_string[2]='W';
      command_string[3]='A';
      command_string[4]='I';
      command_string[5]='T';
      command_string[6]='I';
      command_string[7]='N';
      command_string[8]='G';
```

```
          break;
        case 'S':   // find line angle #1
          lcd_state=BL_lcd;
          followtheline=TRUE;
          battery_level=1;
          state=pause;
          break;
        case 'T':// find line angle #2
          lcd_state=BL_lcd;
          followtheline=TRUE;
          battery_level=2;
          state=pause;
          break;
        case 'U'://find line angle #3
          lcd_state=BL_lcd;
          followtheline=TRUE;
          battery_level=3;
          state=pause;
          break;
        case 'V'://find line angle #4
          lcd_state=BL_lcd;
          followtheline=TRUE;
          battery_level=4;
          state=pause;
          break;
        case 'W'://find line angle #5
          lcd_state=BL_lcd;
          followtheline=TRUE;
          battery_level=5;
          state=pause;
          break;
        case 'X':   //Xit (exit) the circle
          lcd_state=BL_lcd;
          state=turn_right;
          break;

        case '0': // reset timer on display
          disp_time=FALSE;
          break;

        case 'A': // increments counter showing what part of the obstacle course we're on
          pos_count++;
          break;

        case 'Y': // starts leaving the circle
          state =turn_right;
          break;
        }
      }
}
```

9.3. ADC & Interrupts

```
//------------------------------------------------------------------------------
//
//  Description: This file contains every ADC interrupt programmed so far
//
//  Benjamin Ryle
//  Sept 11
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------
#include "StateMachine.h"       // definitions for the autonomous white line following state machine
#include  "functions.h"         // lists function names for all pre-compiled files
#include  "msp430.h"            // provides names for every port
#include <string.h>             // string functions from std library
```

```
#include "macros.h"             // global definitions file
#include "motors.h"             // motor definitions file
#include "timers.h"             // timer definitions file
#include "ports.h"              // port definition translation file

void motors_off(void);          //ability to cut off motors for safety reasons

int Left_Color;                                 // whether the left IR detector sees black or white
int Right_Color;                                // whether the right IR detector sees black or white

extern unsigned int followtheline;              // whether we are following the black line
extern volatile int motor_next_direction_L;     // what the next direction for the left motor will be
extern volatile int motor_direction_L;          // what the current direction for the left motor is.
extern volatile int motor_direction_R;          // what the current direction for the right motor is.
extern volatile int motor_next_direction_R;     // what the next direction for the right motor will be.
volatile unsigned int ADC_Thumb;                // adc value of thumb wheel

int ADC_Channel;                                // next device to be judged by analog to digital converter
unsigned int ADC_Left_Detect;                   // Left Infared Detector ADC value
unsigned int ADC_Right_Detect;                  // Right Infared Detector ADC value

extern int IOT_COUNTDOWN;                       // Countdown before starting next IOT process
extern char LCD_LINE1 [LCD_LEN];                // String outputting to line 1 of the lcd
extern char LCD_LINE2 [LCD_LEN];                // String outputting to line 2 of the lcd
extern char LCD_LINE3 [LCD_LEN];                // String outputting to line 3 of the lcd
extern char LCD_LINE4 [LCD_LEN];                // String outputting to line 4 of the lcd


extern volatile unsigned char display_changed;  // whether the LCD should update
unsigned int Black_Detect=160;                          // ADC threshold (larger is black. smaller is white.)


