MSP430 FRAM Automated Vehicle

[An Embedded Systems Enterprise]

Revision	Description
1	Updated Block Diagram to have “Control Module” and “Motors” **10/13/2018 by Tim Davis**
2	Incorporated Control Module into Sect. 3.6 **10/13/2018 by Tim Davis**
3	Eliminated white space between headers and body text **10/13/2018 by Abdalla Hablas, David Gosnell and Matthew Bradley**
4	Included “FRAM: Ferroelectric Random-Access Memory” in abbreviations **10/13/18 David Gosnell**
5	Incorporated Control Module into Sect. 4.3 Control Module **10/13/2018 by Abdalla Hablas**
6	Consolidated 4.4 into 4.2, making it say Backlight/LCD instead of just Backlight **10/13/2018 by Abdalla Hablas**
7	Fixed table of contents and table of figures **10/13/2018 by Matthew Bradley**
8	Fixed spelling **10/27/2018 Matthew Bradley and Abdalla Hablas**
9	Fixed font **10/27/18 Matthew Bradley and Abdalla Hablas**
10	Revised/Updated the flow chart in section 3 **10/28/2018 Timothy Davis**
11	Created flow charts for Software main.c, interrupts.c, and ADC.c **10/28/2018 Matthew Bradley and Timothy Davis**
12	Inserted section 3.7 and 4.4 for IR emitters and detectors and their descriptions **10/28/2018 Matthey Bradley and Timothy Davis**
13	Added section 6.2 **10/28/2018 Timothy Davis**

Team Members
Matthew Bradley
Abdalla Hablas
David Gosnel
Timothy Davis

Originator: Matthew, Abdalla, David, Timothy

Checked: 11/12/18 | Released:11/12/18

Filename: Automated Vehicle

Title: **MSP430 FRAM Automated Vehicle**

Date:	Document Number:	Rev:	Sheet:
11/12/2018	**1-2456-435-4561-12**	**D**	**1 of 50**

14	Added section 6.3 **10/28/2018 Matthew Bradley**
15	Added section 8.1 & 9.1 **10/28/2019 Matthew Bradley**
16	Added section 8.2, 8.3 , 9.2, 9.3 **10/28/18 Timothy Davis**
17	Added section 8.4 & 9.4 10/28/18 **David Gosnell**
18	Added section 8.5 & 9.5 10/28/18 **Abdalla Hablas**
19	Added section 7 **10/29/18 David Gosnell**
20	Fixed magic numbers in code on 9.1 **11/10/18 Matthew Bradley**
21	Added schematics to all sections of 4 and updated 4.1 sumary **11/10/18 Matthew Bradley**
22	Added missing abreviations **11/11/18 Abdalla Hablas**
23	Added 4.2 and 4.3 to Table of Contents **11/11/18 Matthew Bradley**
24	More detail added to 4.1, 4.2, 4.3, 4.4 **11/11/18 Matthew Bradley, David Gosnell, Abdella Hablas, and Tim Davis**
25	Added more useful and relavant information to section 6 **11/11/18 Tim Davis and Abdalla Hablas**
26	Arrows added to flow charts **11/11/18 Tim Davis**
27	Section 9 code revised and Serial.c added **11/11/18 Matthew Bradley, David Gosnell, Abdalla Hablas, and Tim Davis**
28	Section 7 subsections added **11/11/18 Matthew Bradley, David Gosnell, Abdalla Hablas, and Tim Davis**

Table of Contents

Table of Figures

1. Scope

This document covers the electronics used to build a programmable vehicle with a range of different functions including 3 different shape patterns: Figure 8, Triangle, Circle. Included with the board is an LCD and push buttons that provides a GUI for choosing and displaying the shapes that can be made by the vehicle. The vehicle also includes sensors that can detect and follow a black line. An additional debugging feature for the ADC feature on the MSP430 is an analog thumb wheel to verify that the ADC is working properly.

2. Abbreviations

ADC: Analog-to-Digital Converter

FRAM: Ferroelectric Random-Access Memory

GP I/O: General Purpose Input/ Output

GUI: Graphical User Interface

IAR: Ingenjörsfirman Anders Rundgren (compiler and debugger)

LCD: Liquid Crystal Display

MSP-430: MSP-430/FRAM

N-FET: Negative Field Effect Transistor

PCB: Printed Circuit Board

P-FET: Positive Field Effect Transistor

PWM: Pulse Width Modulation

SD: Secure Digital

USB: Universal Serial Bus

3. Overview

The system uses a LCD and backlight to give user feedback, while using a MSP430FR5994 FRAM board as the operating board. A power switch turns on the device, and the FRAM communicates with the power PCB. Buttons 1 and 2 are incorporated to switch between Thumbwheel settings and IR ON/OFF settings. The display shows real-time updates of the IR detector values.

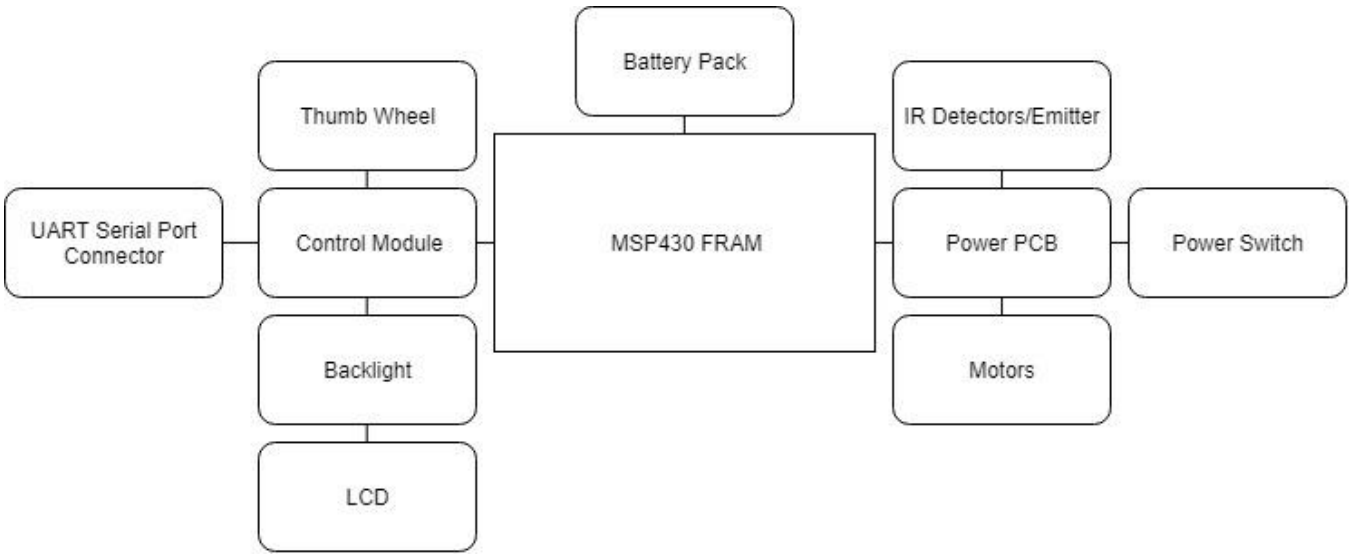


Figure 1: Block Diagram

3.1 MSP-430/FRAM

MSP-430/FRAM contains the microprocessor which stores and runs the code downloaded to the MSP-430. The signals from the precompiled code is then sent to the other parts of the system relevant to the commands being ran. The board is equipped with two analog buttons to provide user input.

3.2 Battery pack

The Battery pack contains 4 “AA” batteries that power the MSP-430, backlight, motors and LCD screen. Power information is displayed in section 4.

3.3 User Interface Blocks (Backlight and LCD)

The User Interface Block contains the LCD which is used to interact with the viewer by giving feedback pertaining to the current task that the system is performing.

3.4 System Input Block (Power Switch)

The Input block contains the power switch which supplies power to the MSP-430 and LCD.

3.5 Power Supply Block (Power PCB)

The Power Supply is used to regulate the voltage from 6V to the 3.3V operation voltage of the MSP-430 .

3.6 Control Module Shield (Control Module and Motors)

The Control Module has a H-Bridge that interfaces between the FRAM and the DC motor to control the movement.

3.7 IR LED emitter/detector

The IR LED is what is used to bounce infrared light onto an object and bounce back to the detectors to see if a color has changed.

4. Hardware

4.1 MSP430FR5994/FRAM

The FRAM board is the primary processor of the system. It also houses and runs the code, sends signals and communicates back and forth between the modules. The board takes the code produced in IAR and converts it into usable commands between the board and the modules the board houses three buttons on the FRAM board, two that are programmable (S1 & S2) and one reset button (S3). Two LEDs one red and one green are also included on the board and are configured by using P1.0 and P1.1. To program code on to the FRAM board J102 holds the Micro USB port for transferring code. While the FRAM board itself only has 256KB of memory, there is also a micro SD card slot for expandable storage.