//usb ring buffer
extern volatile char USB_Char_Rx[SMALL_RING_SIZE] ;// usb  receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr;                    // usb  receive  ring buffer write position
extern unsigned int usb_rx_ring_rd;                     // usb  recieve  ring buffer read  position
extern volatile char USB_Char_Tx[LARGE_RING_SIZE] ;             // usb  transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr;                    // usb  transmit ring buffer write position
extern unsigned int usb_tx_ring_rd;                     // usb  transmit ring buffer read  position

//wifi ring buffer
extern char USB_Char_Rx1[SMALL_RING_SIZE] ;                     // wifi receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr1;                   // wifi recieve  ring buffer write position
extern unsigned int usb_rx_ring_rd1;                    // wifi recieve  ring buffer read  position
extern volatile char USB_Char_Tx1[LARGE_RING_SIZE] ;            // wifi transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr1;                   // wifi transmit ring buffer write position
extern unsigned int usb_tx_ring_rd1;                    // wifi transmit ring buffer read  position

extern char ip_line1 [LCD_LEN]; // what will be printed to the LCD when the Menu state machine wants to print the ip
address.
extern char ip_line2   [LCD_LEN];           // what will be printed to the LCD when the Menu state machine wants to print
the ip address.
extern char command_string [11];            // what command has been recieved from the IOT module via TCP


#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void){
  switch(__even_in_range(ADCIV,ADCIV_ADCIFG)){
  case ADCIV_NONE:
    break;
  case ADCIV_ADCOVIFG: // When a conversion result is written to the ADCMEM0
    // before its previous conversion result was read.
    break;
  case ADCIV_ADCTOVIFG: // ADC conversion-time overflow
```

```
      break;
    case ADCIV_ADCHIIFG: // Window comparator interrupt flags
      break;
    case ADCIV_ADCLOIFG: // Window comparator interrupt flag
      break;
    case ADCIV_ADCINIFG: // Window comparator interrupt flag
      break;
    case ADCIV_ADCIFG: // ADCMEM0 memory register with the conversion result
      ADCCTL0 &= ~ADCENC; // Disable ENC bit.
      switch (ADC_Channel++){
      case left_detect: // Channel A2 Interrupt

        ADC_Left_Detect = ADCMEM0; // Move result into Global
        ADC_Left_Detect = ADC_Left_Detect >> 4; // Divide the result by 8
        if (ADC_Left_Detect-60<Black_Detect)
          Left_Color=WHITE;
        else Left_Color=BLACK;
        ADCMCTL0 &= ~ADCINCH_2; // Disable Last channel A2
        ADCMCTL0 |= ADCINCH_3; // Enable Next channel A3
        break;

      case right_detect:

        ADC_Right_Detect = ADCMEM0; // Move result into Global
        ADC_Right_Detect = ADC_Right_Detect >> 4; // Divide the result by 8
        if (ADC_Right_Detect<Black_Detect)
          Right_Color=WHITE;
        else Right_Color=BLACK;
        ADCMCTL0 &= ~ADCINCH_3; // Disable Last channel A3
        ADCMCTL0 |= ADCINCH_5; // Enable Next channel A4
        break;

      case thumb_wheel:
        ADCMCTL0 &= ~ADCINCH_5; // Disable Last channel A4
        ADCMCTL0 |= ADCINCH_2;
        ADC_Thumb = ADCMEM0;
        ADC_Channel=rst_time;
        ADCCTL0 &= ~ADCENC;
        //     IR_Timer=LED_Refresh_Time;

        break;
      default:
        break;
      }
      ADCCTL0 |= ADCENC; // Enable Conversions
      ADCCTL0 |= ADCSC; // Start next sample
      break;
    default:
      break;
    }
  }
```

9.4. Timers

```
//------------------------------------------------------------------------------
//
//  Description: This file contains every Timer interrupt programmed
//
//  Benjamin Ryle
//  Sept 11
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------
#include "StateMachine.h"             // definitions for the autonomous white line following state machine
#include  "functions.h"               // lists function names for all pre-compiled files
#include  "msp430.h"                  // provides names for every port
```