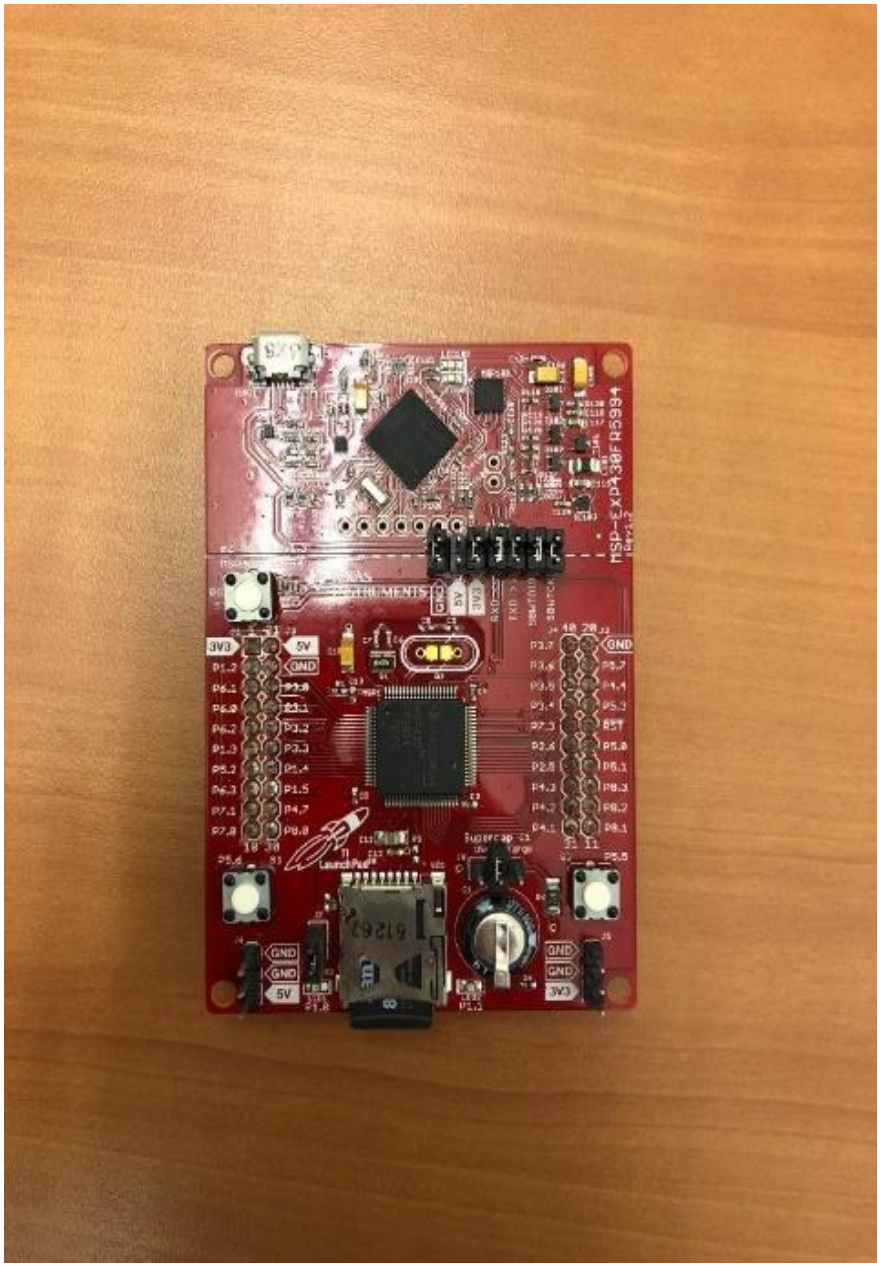
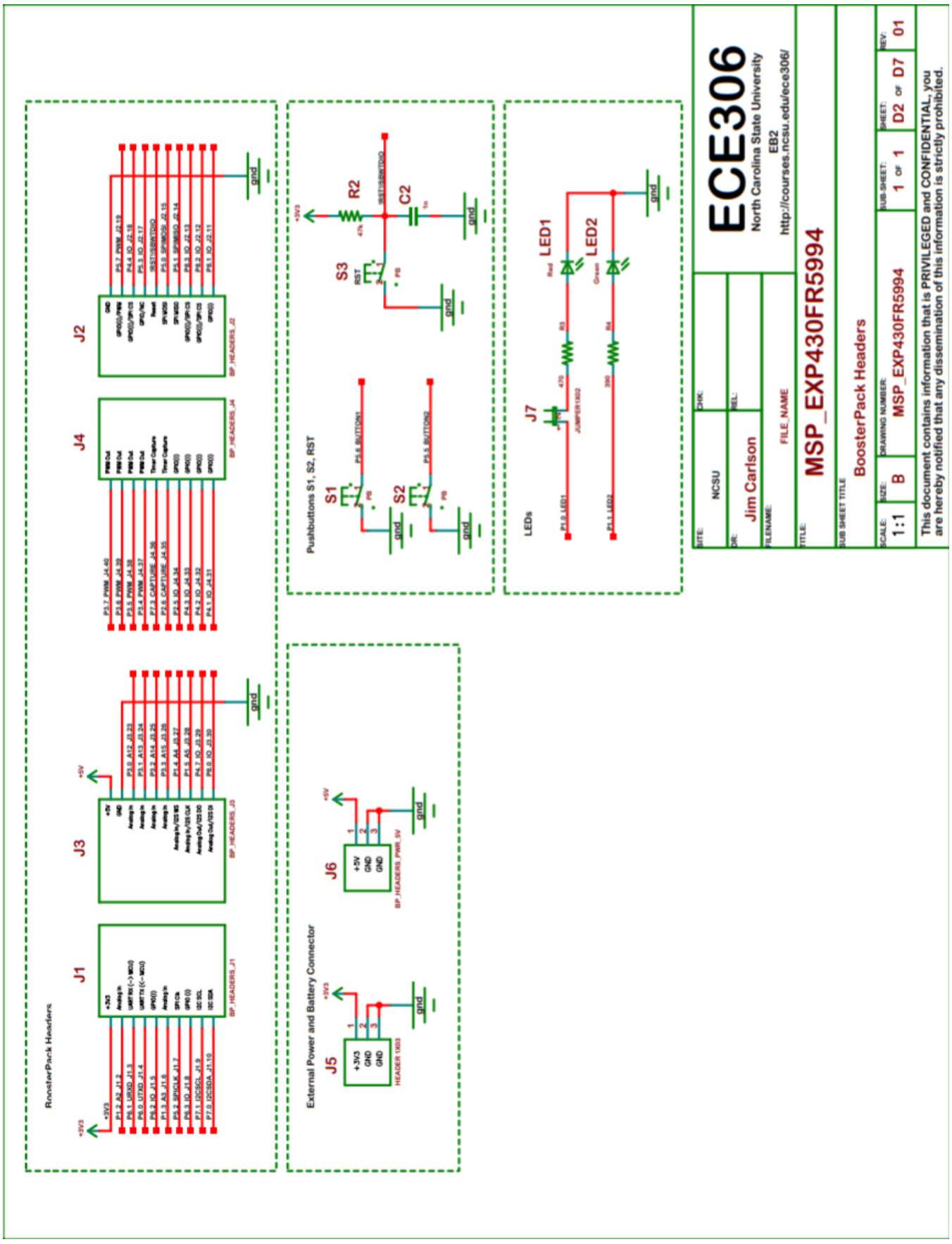
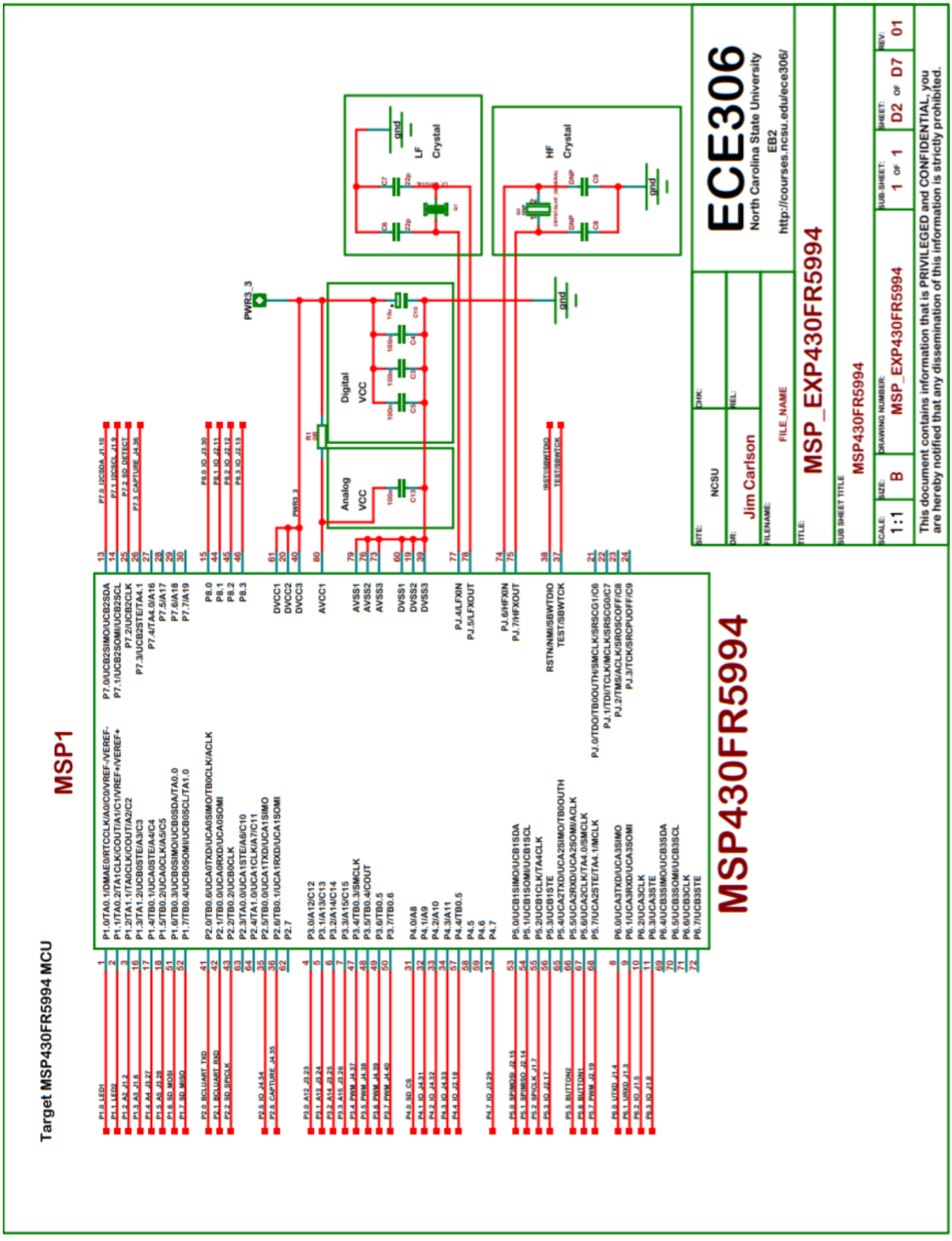
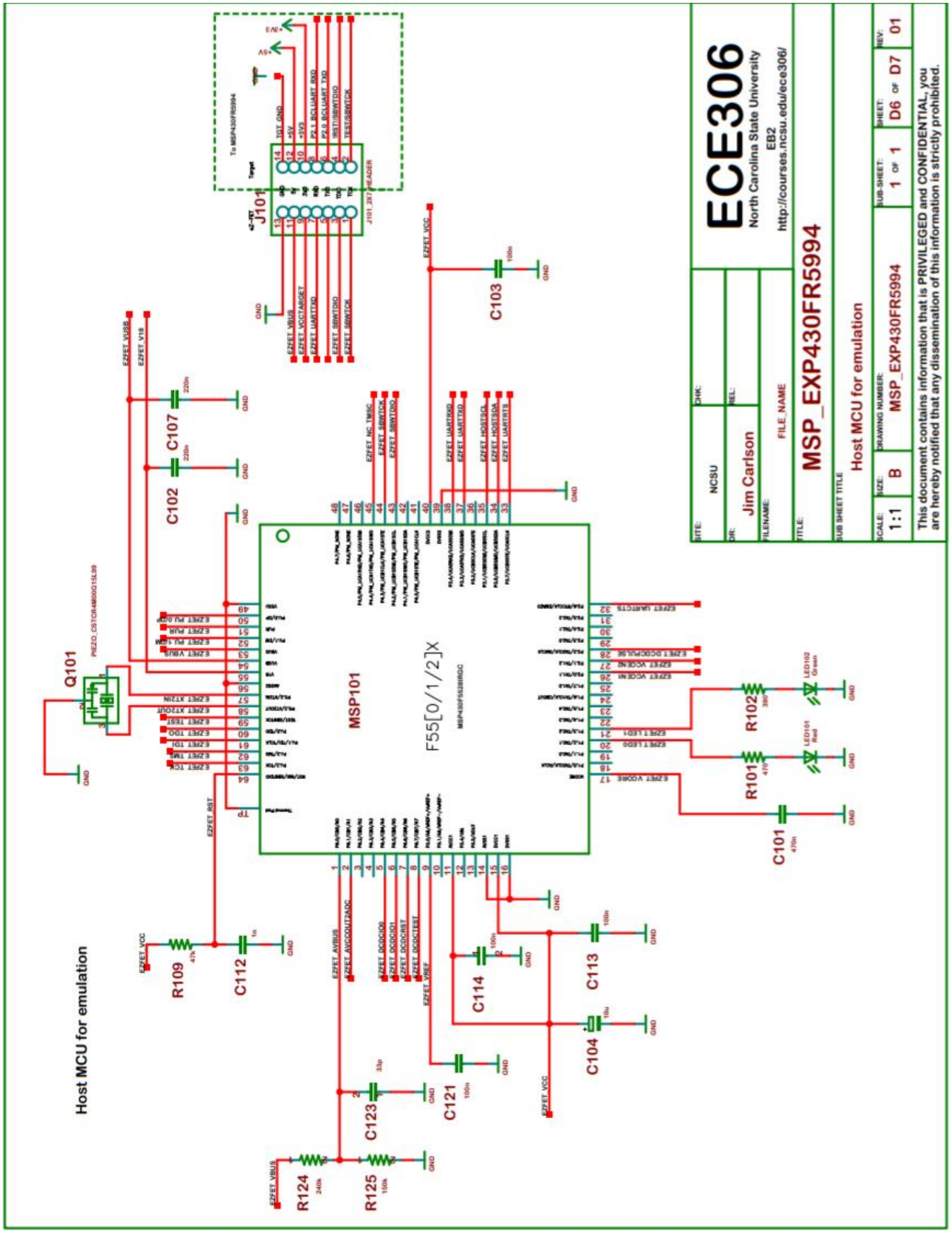
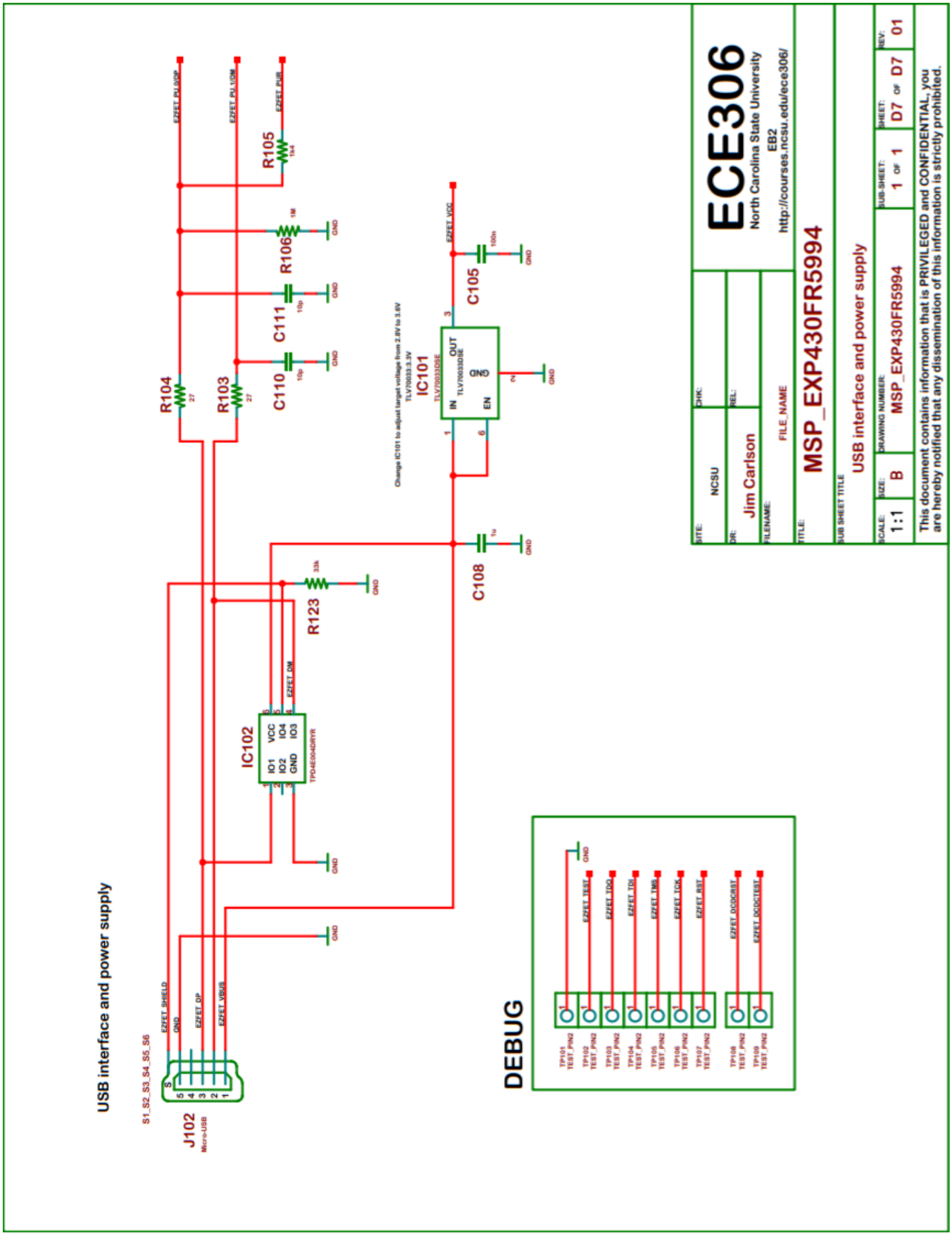


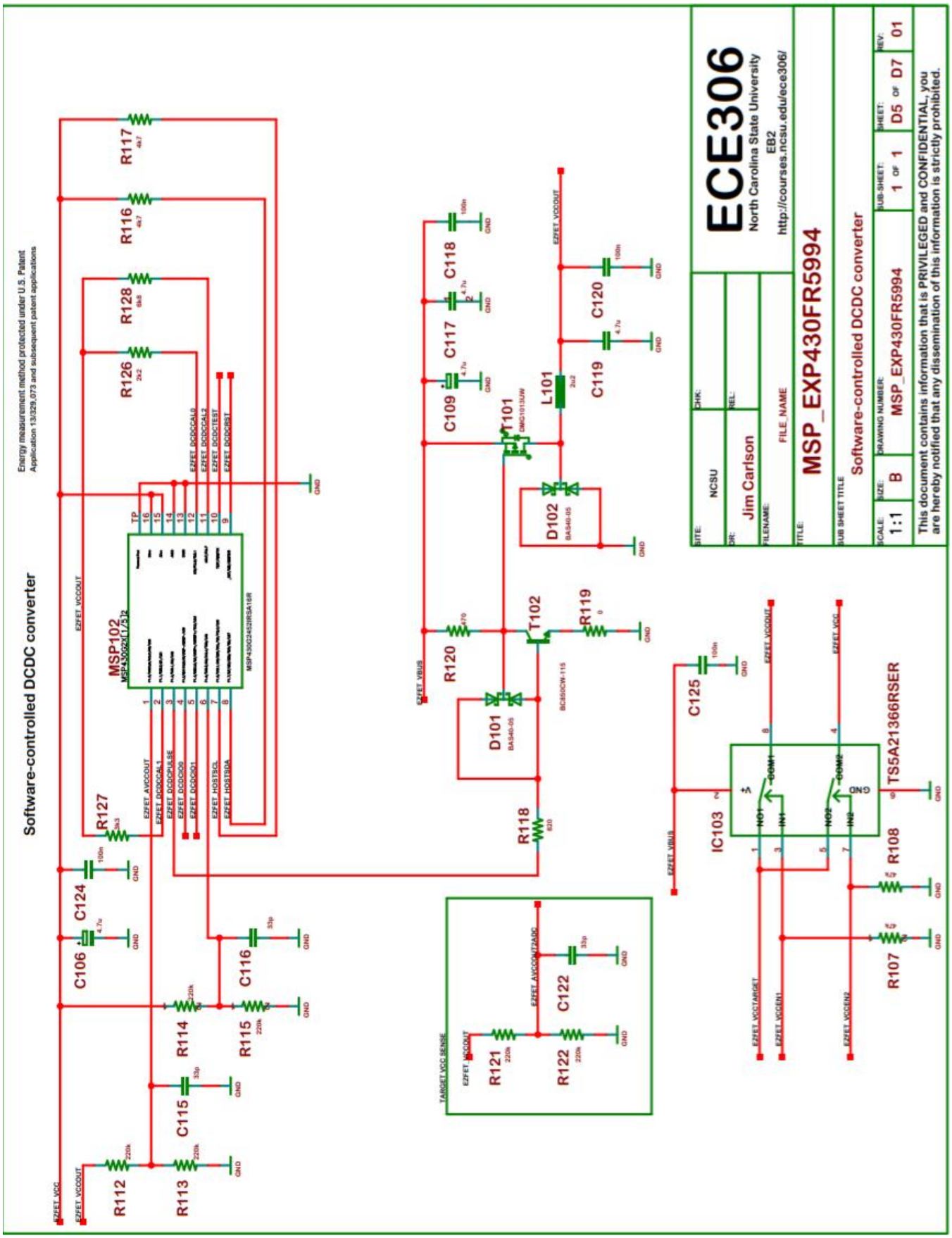
Figure 2: MSP430/FRAM


ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
MSP_EXP430FR5994
BoosterPack Headers
Jim Carlson
NCSU
FILE_NAME
SIZE: B
DRAWING NUMBER: MSP_EXP430FR5994
SUB-SHEET: 1 OF 1
SHEET: D2 OF D7
REV: 01
SCALE: 1:1
This document contains information that is PRIVILEGED and CONFIDENTIAL, you are hereby notified that any dissemination of this information is strictly prohibited.
BoosterPack Headers
J1
J2
J3
J4
Pushbuttons S1, S2, RST
S1
S2
S3
R2
C2
LEDs
LED1
LED2
J7
External Power and Battery Connector
J5
J6
gnd
+3V3
+5V


Target MSP430FR5994 MCU
MSP1
MSP430FR5994
ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
NCSU
Jim Carlson
FILE_NAME
MSP_EXP430FR5994
MSP430FR5994
MSP_EXP430FR5994
1 OF 1
D2 OF D7
01
1:1
B
This document contains information that is PRIVILEGED and CONFIDENTIAL; you are hereby notified that any dissemination of this information is strictly prohibited.
PWR3_3
Digital
VCC
Analog
VCC
LF
Crystal
HF
Crystal
gnd


ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
SITE: NCSU
DR: Jim Carlson
FILENAME:
CHK:
REL:
FILE_NAME
TITLE: MSP_EXP430FR5994
SUB SHEET TITLE: Host MCU for emulation
SCALE: 1:1
SIZE: B
DRAWING NUMBER: MSP_EXP430FR5994
SUB-SHEET: 1 OF 1
SHEET: D6 OF D7
REV: 01
This document contains information that is PRIVILEGED and CONFIDENTIAL., you are hereby notified that any dissemination of this information is strictly prohibited.
Host MCU for emulation
MSP101
F55[0/1/2]X
J101
C101
C102
C103
C104
C107
C112
C113
C114
C121
C123
R101
R102
R109
R124
R125
Q101

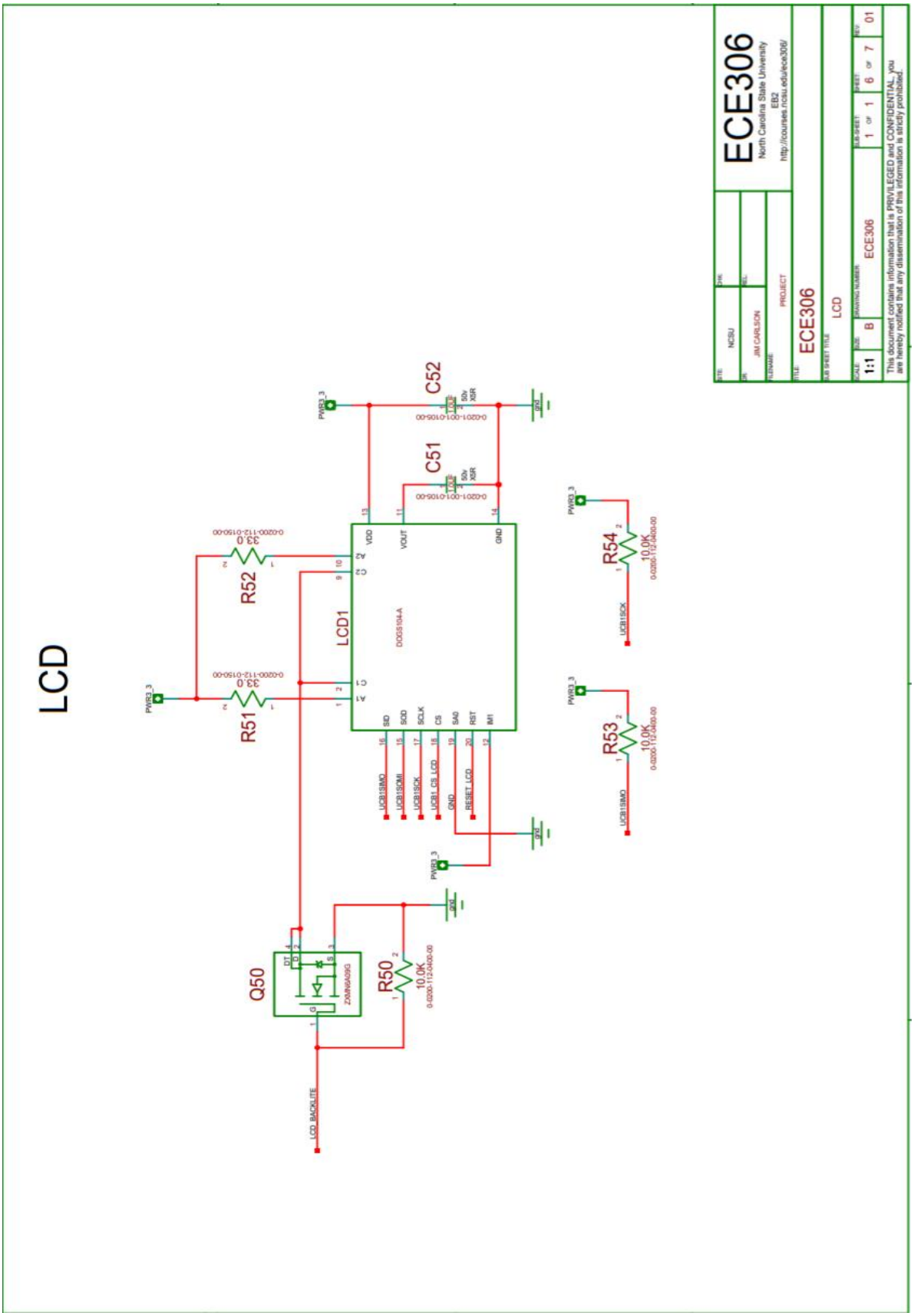

ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
SITE: NCSU
DR: Jim Carlson
CHK:
REL:
FILENAME:
FILE_NAME
TITLE: MSP_EXP430FR5994
SUB SHEET TITLE: USB interface and power supply
SCALE: 1:1
SIZE: B
DRAWING NUMBER: MSP_EXP430FR5994
SUB-SHEET: 1 OF 1
SHEET: D7 OF D7
REV: 01
This document contains information that is PRIVILEGED and CONFIDENTIAL; you are hereby notified that any dissemination of this information is strictly prohibited.
USB interface and power supply
DEBUG
J102
Micro-USB
IC101
IC102
R103
R104
R105
R106
R123
C105
C108
C110
C111
EZFET_VBUS
EZFET_DP
EZFET_DM
EZFET_VCC
EZFET_TEST
EZFET_TDO
EZFET_TDI
EZFET_TMS
EZFET_TCK
EZFET_RST
EZFET_DCDCRST
EZFET_DCDCTEST
GND


Software-controlled DCDC converter
Energy measurement method protected under U.S. Patent Application 13/329,073 and subsequent patent applications
ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
SITE: NCSU
DR: Jim Carlson
FILENAME:
FILE_NAME
TITLE: MSP_EXP430FR5994
SUB SHEET TITLE: Software-controlled DCDC converter
SCALE: 1:1
SIZE: B
DRAWING NUMBER: MSP_EXP430FR5994
SUB-SHEET: 1 OF 1
SHEET: D5 OF D7
REV: 01
This document contains information that is PRIVILEGED and CONFIDENTIAL, you are hereby notified that any dissemination of this information is strictly prohibited.
MSP102
MSP430G2452IRSA16R
IC103
TS5A21366RSER
T101
T102
D101
D102
L101
TARGET VCC SENSE
EZFET_VCCOUT
EZFET_VBUS
EZFET_VCC
EZFET_AVCCOUT
EZFET_VCCTARGET
EZFET_VCCEN1
EZFET_VCCEN2
GND

4.2 Backlight/ LCD

The LCD is a screen that, in combination with the backlight, provides a visual readout to the user. It provides the user with information such as the name of the shape that it is creating, status of which button is pressed. The Power Module transmits data to the LCD, thus allowing it to update real-time information. At its current state, the LCD screen can only display characters of the Ascii Table. The display size is limited to four lines, each holding 10 characters. These display lines are let up as two-dimensional arrays. This allows them to be easily changed. The backlight can be easily turned on and off using port manipulation. To conserve battery, it is recommended that the user keeps the backlight off when at all possible.

Figure 3: LCD/Backlight


LCD
ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
NCSU
JIM CARLSON
PROJECT
ECE306
LCD
SIZE
B
DRAWING NUMBER
ECE306
SCALE
1:1
SHEET
1 OF 1
SUB-SHEET
6 OF 7
REV
01
This document contains information that is PRIVILEGED and CONFIDENTIAL; you are hereby notified that any dissemination of this information is strictly prohibited.
C52
C51
R52
R51
R54
R53
R50
Q50
LCD1
DOGS104-A
VDD
VOUT
GND
SID
SCD
SCLK
CS
SA0
RST
IM1
RESET_LCD
LCD_BACKLITE
UCB1SIMO
UCB1SOMI
UCB1SCK
UCB1_CS_LCD
UCB1SCK
UCB1SIMO
PWR3_3
gnd

4.3 Control Module

The control module holds four P-FETs and nine N-FETS. The four P-FETs and eight of the nine N-FETs control the forward and reverse movement. While the last N-FET controls the IR LEDS and the detectors sensors. The N-FETs and P-FETs that are directly connected to the motor allow the usage of PWM. What PWM is, is a section of code that gives the user control over the voltage regulation to the motors. This allows for finer control over the speed of the motors regardless of the direction of travel.