```
#include <string.h>                   // string functions from std library
#include "macros.h"                   // global definitions file
#include "motors.h"                   // motor definitions file
#include "timers.h"           // timer definitions file
#include "ports.h"            // port definition translation file


int startup=TRUE;             //whether the module is first starting up, used to start IOT setup state machine
extern volatile int motor_next_direction_L;     //next direction for the left motor (forward/reverse)
extern volatile int motor_direction_L;          //current motor direction for the left motor
extern volatile int  motor_direction_R;         //current motor direction for the right motor
extern volatile int motor_next_direction_R;    //next direction for the right motor (forward/reverse)
extern int MotorTimer; // a countdown to when motors will be able to work again after changing directions

volatile unsigned int Time_Sequence;   // a count to activate certain activities after N number of times, or 1 in every n
numbers
unsigned int disp_time;                 // count of how many seconds have passed to print a timer to the lcd display

char display_timer;           // count of how many seconds have passed to print a timer to the lcd display
extern int IOT_COUNTDOWN;             // timer based countdown for IOT state machine
extern int sw1_debounce;              // disables the left switch so 1 button press isn't registered as 2 or more
extern int sw2_debounce;              //disables the right switch so 1 button press isn't registered as 2 or more
extern int Motor_timer;               // countdown until motors are automatically turned off without input

extern volatile unsigned char display_changed;
volatile int State_Timer;             // Counts down on a timer and then sets State_Trigger to true
char State_Trigger;           // nonblocking delay in line following statemacahine
volatile int State_Timer2;            // Counts down on a timer and then sets State_Trigger2 to true
char State_Trigger2;                  // noblocking delay in line following statemachine
void enable_display_update(void);   // enables the lcd screen to be updated
void out_string(void);                // outputs a string to the IOT module

char diss_int []= "AT+WSYNCINTRL=2000";   // changes dissociation timer so the wifi doesn't disconnect
char start_tcp [] ="AT+NSTCP=11100,1";        // opens TCP port 11100
//usb ring buffer
extern volatile char USB_Char_Rx[SMALL_RING_SIZE] ;        // usb  receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr;                 // usb  receive  ring buffer write position
extern unsigned int usb_rx_ring_rd;                                   // usb  recieve  ring buffer read  position
extern volatile char USB_Char_Tx[LARGE_RING_SIZE] ;        // usb  transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr;                 // usb  transmit ring buffer write position
extern unsigned int usb_tx_ring_rd;                                   // usb  transmit ring buffer read  position

//wifi ring buffer
extern char USB_Char_Rx1[SMALL_RING_SIZE] ;     // wifi receive  ring buffer array
extern volatile unsigned int usb_rx_ring_wr1;           // wifi recieve  ring buffer write position
extern unsigned int usb_rx_ring_rd1;                    // wifi recieve  ring buffer read  position
extern volatile char USB_Char_Tx1[LARGE_RING_SIZE] ;   // wifi transmit ring buffer array
extern volatile unsigned int usb_tx_ring_wr1;           // wifi transmit ring buffer write position
extern unsigned int usb_tx_ring_rd1;                    // wifi transmit ring buffer read  position
extern char command_string [11];                        //  string  reflecting  the  current  IOT  command  output  to  LCD
screen
extern char ip_line1   [LCD_LEN];                  // line 1 of string storing ip address to print to lcd
extern char ip_line2   [LCD_LEN];                  // line 2 of string storing ip address to print to lcd
//char quick_array [128];
int gather_ip=TRUE;                             //whether the ip address is being gathered to output to the LCD screen
void motors_off(void);                          //turns off motors
//
#pragma vector = TIMER0_B0_VECTOR
__interrupt void Timer0_B0_ISR(void){//every 1.25 ms
  TB0CCR0 +=TB0CCR0_INTERVAL;                       //adds offset to TBCCR0
  display_timer++;
  if (display_timer==display_update_time){          //if it's time to update the display
    enable_display_update();
    Display_Update(LCD_INST,LCD_MID,LCD_TOP,LCD_TOP);   //run the display update function
    display_timer=rst_time;
    display_changed=rst_time;                              //disable display changed flag
  }
  if (MotorTimer==rst_time){                          //if the motors have been off long enough to deplete any built up charge
    motor_direction_R=motor_next_direction_R;           //set the motor direction from disabled to the next direction
    motor_direction_L=motor_next_direction_L;
    MotorTimer--;                                       //disable the countdown
  }
  else if (MotorTimer>rst_time)
    MotorTimer--;
  if (State_Timer>rst_time)
    State_Timer--;
  else if (State_Timer==rst_time){
    State_Trigger=TRUE;
    State_Timer--;
  }
  if (State_Timer2>rst_time)
    State_Timer2--;
  else if (State_Timer2==rst_time){
    State_Trigger2=TRUE;
    State_Timer2--;
  }
}