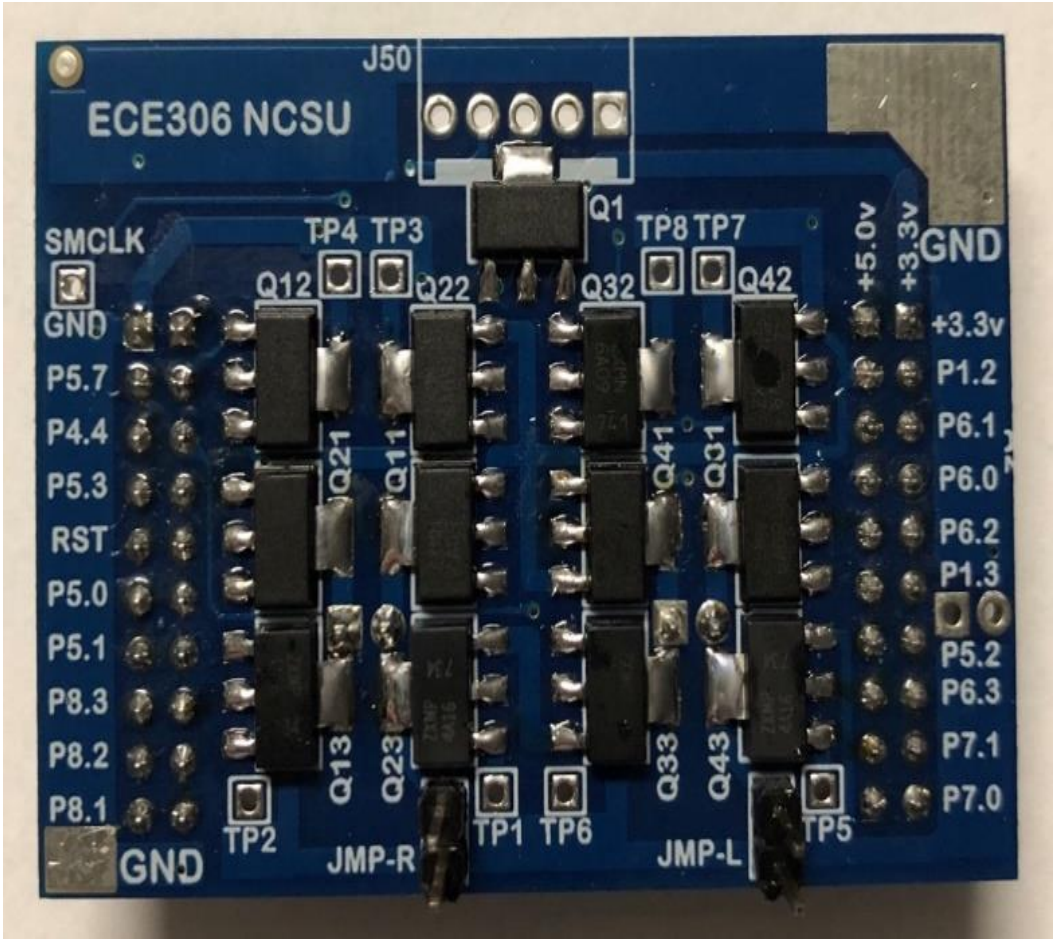
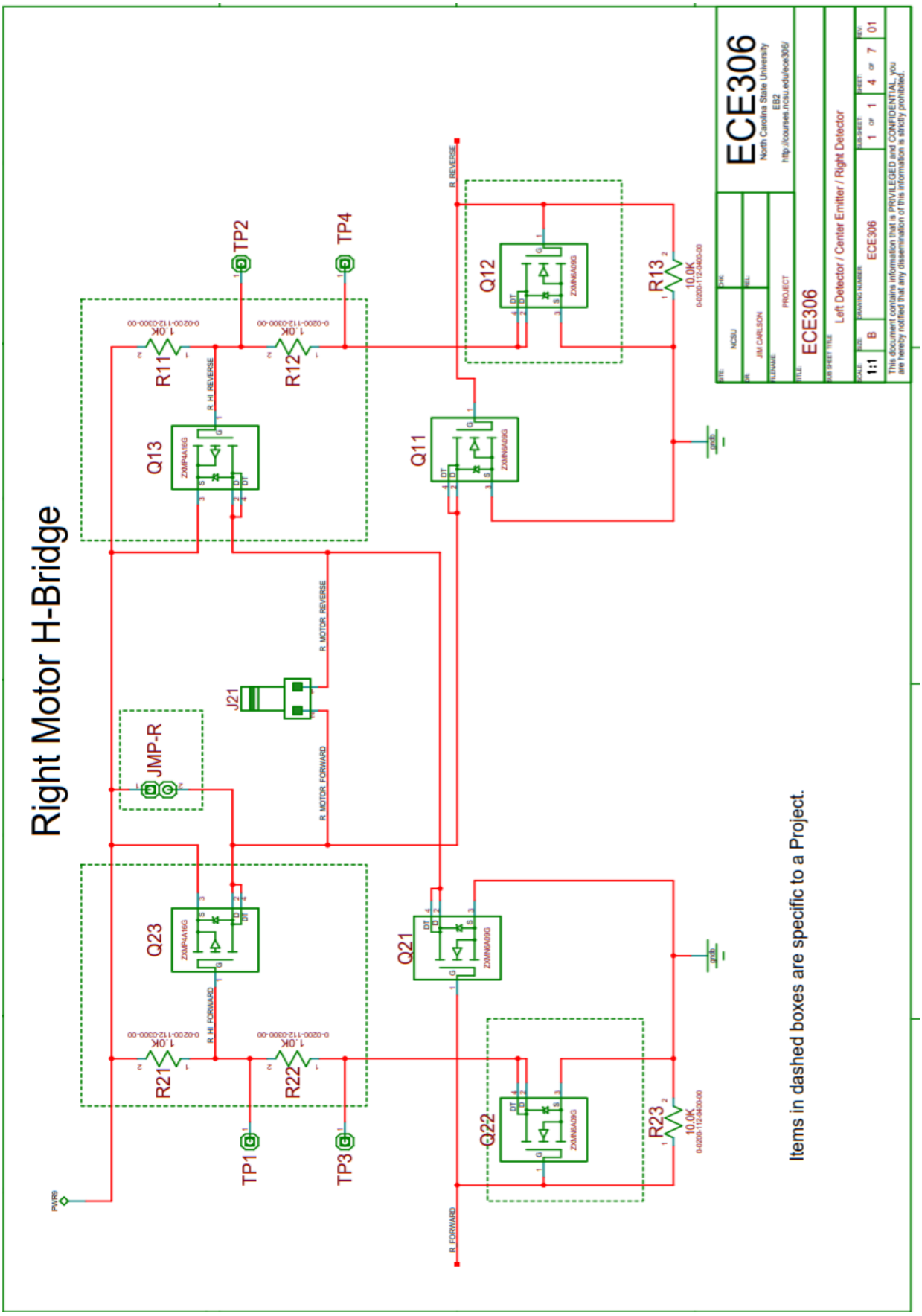
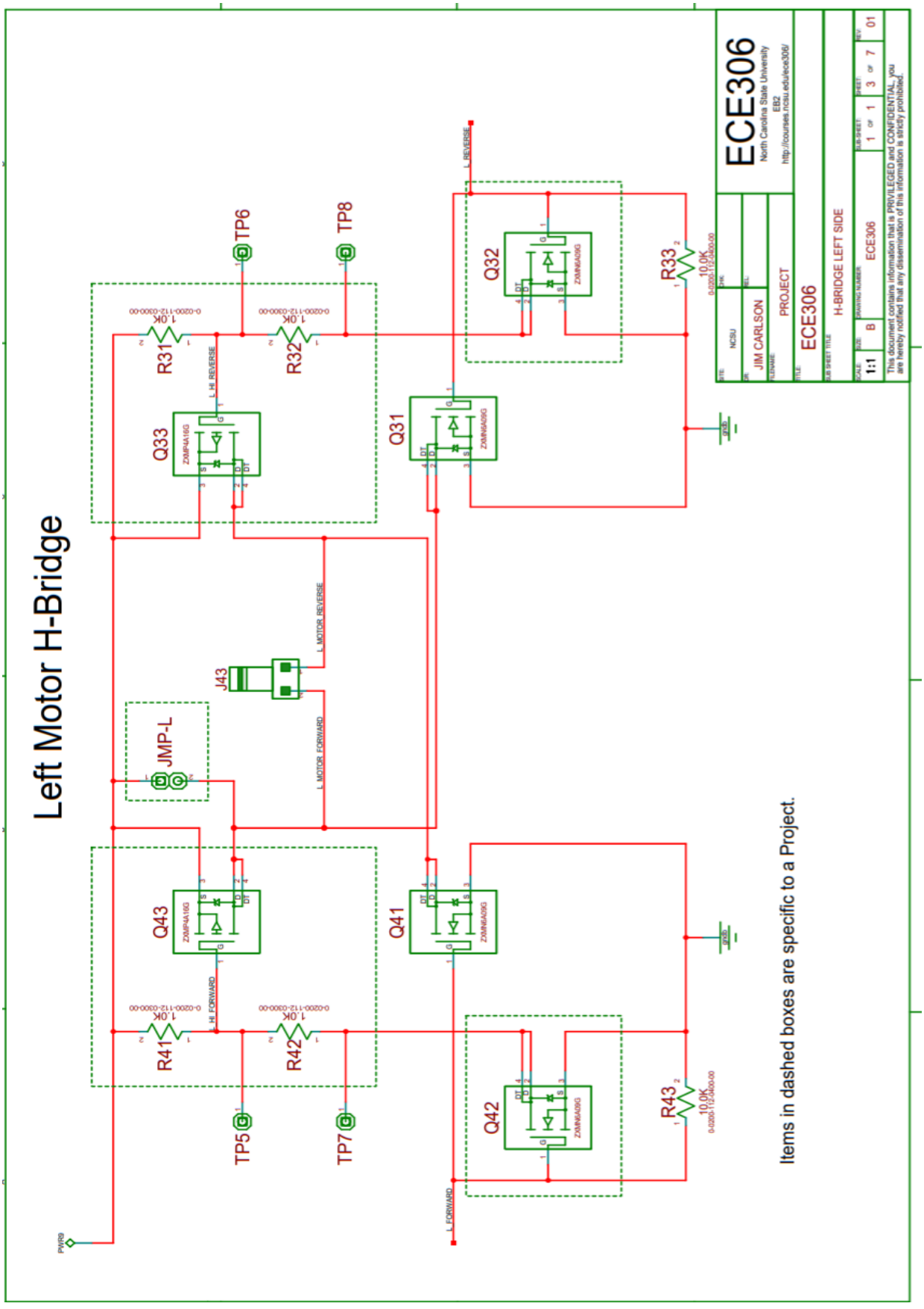


Figure 4: Control Module



Date:	Document Number:	Rev:	Sheet:
11/12/2018	1-2456-435-4561-12	D	16 of 50


Left Motor H-Bridge
ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
JIM CARLSON
NCSU
PROJECT
ECE306
H-BRIDGE LEFT SIDE
ECE306
B
1:1
1 OF 1
3 OF 7
01
This document contains information that is PRIVILEGED and CONFIDENTIAL; you are hereby notified that any dissemination of this information is strictly prohibited.
Items in dashed boxes are specific to a Project.
TP6
TP8
TP5
TP7
R31
R32
R41
R42
R33
R43
1.0K
10.0K
Q33
Q43
Q31
Q41
Q32
Q42
J43
JMP-L
L HI REVERSE
L HI FORWARD
L MOTOR REVERSE
L MOTOR FORWARD
L REVERSE
L FORWARD
PWRB
gnd

4.4 IR LED/Detectors

The IR LED/Detectors help transmit and detect infrared signals to the surface beneath the vehicle. These are the eyes of the vehicle. The intensity of the infrared beams detected by the left and right detectors determine whether the vehicle is resting on a black line or a white surface. This intensity of the IR beams has values ranging anywhere from 0-4096. The IR LED can be easily turned on and off using port manipulation. To conserve battery, it is recommended that the user keeps the backlight off when at all possible.

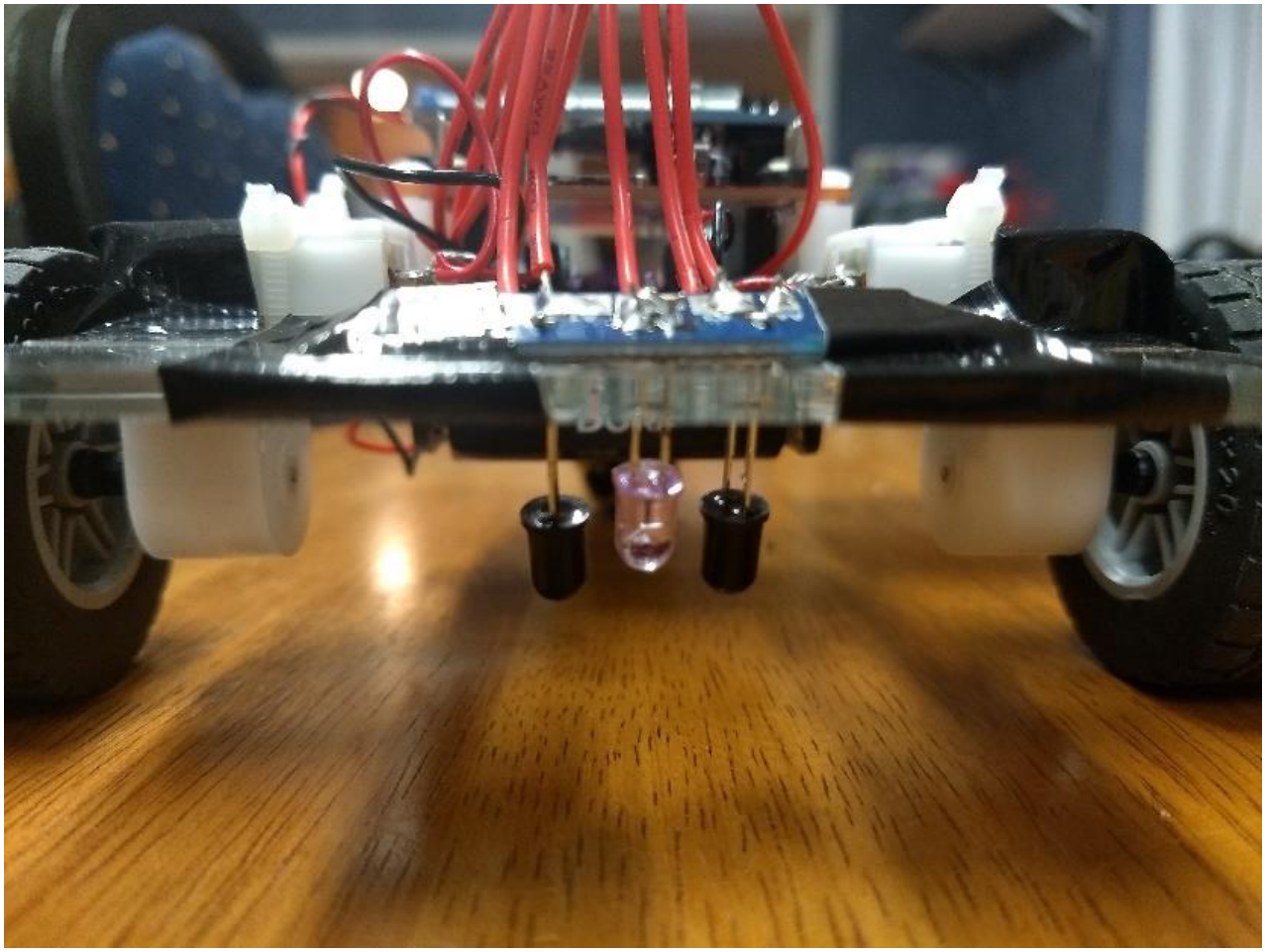
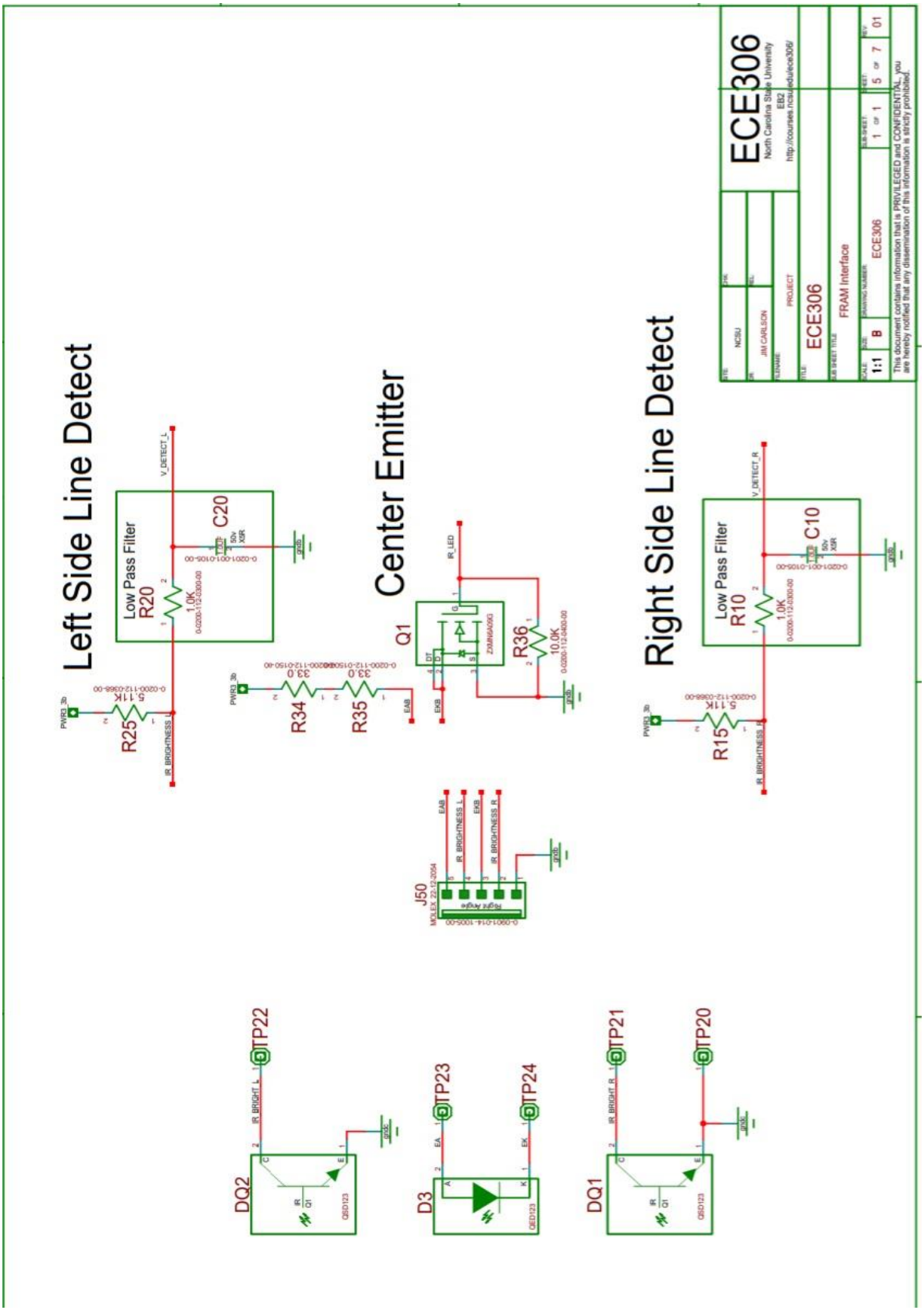


Figure 5: IR LED Emitter/Detectors


Left Side Line Detect
Center Emitter
Right Side Line Detect
Low Pass Filter
R20
1.0K
C20
R25
5.11K
R34
33.0
R35
33.0
Q1
R36
10.0K
Low Pass Filter
R10
1.0K
C10
R15
5.11K
J50
TP22
TP23
TP24
TP21
TP20
DQ2
D3
DQ1
ECE306
North Carolina State University
EB2
http://courses.ncsu.edu/ece306/
JIM CARLSON
PROJECT
ECE306
FRAM Interface
ECE306
1:1
B
1 OF 1
5 OF 7
01
This document contains information that is PRIVILEGED and CONFIDENTIAL. you are hereby notified that any dissemination of this information is strictly prohibited.

6. Test Process

This section describes what was used to test the different components of the system.

6.1 Power PCB

Use 4.5V power supply and a multimeter. Connect the positive end of the power supply to J0 and ground to ground on the power PCB. Use the multimeter to check the output voltage to verify the proper voltage 3.3V. Connect the negative end of the multimeter to the ground connector on the power PCB and connect the positive end of the multimeter to Pin J12 to receive a voltage reading of 3.3V

6.2 LCD

By inserting characters into the display string, the user can confirm that proper updating of characters occurred as well as real-time information is being displayed on the screen. Correct port configurations as well as proper soldering of connections ensured that the LCD is functioning properly. The LCDs operating voltage connection is specified to operate around 3.3V, this voltage value confirms the LCD is set up correctly.

6.3 Motors

The recommended operating voltage range for this motor is 3 to 6 V, though the gear motor can start rotating at voltages as low as 0.6 V. At 4.5 V, the motor has a free-run current of 80 mA. An important note involving the motors is the fragile state of the tabs used to wire the motors to the rest of the system. It is recommended that the user is careful with soldering the tabs, and that the user should add points of strain relief for the wires of the motor. Following the previous recommendations will increase the longevity of the motor tabs.

7. Software

After all the initialization functions are called, the main loop starts. The main loop has two primary functions, taking the values seen by the IR sensors and the Thumb Wheel and storing them into an array. Then pushing the arrays to the display for the user to see. The software uses a HEX to ADC converter function to store the readable values into the array that is copied to the display. The advantage of having the code in the main loop, over somewhere else in the code, is to allow the display to update live according to the IR sensors and the Thumb Wheel.

7.1 Main.c

Main.c is the heart of the code, it goes through and initializes functions that will be used like Ports.c. Once it gets into the main while loop it checks to see if it has received something, if it does not it keeps looping through. When it sees there is a change it copies the contents into a different array that just holds it for future use. It then displays the contents onto the display and says it “Received” the content. From there if button one is pressed the contents that was received will be transferred and “transfer” will be displayed along with what is being transferred. When button two is pressed the baud rate will changed between 115,200Hz and 460,800Hz.

7.2 ADC.c

ADC.c configures the following:

- ADC clock speed
- ADC resolution steps
- ADC mode
- ADC sampling rate
- ADC trigger mode
- ADC reference voltage

7.3 Interrupts.c

Interrupt.c is where any of the interrupts used in the program are stored. The interrupts that are used most frequently are the three analog interrupts, Button one and two and the thumb wheel. Sending and Receiving is dealt with in the Serial.c file.

7.4 Timers.c

Like the Interrupt.c file. Timers.c contains more interrupt functions. The main difference between the two files is that Timers.c interrupts pertain specifically to the timers used in the code. Button debouncing and display updating takes place in this section of the code.

7.5 Ports.c

Ports.c configures all ports to their proper functions. Each port contains eight pins that can be configured individually to digital GP I/O or a specific function determined by the MSP430 datasheet. Pins can be configured using the hex binary value associated with the pin number.

7.6 Serial.c

Interrupts for Serial port vectors are in Serial.c. Whenever a command or element is sent in through the specified serial port, interrupt flags are excited on UCA0/UCA3 interrupt vectors. The receive vector is configured to receive incoming characters and place them into a ring buffer until further use. The transmit vector is configured to transmit characters out, appending the string with a carriage return (0x0D) and a line feed (0x0A). The function “out character” helps implement the transmission process and insures no data is lost during transmission.

8. Software Flow Charts

This section describes what the code is doing using flow charts.