//this is the interrupt responsible for re-enabling the buttons about 500ms after it's pressed.
#pragma vector=TIMER0_B1_VECTOR
__interrupt void TIMER0_B1_ISR(void){
  //timerB0 1-2, Overflow interrupt vector (TBV1 handler)

  switch(__even_in_range(TB0IV,overflow)){
  case disabled: break;
  case ccr1: //timer debounce every 62.5 ms
    if (Time_Sequence%4==BEGINNING & ADC_ENABLE){
      ADCCTL0 |= ADCENC;
    }
    Time_Sequence++;
    if (sw1_debounce>disabled) //we are counting our disabled timer down to 0
      sw1_debounce--;
    if (sw2_debounce>disabled)//at 0, we can press our button again in it's respective interrupt
      sw2_debounce--;

    TB0CCR2 +=TB0CCR1_INTERVAL;
    break;
  case ccr2:       //ccr2 every 25 ms
    TB0CCR2 +=TB0CCR2_INTERVAL;
    disp_time+=disp_increment;
    if (Motor_timer>BEGINNING)
      Motor_timer--;
    else if (Motor_timer==BEGINNING){                         //if the motors are turned off
      motors_off();
      Motor_timer--;
      command_string[2]='S';
      command_string[3]='T';                               //set the LCD display to say STOP
      command_string[4]='O';                               //(depending on the menu state machine)
      command_string[5]='P';
      command_string[6]=' ';
      command_string[7]=' ';
      command_string[8]=' ';
    }
    if(IOT_COUNTDOWN>BEGINNING)
      IOT_COUNTDOWN--;
    else
    switch (startup){                                             //state machine for IOT startup (wifi startup)
```

```
        case expire:
          for (int i=BEGINNING; i<(sizeof(diss_int)+1); i++){ // set dissociation timer high enough to not disconnect from the
network
            USB_Char_Tx[usb_tx_ring_wr]=diss_int[i];
            usb_tx_ring_wr++;
            if (usb_tx_ring_wr>sizeof(USB_Char_Tx))
              usb_tx_ring_wr=BEGINNING;
          }
          //startup=3;
          startup++;
          IOT_COUNTDOWN=expire_delay;
          USB_Char_Tx[usb_tx_ring_wr-1]= '\r'; // newline tells the IOT module to run the previous command
          out_string();
        break;

        case turnon:
          P5OUT |= IOT_RESET;
          startup++;
          IOT_COUNTDOWN=turnon_delay;                                    //delay to display on screen first
        break;

        case tcp_setting:
          startup=FALSE;
          for (int i=BEGINNING; i<sizeof(start_tcp); i++){          // tell IOT module to set up a TCP server
            USB_Char_Tx[usb_tx_ring_wr]=start_tcp[i];
            usb_tx_ring_wr++;
            if (usb_tx_ring_wr>sizeof(USB_Char_Tx))
              usb_tx_ring_wr=BEGINNING;
          }
          IOT_COUNTDOWN=tcp_setting_delay;//EVERY 5 SECONDS
          USB_Char_Tx[usb_tx_ring_wr-1]= '\r';
          out_string();
           ip_line2[8]='O';                                // print OK to the display to show the state machine is done
           ip_line2[9]='K';
        break;

        }
        break;
      case  overflow: //overflow
        break;
      default: break;
      }
}
```

9.5. Ports

```
//------------------------------------------------------------------------------
//
//  Description: This file contains a chart of port definitions and
//                    all the initialization functions for the msp430's ports.
//
//  Benjamin Ryle
//  Sept 11
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------


#include  "functions.h"           // lists function names for all pre-compiled files
#include  "msp430.h"              // provides names for every port
```