8.1 Main.c

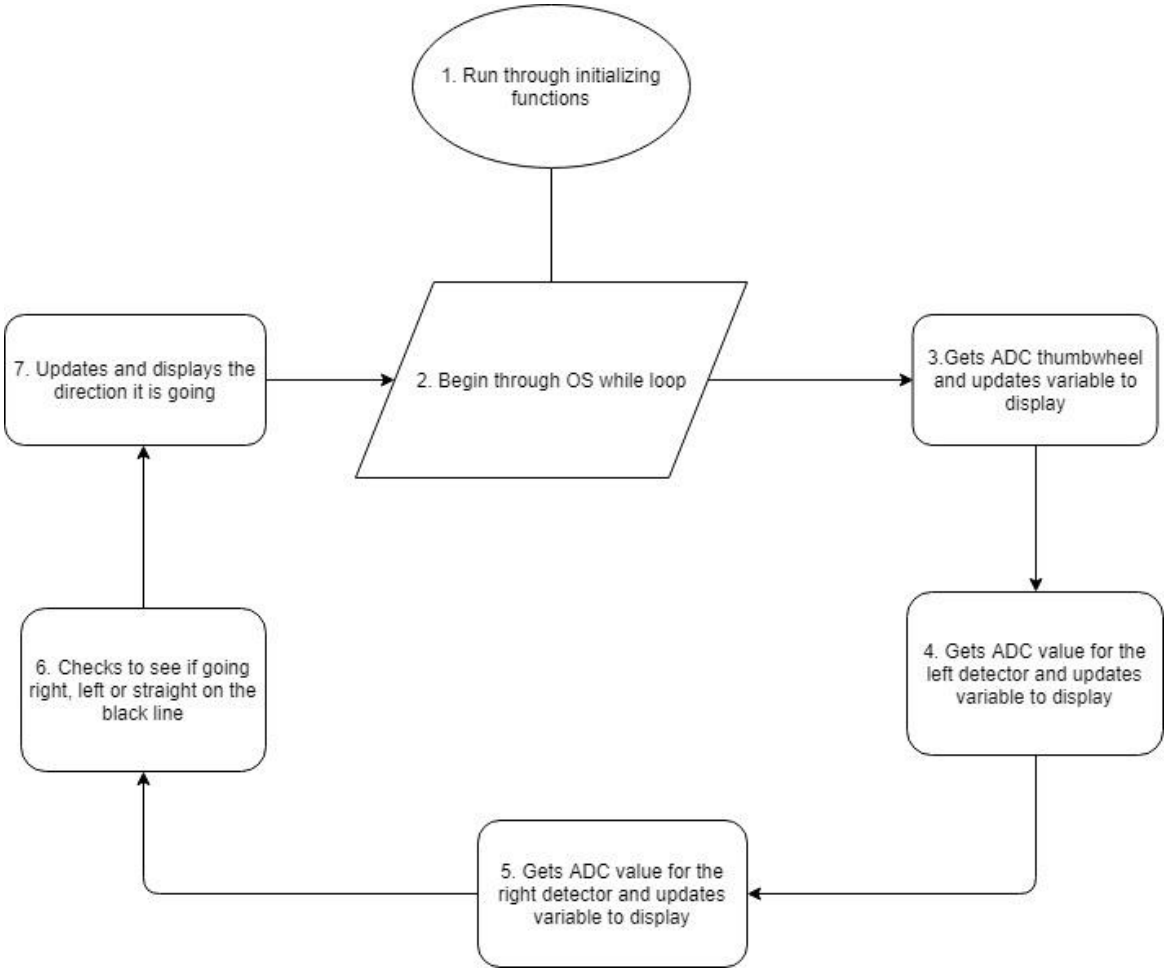


Figure 6: Main.c Flow Chart

8.2 ADC.c

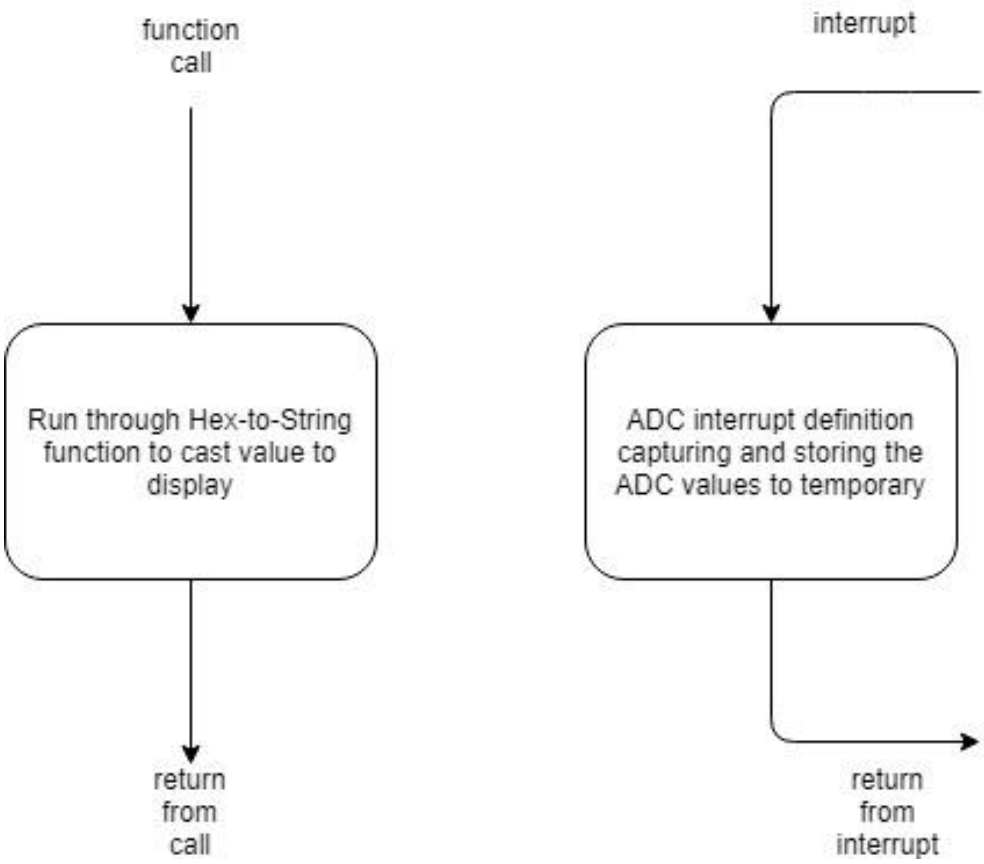


Figure 7 : ADC.c Flow Chart

8.3 Interrupts.c

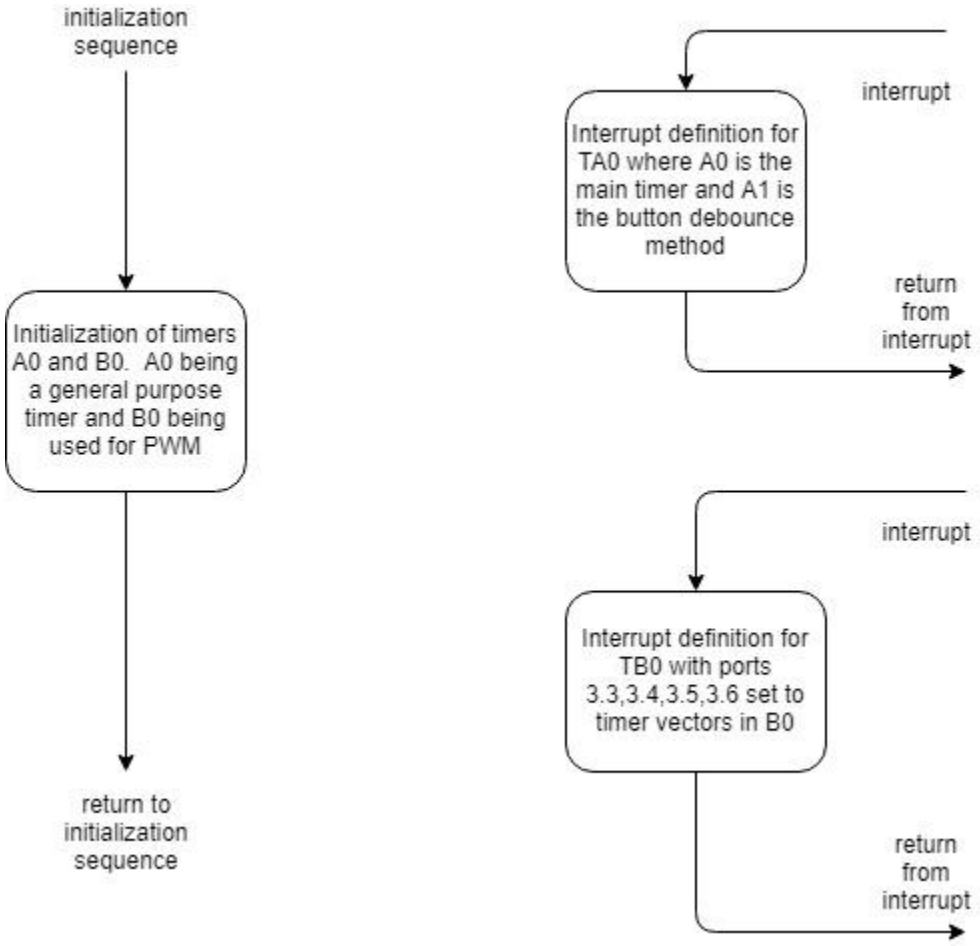


Figure 8: Interrupts.c Flow Chart

8.4 Timers.c

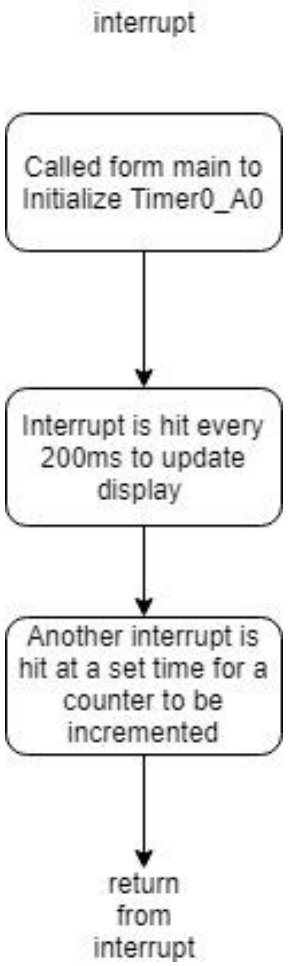


Figure 9: Timers.c Flow Chart

8.5 Ports.c

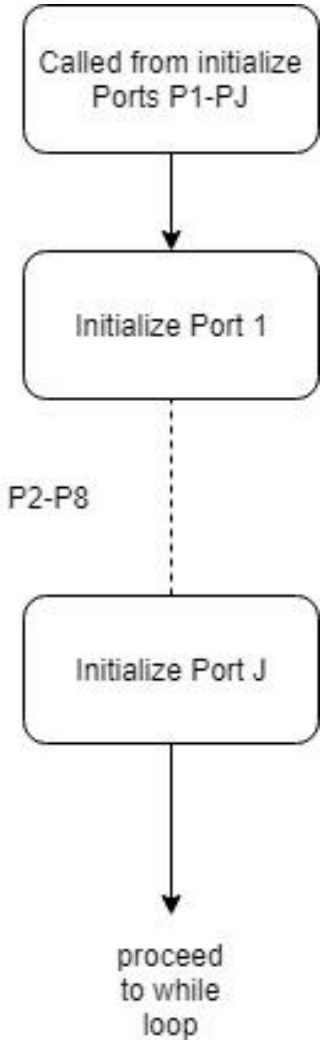


Figure 10: Ports.c Flow Char

8.6 Serial.c

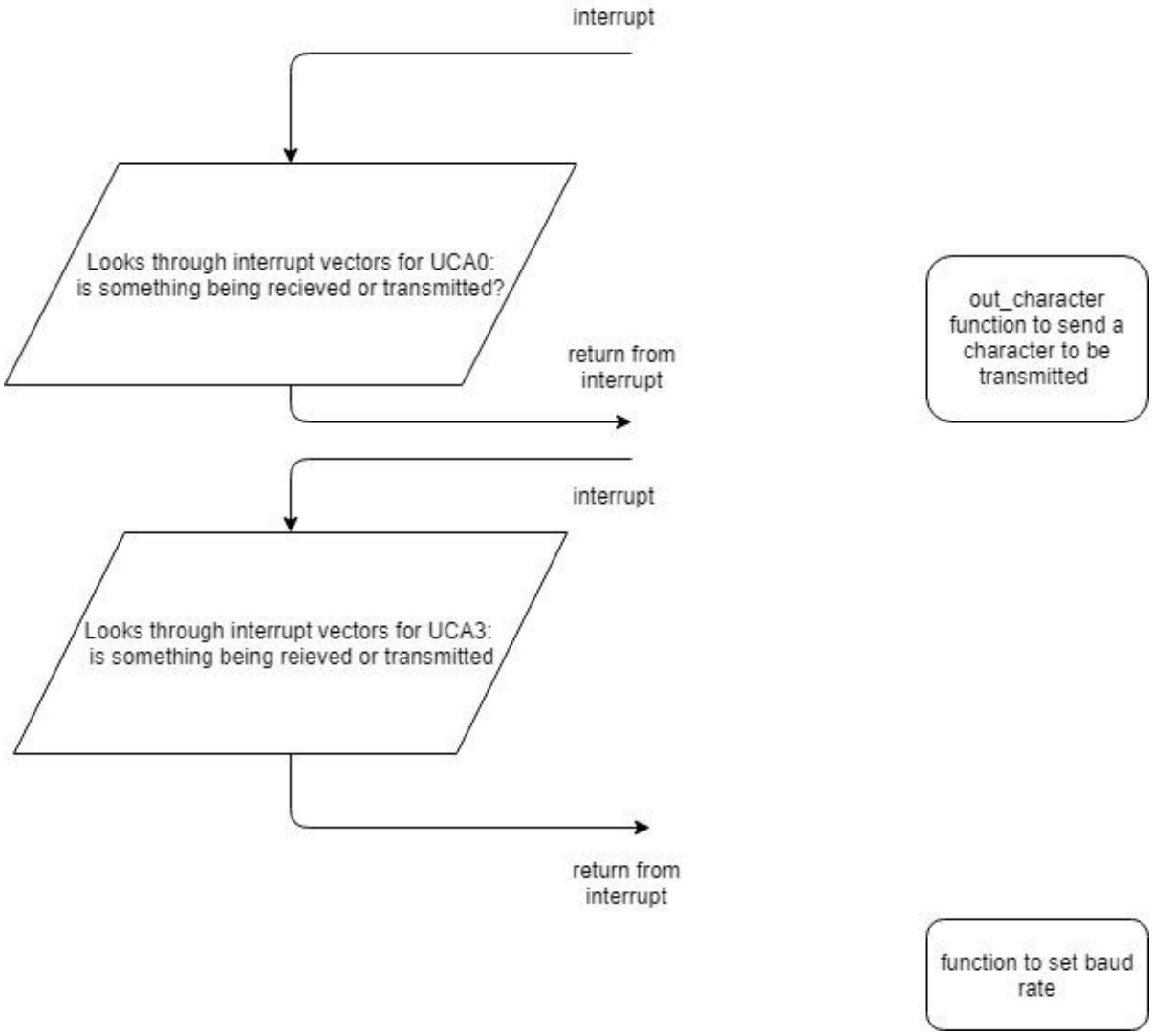


Figure 11: Serial.c Flow Chart

9. Software Listing

This section is a printout of the actual code that is used in the automated vehicle.

9.1 Main.c

```
//-------------------------------------------------------------------------------
//
//  This file contains the Main Routine - "While" Operating System
//This executes serial communication with the device
//
//  Matthew Bradley
//
//
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
#include  "msp430.h"
#include  "functions.h"
#include <string.h>
#include "macros.h"

// Global Variables
//-------------------------------------------------------------------------------
extern volatile unsigned int DEBOUNCE;
extern char display_line[LINE_FOUR][CHARACTER_ELEVEN];
extern char *display[LINE_FOUR];
extern unsigned char display_mode;
extern volatile unsigned char display_changed;
extern volatile unsigned char update_display;
extern volatile unsigned int update_display_count;
extern int unsigned T_COUNT;
extern volatile char USB_Char_Rx[LIMIT1];
extern volatile char HOLD_ARRAY[LIMIT1] = "$test";
volatile char HOLD_A2[LIMIT1];
extern volatile unsigned char Button_p ;
volatile unsigned char CHANGED ;
extern volatile unsigned int usb_rx_ring_wr;
 //-------------------------------------------------------------------------------------------
//Local Variables
//none

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_Timer_A0();                    // Initialize Timers
  Init_LCD();   // Initialize LCD
// Init_Serial_UCA0();
  Init_Serial_UCA3();
```

```
Button_p = 'N';
CHANGED = 'N';
UCA3_index = LOW;
UCA3TXBUF = test_command[LOW];
//-------------------------------------------------------------------------
// Begining of the "While" Operating System
//-------------------------------------------------------------------------
  while(ALWAYS) {                    // Can the Operating system run


    usb_rx_ring_wr = LOW;

    CHECK_REV();

    switch (CHANGED){
    case array_stuff :
      SWAP_ARRAY();
      break;
    }

    switch (Button_p){
    case PRESSED_1:
      Trans_B1();
      break;
    case PRESSED_2:
      Baud_change();
      break;
//      default break;
    }

    if(startt == LOW){
      strcpy(display_line[LINE_ZERO], "  Waiting ");
      strcpy(display_line[LINE_TWO], "  460,800 ");
      LCD_UPReset();

    startt = HI;
    }

      if (DEBOUNCE > dbounc ){
        P5IE |= BUTTON1;
        P5IE |= BUTTON2;
//        P5IFG |= BUTTON1;
//        P5IFG |= BUTTON2;
        TA0CCTL1 &= ~CCIE;
       DEBOUNCE = LOW;

      }
  Display_Process();
  LCD_UPReset()
  }

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

9.2 ADC.c

```
//-------------------------------------------------------------------------------
// this file configure ADC
// Abdalla Hablas
// ahablas@ncsu.edu
// ECE306
//-------------------------------------------------------------------------------

#include "macros.h"

//********************************** ADC 12 ************************************
//-------------------------------------------------------------------------------

// this function Configures ADC12
void Init_ADC(void){

  // ADC12CTL0 Register Description
  ADC12CTL0 = RESET;
  ADC12CTL0 |= ADC12SHT0_2;                 // 16 ADC clocks for sampling period
  ADC12CTL0 |= ADC12SHT1_2;                 // 16 ADC clocks for sampling period
  ADC12CTL0 |= ADC12MSC;               // First rising edge of the SHI signal triggers sampling timer
  ADC12CTL0 |= ADC12ON;                     // ADC12 on

  // ADC12CTL1 Register Description
  ADC12CTL1 = RESET;
  ADC12CTL1 |= ADC12PDIV_0;                 // Predivide ADC12_B clock source by 1
  ADC12CTL1 |= ADC12SHS_0;                  // sample-and-hold source ADC12SC
  ADC12CTL1 |= ADC12SHP;                   // SAMPCON signal is sourced from the sampling timer.
  ADC12CTL1 |= ADC12ISSH_0;                 // sample-input signal is not inverted
  ADC12CTL1 |= ADC12DIV_0;                 // / 1 clock divider
  ADC12CTL1 |= ADC12SSEL0;                  // ADC12OSC (MODOSC)
  ADC12CTL1 |= ADC12CONSEQ_3;                // Repeat-sequence-of-channels

  // ADC12CTL2 Register Description
  ADC12CTL2 = RESET;
  ADC12CTL2 |= ADC12RES_2;                  // 12-bit conversion results / 14 clock cycle conversion
  ADC12CTL2 |= ADC12DF_0;                  // data read format is Binary unsigned
  ADC12CTL2 |= ADC12PWRMD_0;       // Regular power mode where sample rate is not restricted

  // ADC12CTL3 Register Description
  ADC12CTL3 = RESET;
  ADC12CTL3 |= ADC12ICH3MAP_0;              // external pin is selected for ADC input channel A26
  ADC12CTL3 |= ADC12ICH2MAP_0;              // external pin is selected for ADC input channel A27
  ADC12CTL3 |= ADC12ICH1MAP_0;              // external pin is selected for ADC input channel A28
  ADC12CTL3 |= ADC12ICH0MAP_0;              // external pin is selected for ADC input channel A29
  ADC12CTL3 |= ADC12TCMAP_1;        // ADC internal temperature sensor ADC input channel A30
  ADC12CTL3 |= ADC12BATMAP_1;       // ADC internal 1/2 x AVCC is ADC input channel A31
  ADC12CTL3 |= ADC12CSTARTADD_0;

  // ADC12MCTL0 Register (Thumb wheel)
  ADC12MCTL0 = RESET;
  ADC12MCTL0 |= ADC12WINC_0;                // Comparator window disabled
  ADC12MCTL0 |= ADC12DIF_0;                // Single-ended mode enabled
  ADC12MCTL0 |= ADC12VRSEL_0;               // VR+ = AVCC, VR- = AVSS
  ADC12MCTL0 |= ADC12INCH_2;               // channel = A2 Thumb Wheel

  // ADC12MCTL1 Register (left detector)
```

```
  ADC12MCTL1 = RESET;
  ADC12MCTL1 |= ADC12WINC_0;              // Comparator window disabled
  ADC12MCTL1 |= ADC12DIF_0;              // Single-ended mode enabled
  ADC12MCTL1 |= ADC12VRSEL_0;             // VR+ = AVCC, VR- = AVSS
  ADC12MCTL1 |= ADC12INCH_5;             // channel = A5 Left

  // ADC12MCTL2 Register (right detector)
  ADC12MCTL2 = RESET;
  ADC12MCTL2 |= ADC12WINC_0;              // Comparator window disabled
  ADC12MCTL2 |= ADC12DIF_0;              // Single-ended mode enabled
  ADC12MCTL2 |= ADC12VRSEL_0;             // VR+ = AVCC, VR- = AVSS
  ADC12MCTL2 |= ADC12INCH_4;             // channel = A4 Right

  // ADC12MCTL3 Register (temp sensor)
  ADC12MCTL3 = RESET;
  ADC12MCTL3 |= ADC12WINC_0;              // Comparator window disabled
  ADC12MCTL3 |= ADC12DIF_0;              // Single-ended mode enabled
  ADC12MCTL3 |= ADC12VRSEL_0;             // VR+ = AVCC, VR- = AVSS
  ADC12MCTL3 |= ADC12INCH_30;             // Temp sensor

  // ADC12MCTL4 Register (battery monitor)
  ADC12MCTL4 = RESET;
  ADC12MCTL4 |= ADC12WINC_0;              // Comparator window disabled
  ADC12MCTL4 |= ADC12DIF_0;              // Single-ended mode enabled
  ADC12MCTL4 |= ADC12VRSEL_0;             // VR+ = AVCC, VR- = AVSS
  ADC12MCTL4 |= ADC12INCH_31;             // Battery voltage monitor
  ADC12MCTL4 |= ADC12EOS;                // End of Sequence

  // ADC12IER0-2 Register Descriptions
  ADC12IER0 = RESET;                    // Interrupts for channels  0 - 15
  ADC12IER1 = RESET;                    // Interrupts for channels 16 - 31
  ADC12IER2 = RESET;                    // Interrupts for ADC12RDYIE ADC12TOVIE ADC12OVIE

  // ADC12HIIE ADC12LOIE ADC12INIE
  //ADC12IER0 |= ADC12IE4;                // Generate Interrupt for MEM2 ADC Data load
  ADC12IER0 |= ADC12IE2;                // Generate Interrupt for MEM2 ADC Data load
  //ADC12IER0 |= ADC12IE0;                // Enable ADC conv complete interrupt

  ADC12CTL0 |= ADC12ENC;                // enable conversion
  ADC12CTL0 |= ADC12SC;                 // Start conversion

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

9.3 Interrupts.c

```
//-------------------------------------------------------------------------------
//
//  Description: This file contains the Fuctions deffinitions
//   called during the operation of the program
//
//
//  David Gosnell
//  Sep 2018
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//-------------------------------------------------------------------------------
```

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

extern volatile unsigned int Time_Sequence;
unsigned int cycle_time;
extern unsigned int time_change;
extern volatile unsigned char event = HIGH;
extern unsigned int GO;
extern unsigned int my_time;

int press_count = LOW;
extern unsigned int debounce_SW1;
extern unsigned int debounce_SW2;
extern volatile unsigned int debounce_SW1_count;
extern volatile unsigned int debounce_SW2_count;

extern unsigned int action;
extern unsigned int Sending = LOW;
extern unsigned int Receiving = LOW;

extern volatile unsigned int White_Thresh = LOW;
extern volatile unsigned int Black_Thresh = LOW;

extern unsigned int surface;
extern unsigned int matts_angry_number;

extern unsigned int ADC_Thumb;
extern unsigned int ADC_Right_Detector;
extern unsigned int ADC_Left_Detector;
extern char adc_char[QUAD];
extern char NCSU_NO1[NCSUARRAY];

extern volatile unsigned int Time_Seconds;
extern char display_line[CHAR_SIZEX][CHAR_SIZEY];

//MISC============================================================================
void out_character(char character){
//-------------------------------------------------------------------------------
// The while loop will stall as long as the Flag is not set [port is busy]
while (!(UCA3IFG & UCTXIFG)); // USCI_A0 TX buffer ready?
Sending = HIGH;
UCA3TXBUF = character;
//-------------------------------------------------------------------------------
}

//================================================================================
// Interupts

#pragma vector=PORT5_VECTOR
__interrupt void BUTTON_interrupt(void){
//Button 1
  if (P5IFG & BUTTON1)  {
    //Debounce
    P5IE &= ~BUTTON1;                       //Disabling Button1 interrupt for the duration of the interupt
```

```
    debounce_SW1 = HIGH;                        // Setting a varible to identify the button has been pressed
    debounce_SW1_count = LOW;                   // Resetting the debounce counter
    //Action
    matts_angry_number = LOW;
    Sending = HIGH;
  }
//Button 2
  if  (P5IFG & BUTTON2)  {
    //Debounce
    P5IE &= ~BUTTON2;                           //Disabling Button1 interrupt for the duration of the interupt
    debounce_SW2 = HIGH;                        // Setting a varible to identify the button has been pressed
    debounce_SW2_count = LOW;                   // Resetting the debounce counter
    //Action
    switch(event){
      case CASE0:
        UCA0BRW = BRW460; // 460,800 Baud
        UCA0MCTLW = TLW460 ;
        UCA3BRW = BRW460; // 460,800 Baud
        UCA3MCTLW = TLW460 ;
        event = CASE1;
        break;
      case CASE1:
        UCA0BRW = BRW115; // 115,200 Baud
        UCA0MCTLW = TLW115 ;
        UCA3BRW = BRW115; // 115,200 Baud
        UCA3MCTLW = TLW115 ;
        event = CASE0;
        break;
    }
  }
}
#pragma vector = ADC12_B_VECTOR
__interrupt void ADC12_ISR(void){
switch(__even_in_range(ADC12IV, ADC12IV__ADC12RDYIFG)){
    case ADC12IV__NONE:
      break;                                    // Vector 0: No interrupt
    case ADC12IV__ADC12OVIFG:
      break;                                    // Vector 2: ADC12MEMx Overflow
    case ADC12IV__ADC12TOVIFG:
      break;                                    // Vector 4: Conversion time overflow
    case ADC12IV__ADC12HIIFG:
      break;                                    // Vector 6: ADC12BHI
    case ADC12IV__ADC12LOIFG:
      break;                                    // Vector 8: ADC12BLO
    case ADC12IV__ADC12INIFG:
      break;                                    // Vector 10: ADC12BIN
    case ADC12IV__ADC12IFG0:
      break;                                    // Vector 12: ADC12MEM0 Interrupt
    case ADC12IV__ADC12IFG1:
      break;                                    // Vector 14: ADC12MEM1 Interrupt
    case ADC12IV__ADC12IFG2:                                // Vector 16: ADC12MEM2 Interrupt
      ADC_Thumb = ADC12MEM0;                                 // A02 ADC10INCH_2
      ADC_Right_Detector = ADC12MEM1;                        // A05 ADC10INCH_4
      ADC_Left_Detector = ADC12MEM2;                         // A04 ADC10INCH_5
      break;
    case ADC12IV__ADC12IFG3:
      break; // Vector 18: ADC12MEM3
```

```
  case ADC12IV__ADC12IFG4:
    break; // Vector 20: ADC12MEM4
  case ADC12IV__ADC12IFG5:
    break; // Vector 22: ADC12MEM5
  case ADC12IV__ADC12IFG6:
    break; // Vector 24: ADC12MEM6
  case ADC12IV__ADC12IFG7:
    break; // Vector 26: ADC12MEM7
  case ADC12IV__ADC12IFG8:
    break; // Vector 28: ADC12MEM8
  case ADC12IV__ADC12IFG9:
    break; // Vector 30: ADC12MEM9
  case ADC12IV__ADC12IFG10:
    break; // Vector 32: ADC12MEM10
  case ADC12IV__ADC12IFG11:
    break; // Vector 34: ADC12MEM11
  case ADC12IV__ADC12IFG12:
    break; // Vector 36: ADC12MEM12
  case ADC12IV__ADC12IFG13:
    break; // Vector 38: ADC12MEM13
  case ADC12IV__ADC12IFG14:
    break; // Vector 40: ADC12MEM14
  case ADC12IV__ADC12IFG15:
    break; // Vector 42: ADC12MEM15
  case ADC12IV__ADC12IFG16:
    break; // Vector 44: ADC12MEM16
  case ADC12IV__ADC12IFG17:
    break; // Vector 46: ADC12MEM17
  case ADC12IV__ADC12IFG18:
    break; // Vector 48: ADC12MEM18
  case ADC12IV__ADC12IFG19:
    break; // Vector 50: ADC12MEM19
  case ADC12IV__ADC12IFG20:
    break; // Vector 52: ADC12MEM20
  case ADC12IV__ADC12IFG21:
    break; // Vector 54: ADC12MEM21
  case ADC12IV__ADC12IFG22:
    break; // Vector 56: ADC12MEM22
  case ADC12IV__ADC12IFG23:
    break; // Vector 58: ADC12MEM23
  case ADC12IV__ADC12IFG24:
    break; // Vector 60: ADC12MEM24
  case ADC12IV__ADC12IFG25:
    break; // Vector 62: ADC12MEM25
  case ADC12IV__ADC12IFG26:
    break; // Vector 64: ADC12MEM26
  case ADC12IV__ADC12IFG27:
    break; // Vector 66: ADC12MEM27
  case ADC12IV__ADC12IFG28:
    break; // Vector 68: ADC12MEM28
  case ADC12IV__ADC12IFG29:
    break; // Vector 70: ADC12MEM29
  case ADC12IV__ADC12IFG30:
    break; // Vector 72: ADC12MEM30
  case ADC12IV__ADC12IFG31:
    break; // Vector 74: ADC12MEM31
  case ADC12IV__ADC12RDYIFG:
```

```
      break; // Vector 76: ADC12RDY
    default:
      break;
}
}

//-------------------------------------------------------------------------
//*************************************************************************
// Hex to BCD Conversion
// Convert a Hex number to a BCD for display on an LCD or monitor
//
//-------------------------------------------------------------------------
void HEXtoBCD(int hex_value){
  int value = LOW;
  adc_char[DLINE1] = '0';
  adc_char[DLINE2] = '0';
  adc_char[DLINE3] = '0';
  adc_char[DLINE4] = '0';
  while (hex_value > HUNDREDS){
    hex_value = hex_value - BIGLARGE;
    value++;
    adc_char[DLINE1] = ASCIIOFFSET + value;
  }
  value = LOW;
  while (hex_value > TENS){
    hex_value = hex_value - FRANKLIN;
    value++;
    adc_char[DLINE2] = ASCIIOFFSET + value;
  }
  value = LOW;
  while (hex_value > ONES){
    hex_value = hex_value - HAM;
    value++;
    adc_char[DLINE3] = ASCIIOFFSET + value;
  }
  adc_char[DLINE4] = ASCIIOFFSET + hex_value;
}
//*************************************************************************
          //-------------------------------------------------------------------------
```

9.4 Timers.c

```
//-------------------------------------------------------------------------------
//
//  Description: This file contains the Timer interrupts
//   called during the operation of the program
//
//
//  David Gosnell
//  Sep 2018
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//-------------------------------------------------------------------------
#include  "msp430.h"
#include  "functions.h"
#include <string.h>
#include  "macros.h"

//Globals=========================================================================
```

```
extern volatile unsigned int Time_Sequence = LOW;
extern volatile unsigned char update_display;
extern unsigned int debounce_SW1 = LOW;
extern unsigned int debounce_SW2= LOW;
extern volatile unsigned int debounce_SW1_count = LOW;
extern volatile unsigned int debounce_SW2_count = LOW;
extern unsigned int section = LOW;
extern unsigned int Phase = LOW;
extern unsigned int GO;
unsigned int pause = HIGH;

extern volatile unsigned int hold;
extern unsigned int delayed = LOW;

extern volatile unsigned int Time_Seconds = LOW;
extern int Timer_Off;

int screen_count = LOW;
int backlight_count = LOW;
int bounce_count = LOW;
int debounce_check = LOW;
int loop_count = LOW;
int loop2_cnt = LOW;

int twosecpause = LOW;
int Dongle_Count = LOW;

//-------------------------------------------------------------------------
// Timer A0 initializaiton sets up both A0_0, A0_1-A0_2 and overflow

void Init_Timer_A0(void) {
  TA0CTL = TASSEL__SMCLK;                   // SMCLK source
  TA0CTL |= TACLR;                          // Resets TA0R, clock Divider, count direction
  TA0CTL |= MC__CONTINOUS;                  // Continuous up
  TA0CTL |= ID__2;                          // Deicid clock by 2

  TA0EX0 = TAIDEX__8;                       // Divide clock by an additional 8

  TA0CCR0 = TA0CCR0_INTERVAL;               // CCR0
  TA0CCTL0 |= CCIE;                         // CCR0 enable interupt

  TA0CCR1 = TA0CCR1_INTERVAL;               // CCR1
  TA0CCTL1 |= CCIE;                         // CCR1 enable interupt

  TA0CCR2 = TA0CCR2_INTERVAL;               // CCR2
  TA0CCTL2 |= CCIE;                         // CCR2 enable interupt

  TA0CTL &= ~TAIE;                          // Disable Overflow Interrupt
  TA0CTL &= ~TAIFG;                         // Clear Overflow Interrupt Flag
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer0_A0_ISR(void){
//-------------------------------------------------------------------------
// TimerA0 0 Interrupt handler
//-------------------------------------------------------------------------
  //..... Add What you need happen in the interrupt .....
```

```
  Time_Sequence++;                          // Time_Sequence Timer
  if(Time_Sequence >= TS_THRESHOLD){
    Time_Sequence = LOW;
  }


  TA0CCR0 += TA0CCR0_INTERVAL;                    // Add Offset to TACCR0
//-------------------------------------------------------------------------
}

#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void){
//-------------------------------------------------------------------------
// TimerA0 1-2, Overflow Interrupt Vector (TAIV) handler
//-------------------------------------------------------------------------
  switch(__even_in_range(TA0IV,FORTNIGHT)){
    case LOW: break;                            // No interrupt
    case TWICE:                     // Case 2
      screen_count++;
      if(screen_count >= SCREEN_COUNT){
        update_display = HIGH;
        screen_count = LOW;
      }
      TA0CCR1 += TA0CCR1_INTERVAL;                  // Add Offset to TACCR1
      break;
    case DEBOUNCE_CASE:                         // Case 4
      loop_count++;                         // This is the DeBounce timer for SW1 and SW2
      if(loop_count <= LOOP_COUNT){                 // Remove this if() and change macro back if this doesnt
work
        bounce_count++;
        if(debounce_SW1){
          debounce_SW1_count++;
        }
        if(debounce_SW2){
          debounce_SW2_count++;
        }
        if(debounce_SW1_count >= BOUNCE_LIMIT){
          P5IE |= BUTTON1;
          P5IFG &= ~ BUTTON1;
          debounce_SW1 = LOW;
          debounce_SW1_count = LOW;
        }
        if(debounce_SW2_count >= BOUNCE_LIMIT){
          P5IE |= BUTTON2;
          P5IFG &= ~ BUTTON2;
          debounce_SW2 = LOW;
          debounce_SW2_count = LOW;
        }
        loop_count = LOW;
      }
      TA0CCR2 += TA0CCR2_INTERVAL;                  // Add Offset to TACCR2
      break;
    case FORTNIGHT:                             // overflow Case 14
      break;
    default: break;
  }
//-------------------------------------------------------------------------
```

```
}

void Init_Timer_B0(void) {
//-------------------------------------------------------------------------------
// SMCLK source, up count mode, PWM Right Side
// TB0.3 P3.4 L_REVERSE TB0.1 P3.6 R_REVERSE
// TB0.4 P3.5 L_FORWARD TB0.2 P3.7 R_FORWARD
//-------------------------------------------------------------------------------
  TB0CTL = TBSSEL__SMCLK;                       // SMCLK
  TB0CTL |= MC__UP;                           // Up Mode
  TB0CTL |= TBCLR;                           // Clear TAR
  TB0CCR0 = WHEEL_PERIOD;                       // PWM Period

  TB0CCTL3 = OUTMOD_7;                        // CCR1 reset/set
  LEFT_REVERSE_SPEED = WHEEL_OFF;                   // P3.4 Left Reverse PWM duty cycle

  TB0CCTL4 = OUTMOD_7;                        // CCR2 reset/set
  LEFT_FORWARD_SPEED = WHEEL_OFF;                   // P3.5 Left Forward PWM duty cycle

  TB0CCTL5 = OUTMOD_7;                        // CCR1 reset/set
  RIGHT_REVERSE_SPEED = WHEEL_OFF;                  // P3.6 Right Reverse PWM duty cycle

  TB0CCTL6 = OUTMOD_7;                        // CCR2 reset/set
  RIGHT_FORWARD_SPEED = WHEEL_OFF;                  // P3.7 Right Forward PWM duty cycle
          //-------------------------------------------------------------------------------
```

9.5 Ports.c

```
//-------------------------------------------------------------------------------
// this file to configure input and output ports on MSP430
// Abdalla Hablas
// ahablas@ncsu.edu
// ECE306
//-------------------------------------------------------------------------------

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


//************************************ PORT_1 *********************************
//-------------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_1
void Init_Port_1(void){

  P1SEL1 = RESET;             // RESET PORT
  P1SEL0 = RESET;             // RESET PORT

  //PIN 0
  P1SEL1 &= ~RED_LED;          // SET PIN AS GP I/O
  P1SEL0 &= ~RED_LED;           // SET PIN A GP I/O
  P1OUT &= ~RED_LED;           // LED OFF
  P1DIR |= RED_LED;            // SET PIN TO OUTPUT

  //PIN 1
```

```
  P1SEL1 &= ~GRN_LED;           // SET PIN AS GP I/O
  P1SEL0 &= ~GRN_LED;           // SET PIN A GP I/O
  P1OUT &= ~GRN_LED;           // LED OFF
  P1DIR |= GRN_LED;             // SET PIN TO OUTPUT