```
#include  <string.h>              // string functions from std library
#include  "macros.h"               // global definitions file
#include "ports.h"                 // translation of port bit numbers to port names
int IOT_COUNTDOWN=100;             // countdown until an iot process is run
//-----------------------------------------------------------------------------------------
//all values are instantiated to the safest position
//in this case, it's low output gpio
//all port values set to 0
//all ports set to outputs
//ONLY PORTS WITH DEFINED VALUES NEED TO BE WRITTEN HERE
//a value will only be instantiated if a change from the constructor is needed.
//-----------------------------------------------------------------------------------------

//PORT 1
//left as GPIO output 0
void Init_Port_1(void){
  P1OUT=GPIO_all;
  P1SEL0=GPIO_all;
  P1SEL1=GPIO_all;
  P1DIR=output_all;



  //P1 PIN 0
  //P1 PIN 1
  P1SEL0 |=  A1_SEEED;
  P1SEL1 |=  A1_SEEED;
  //P1 PIN 2
  P1SEL0 |=  V_DETECT_L;
  P1SEL1 |=  V_DETECT_L;
    P1REN  &=  ~V_DETECT_L;
  //P1 PIN 3

  P1SEL0 |=  V_DETECT_R;
  P1SEL1 |=  V_DETECT_R;
  P1REN  &=  ~V_DETECT_R;
  //P1 PIN 4
  P1SEL0 |=  V_BAT;
  P1SEL1 |=  V_BAT;
  //P1 PIN 5
  P1SEL0 |=  V_THUMB;
  P1SEL1 |=  V_THUMB;
  //P1 PIN 6
  P1SEL0 |=  UCA0RXD;
  P1SEL1 &= ~UCA0RXD;
  //P1 PIN 7
  P1SEL0 |=  UCA0TXD;
  P1SEL1 &= ~UCA0TXD;
}

//PORT 2
//left as GPIO output 0
void Init_Port_2(void){
  P2OUT=GPIO_all;
  P2SEL0=GPIO_all;
  P2SEL1=GPIO_all;
  P2DIR=output_all;



  //P2 PIN 0
  //P2 PIN 1
  //P2 PIN 2
  //P2 PIN 3
  P2SEL0 &= ~SW2;
  P2SEL1 &= ~SW2;
  P2DIR &= ~SW2;
  P2OUT |=  SW2;
  P2REN |=  SW2;
  P2IES |= SW2;
  P2IFG &= ~SW2;
  P2IE |= SW2;
  //P2 PIN 4
  //P2 PIN 5
  //P2 PIN 6
  P2SEL0 &= ~LFXOUT;                    // Primary (MCLK)
  P2SEL1 |= LFXOUT;                     // Primary (MCLK)
  //P2 PIN 7
  P2SEL0 &= ~LFXIN;                     // Primary (TB0CLK)
  P2SEL1 |= LFXIN;                      // Primary (TB0CLK)
}

//PORT 3
//left as GPIO output 0
```

```
void Init_Port_3(int use_smclk){
  P3OUT=GPIO_all;
  P3SEL0=GPIO_all;
  P3SEL1=GPIO_all;
  P3DIR=output_all;

  //P3 PIN 0
  //P3 PIN 1
  //P3 PIN 2
  //P3 PIN 3
  if (use_smclk)
    P3SEL0 |=SMCLK;
  //P3 PIN 4
  //P3 PIN 5
  //P3 PIN 6
  P3OUT &= ~IOT_LINK;
  //P3 PIN 7

}

//PORT 4
//sets up the lcd screen to use ucb1 as a spi interface instead of gpio
void Init_Port_4(void){
  P4OUT=GPIO_all;
  P4SEL0=GPIO_all;
  P4SEL1=GPIO_all;
  P4DIR=output_all;

  //P4 PIN 0
  //LCD_RESET GETS RESET CORRECTLY AT TOP
  P4SEL0 &= ~RESET_LCD;
  P4SEL1 &= ~RESET_LCD;
  P4OUT  &= ~RESET_LCD;
  P4DIR  |=  RESET_LCD;

  //P4 PIN 1
  P4SEL0 &= ~SW1; //sw1 set as io
  P4SEL1 &= ~SW1;
  P4DIR &= ~SW1; //SW1 direction input
  P4OUT |=  SW1; // SW1 PULL UP RESISTOR CONFIG
  P4REN |=  SW1;  //SW1 PULL UP RESISTOR ENABLE
  P4IES |= SW1; //HI/LO EDGE INTERRUPT
  P4IFG &= ~SW1; //SW1 CLEARED
  P4IE |= SW1; //INTERRUPT ENABLED

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

  //P4 PIN 3
  P4SEL1 &= ~UCA1RXD;
  P4SEL0 |=  UCA1RXD;

  //P4 PIN 4
  P4SEL1 &= ~UCB1_CS_LCD;
  P4SEL0 &= ~UCB1_CS_LCD;// sets SPI_CS_LCD off (high)
  P4OUT |= UCB1_CS_LCD;
  P4DIR |= UCB1_CS_LCD;

  //P4 PIN 5
  P4SEL0 |=  UCB1CLK; // this defines it as a spi interface
  P4SEL1 &= ~UCB1CLK; //P4SELn=(0,1)

  //P4 PIN 6
  P4SEL0 |=  UCB1SIMO; //
  P4SEL1 &= ~UCB1SIMO; //P4SELn=(0,1)

  //P4 PIN 7
  P4SEL0 |=  UCB1SOMI; //
  P4SEL1 &= ~UCB1SOMI; //P4SELn=(0,1)

}

//PORT 5
//left as GPIO output 0
void Init_Port_5(void){
  P5OUT=GPIO_all;
  P5SEL0=GPIO_all;
  P5SEL1=GPIO_all;
  P5DIR=output_all;

  //P5 PIN 0
  P5OUT &= ~IOT_RESET;
  //P5 PIN 1
  P5SEL0 &= ~IR_LED;
  P5SEL1 &= ~IR_LED;
  P5OUT |= IR_LED;
  P5DIR |= IR_LED;
  //P5 PIN 2
  //P5 PIN 3
  //P5 PIN 4
  //P5 PIN 5
  //P5 PIN 6
  //P5 PIN 7
}

//PORT 6
//the lcd backlight is enabled. this a higher level non-gpio function.
void Init_Port_6(void){
  P6OUT=GPIO_all;
  P6SEL0=GPIO_all;
  P6SEL1=GPIO_all;
  P6DIR=output_all;

  //P6 PIN 0 R_FORWARD

  P6SEL0 |= R_FORWARD;
  P6SEL1 &= ~R_FORWARD;
  P6DIR |=R_FORWARD;
  P6OUT |= R_FORWARD;
  //P6 PIN 1
  P6SEL0 |= L_FORWARD;
  P6SEL1 &= ~L_FORWARD;
  P6DIR |= L_FORWARD;
  P6OUT |= L_FORWARD;
  //P6 PIN 2
  P6SEL0 |= R_REVERSE;
  P6SEL1 &= ~R_REVERSE;
  P6DIR |= R_REVERSE;
  P6OUT |= R_REVERSE;
  //P6 PIN 3
  P6SEL0 |= L_REVERSE;
  P6SEL1 &= ~L_REVERSE;
  P6DIR |= L_REVERSE;
  P6OUT |= L_REVERSE;
  //P6 PIN 4
  //P6DIR  &= ~LCD_BACKLIGHT; //backlight signal is an output
  P6OUT |= LCD_BACKLIGHT; //comment out to turn off light
  //P6 PIN 5
  //P6 PIN 6
  //P6 PIN 7
}



//this is the main function you will be calling externally
//it runs an initiation function for every port 1 through 6 sequentially.
void Init_Ports(void){
  Init_Port_1();
  Init_Port_2();
  Init_Port_3(USE_SMCLK);
  Init_Port_4();
  Init_Port_5();
  Init_Port_6();
  IOT_COUNTDOWN = 10;
}
```