  //PIN 2
  P1SEL1 |= V_THUMB;           // ADC INPUT FOR THUMB WHEEL
  P1SEL0 |= V_THUMB;           // ADC INPUT FOR THUMB WHEEL
  //P1DIR &= ~V_THUMB;          // ADC IN

  //PIN 3
  P1SEL1 &= ~TEST_PROBE;       // SET PIN AS GP I/O
  P1SEL0 &= ~TEST_PROBE;       // SET PIN AS GP I/O
  P1OUT &= ~TEST_PROBE;          // SET PIN LOW
  P1DIR |= TEST_PROBE;          // SET PIN TO OUTPUT

  //PIN 4
  P1SEL1 |= V_DETECT_R;          // ADC INPUT FOR RIGHT DETECTOR
  P1SEL0 |= V_DETECT_R;          // ADC INPUT FOR RIGHT DETECTOR

  //PIN 5
  P1SEL1 |= V_DETECT_L;          // ADC INPUT FOR LEFT DETECTOR
  P1SEL0 |= V_DETECT_L;          // ADC INPUT FOR LEFT DETECTOR

  //PIN 6
  P1SEL1 |= SD_MOSI_UCB0SIMO;       // UCB0SIMO_SD CARD SIMO
  P1SEL0 &= ~SD_MOSI_UCB0SIMO;       // UCB0SIMO_SD CARD SIMO

  //PIN 7
  P1SEL1 |= SD_MISO_UCB0SOMI;       // UCB0SOMI_SD CARD MISO
  P1SEL0 &= ~SD_MISO_UCB0SOMI;       // UCB0SOMI_SD CARD MISO

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

//************************************ PORT_2 *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_2
void Init_Port_2(void){

  P2SEL1 = RESET;           // RESET PORT
  P2SEL0 = RESET;           // RESET PORT

  //PIN 0
  P2SEL1 |= USB_TXD;       // USB_TXD_UCA0
  P2SEL0 &= ~USB_TXD;       // USB_TXD_UCA0

  //PIN 1
  P2SEL1 |= USB_RXD;       // BCLUART_RXD_UCA0
  P2SEL0 &= ~USB_RXD;       // BCLUART_RXD_UCA0

  //PIN 2
  P2SEL1 |= SD_SPICLK;       // SD_SPICLK_USB0
  P2SEL0 &= ~SD_SPICLK;     // SD_SPICLK_USB0

  //PIN 3
  P2SEL1 &= ~P2_3;           // SET PIN AS GP I/O
```

```
  P2SEL0 &= ~P2_3;          // SET PIN A GP I/O
  //P2OUT |= P2_3;          // PULLUP RESISTOR
  P2DIR &= ~P2_3;           // SET PIN TO INPUT
  //P2REN |= P2_3;          // RESISTOR ENABLED

  //PIN 4
  P2SEL1 &= ~P2_4;          // SET PIN AS GP I/O
  P2SEL0 &= ~P2_4;          // SET PIN A GP I/O
  //P2OUT |= P2_4;          // PULLUP RESISTOR
  P2DIR &= ~P2_4;           // SET PIN TO INPUT
  //P2REN |= P2_4;          // RESISTOR ENABLED

  //PIN 5
  P2SEL1 |= UCA1TXD;       // UCA1TXD
  P2SEL0 &= ~UCA1TXD;      // UCA1TXD

  //PIN 6
  P2SEL1 |= UCA1RXD;       // UCA1RXD
  P2SEL0 &= ~UCA1RXD;      // UCA1RXD

  //PIN 7
  P2SEL1 &= ~P2_7;          // SET PIN AS GP I/O
  P2SEL0 &= ~P2_7;          // SET PIN A GP I/O
  //P2OUT |= P2_7;          // PULLUP RESISTOR
  P2DIR &= ~P2_7;           // SET PIN TO INPUT
  //P2REN |= P2_7;          // RESISTOR ENABLED

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

//************************************ PORT_3 *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_3
void Init_Port_3(char port_functionality){

  P3SEL1 = RESET;              // RESET PORT
  P3SEL0 = RESET;              // RESET PORT

  //PIN 0
  P3SEL1 &= ~IOT_RESET;        // SET PIN AS GP I/O
  P3SEL0 &= ~IOT_RESET;         // SET PIN A GP I/O
  P3OUT |= IOT_RESET;          // SET PIN HIGH
  P3DIR |= IOT_RESET;          // SET PIN TO OUTPUT

  //PIN 1
  P3SEL1 &= ~IOT_PROG_MODE;       // SET PIN AS GP I/O
  P3SEL0 &= ~IOT_PROG_MODE;       // SET PIN A GP I/O
  //P3OUT |= IOT_PROG_MODE;       // PULLUP RESISTOR
  P3DIR &= ~IOT_PROG_MODE;       // SET PIN TO INPUT
  //P3REN |= IOT_PROG_MODE;       // RESISTOR ENABLED

  //PIN 2
  P3SEL1 &= ~IOT_LINK;          // SET PIN AS GP I/O
  P3SEL0 &= ~IOT_LINK;          // SET PIN A GP I/O
  //P3OUT |= IOT_LINK;          // PULLUP RESISTOR
  P3DIR &= ~IOT_LINK;          // SET PIN TO INPUT
  //P3REN |= IOT_LINK;          // RESISTOR ENABLED
```

```
  //PIN 3
  P3SEL1 &= ~IOT_PROG_SEL;        // SET PIN AS GP I/O
  P3SEL0 &= ~IOT_PROG_SEL;        // SET PIN A GP I/O
  //P3OUT |= IOT_PROG_SEL;        // PULLUP RESISTOR
  P3DIR &= ~IOT_PROG_SEL;         // SET PIN TO INPUT
  //P3REN |= IOT_PROG_SEL;        // RESISTOR ENABLED

  //PIN 4
  switch (port_functionality){
   case USE_L_REVERSE:          // SET PIN AS PWM (TB0.3)
    P3SEL1 &= ~L_REVERSE;       // SET PIN AS PWM (TB0.3)
    P3SEL0 |= L_REVERSE;        // SET PIN AS PWM (TB0.3)
    //P3OUT &= ~L_REVERSE;      // SET PIN LOW
    P3DIR |= L_REVERSE;         // SET PIN TO OUTPUT
    break;

   case USE_SMCLK:              // use pin as SMCLK
    P3SEL1 |= SMCLK;            //
    P3DIR |= SMCLK;             //
    break;
   default: break;
  }

  //PIN 5
  P3SEL1 &= ~L_FORWARD;         // SET PIN AS PWM (TB0.4)
  P3SEL0 |= L_FORWARD;          // SET PIN AS PWM (TB0.4)
  //P3OUT &= ~L_FORWARD;        // SET PIN LOW
  P3DIR |= L_FORWARD;           // SET PIN TO OUTPUT

  //PIN 6
  P3SEL1 &= ~R_REVERSE;         // SET PIN AS PWM (TB0.5)
  P3SEL0 |= R_REVERSE;          // SET PIN AS PWM (TB0.5)
  //P3OUT &= ~R_REVERSE;        // SET PIN LOW
  P3DIR |= R_REVERSE;           // SET PIN TO OUTPUT

  //PIN 7
  P3SEL1 &= ~R_FORWARD;         // SET PIN AS PWM (TB0.6)
  P3SEL0 |= R_FORWARD;          // SET PIN AS PWM (TB0.6)
  //P3OUT &= ~R_FORWARD;        // SET PIN LOW
  P3DIR |= R_FORWARD;           // SET PIN TO OUTPUT

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

//************************************ PORT_4 *********************************
//------------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_4
void Init_Port_4(void){

  P4SEL1 = RESET;               // RESET PORT
  P4SEL0 = RESET;               // RESET PORT

  //PIN 0
  P4SEL1 |= SD_CS;              // ADC_SD_CS
  P4SEL0 |= SD_CS;              // ADC_SD_CS
```

```
  //PIN 1
  P4SEL1 &= ~IO_J4_31;          // SET PIN AS GP I/O
  P4SEL0 &= ~IO_J4_31;          // SET PIN A GP I/O
  //P4OUT |= IO_J4_31;          // PULLUP RESISTOR
  P4DIR &= ~IO_J4_31;           // SET PIN TO INPUT
  //P4REN |= IO_J4_31;          // RESISTOR ENABLED

  //PIN 2
  P4SEL1 &= ~IO_J4_32;          // SET PIN AS GP I/O
  P4SEL0 &= ~IO_J4_32;          // SET PIN A GP I/O
  //P4OUT |= IO_J4_32;          // PULLUP RESISTOR
  P4DIR &= ~IO_J4_32;           // SET PIN TO INPUT
  //P4REN |= IO_J4_32;          // RESISTOR ENABLED

  //PIN 3
  P4SEL1 &= ~IO_J4_33;          // SET PIN AS GP I/O
  P4SEL0 &= ~IO_J4_33;          // SET PIN A GP I/O
  //P4OUT |= IO_J4_33;          // PULLUP RESISTOR
  P4DIR &= ~IO_J4_33;           // SET PIN TO INPUT
  //P4REN |= IO_J4_33;          // RESISTOR ENABLED

  //PIN 4
  P4SEL1 &= ~UCB1_CS_LCD;       // SET PIN AS GP I/O
  P4SEL0 &= ~UCB1_CS_LCD;       // SET PIN A GP I/O
  P4OUT |= UCB1_CS_LCD;         // SET PIN LOW
  P4DIR |= UCB1_CS_LCD;         // SET PIN TO OUTPUT

  //PIN 5
  P4SEL1 &= ~P4_5;              // SET PIN AS GP I/O
  P4SEL0 &= ~P4_5;              // SET PIN A GP I/O
  //P4OUT |= P4_5;              // PULLUP RESISTOR
  P4DIR &= ~P4_5;               // SET PIN TO INPUT
  //P4REN |= P4_5;              // RESISTOR ENABLED

  //PIN 6
  P4SEL1 &= ~P4_6;              // SET PIN AS GP I/O
  P4SEL0 &= ~P4_6;              // SET PIN A GP I/O
  //P4OUT |= P4_6;              // PULLUP RESISTOR
  P4DIR &= ~P4_6;               // SET PIN TO INPUT
  //P4REN |= P4_6;              // RESISTOR ENABLED

  //PIN 7
  P4SEL1 &= ~IO_J3_29;          // SET PIN AS GP I/O
  P4SEL0 &= ~IO_J3_29;          // SET PIN A GP I/O
  //P4OUT |= IO_J3_29;          // PULLUP RESISTOR
  P4DIR &= ~IO_J3_29;           // SET PIN TO INPUT
  //P4REN |= IO_J3_29;          // RESISTOR ENABLED

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

//************************************ PORT_5 *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_5
void Init_Port_5(void){

  P5SEL1 = RESET;               // RESET PORT
```

```
  P5SEL0 = RESET;              // RESET PORT

  //PIN 0
  P5SEL1 &= ~SPI_UCB1SIMO;       // SPI_UCB1_SIMO
  P5SEL0 |= SPI_UCB1SIMO;       // SPI_UCB1_SIMO

  //PIN 1
  P5SEL1 &= ~SPI_UCB1SOMI;       // SPI_UCB1_SOMI
  P5SEL0 |= SPI_UCB1SOMI;       // SPI_UCB1_SOMI

  //PIN 2
  P5SEL1 &= ~SPI_UCB1CLK;       // SPI_UCB1_CLK
  P5SEL0 |= SPI_UCB1CLK;       // SPI_UCB1_CLK

  //PIN 3
  P5SEL1 &= ~RESET_LCD;          // SET PIN AS GP I/O
  P5SEL0 &= ~RESET_LCD;          // SET PIN A GP I/O
  P5OUT |= RESET_LCD;          // SET PIN HIGH
  P5DIR |= RESET_LCD;          // SET PIN TO OUTPUT

  //PIN 4
  P5SEL1 &= ~P5_4;             // SET PIN AS GP I/O
  P5SEL0 &= ~P5_4;             // SET PIN A GP I/O
  //P5OUT |= P5_4;             // PULLUP RESISTOR
  P5DIR &= ~P5_4;              // SET PIN TO INPUT
  //P5REN |= P5_4;             // RESISTOR ENABLED

  //PIN 5
  P5SEL1 &= ~BUTTON2;          // SET PIN AS GP I/O
  P5SEL0 &= ~BUTTON2;          // SET PIN A GP I/O
  P5OUT |= BUTTON2;            // PULLUP RESISTOR
  P5DIR &= ~BUTTON2;           // SET PIN TO INPUT
  P5REN |= BUTTON2;            // RESISTOR ENABLED
  P5IES |= BUTTON2;            // HI/ LO EDGE INTERRUPT
  P5IFG &= ~BUTTON2;           // CLEAR IFG
  P5IE  |= BUTTON2;            // INTERRUPT ENABLED

  //PIN 6
  P5SEL1 &= ~BUTTON1;          // SET PIN AS GP I/O
  P5SEL0 &= ~BUTTON1;          // SET PIN A GP I/O
  P5OUT |= BUTTON1;            // PULLUP RESISTOR
  P5DIR &= ~BUTTON1;           // SET PIN TO INPUT
  P5REN |= BUTTON1;            // RESISTOR ENABLED
  P5IES |= BUTTON1;            // HI/ LO EDGE INTERRUPT
  P5IFG &= ~BUTTON1;           // CLEAR IFG
  P5IE  |= BUTTON1;            // INTERRUPT ENABLED

  //PIN 7
  P5SEL1 |= LCD_BACKLITE;       // SET PIN AS GP I/O
  P5SEL0 &= ~LCD_BACKLITE;       // SET PIN A GP I/O
  //P5OUT |= LCD_BACKLITE;       // SET PIN HIGH
  P5DIR |= LCD_BACKLITE;       // SET PIN TO OUTPUT

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

//************************************ PORT_6 *********************************
```

```
//-------------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_6
void Init_Port_6(void){

  P6SEL1 = RESET;          // RESET PORT
  P6SEL0 = RESET;          // RESET PORT

  //PIN 0
  P6SEL1 &= ~UCA3TXD;      // UTXD_UCA3TXD
  P6SEL0 |= UCA3TXD;       // UTXD_UCA3TXD


  //PIN 1
  P6SEL1 &= ~UCA3RXD;      // URXD_UAC3RXD
  P6SEL0 |= UCA3RXD;       // URXD_UAC3RXD


  //PIN 2
  P6SEL1 &= ~IO_J1_5;      // SET PIN AS GP I/O
  P6SEL0 &= ~IO_J1_5;      // SET PIN A GP I/O
  //P6OUT |= IO_J1_5;      // PULLUP RESISTOR
  P6DIR &= ~IO_J1_5;       // SET PIN TO INPUT
  //P6REN |= IO_J1_5;      // RESISTOR ENABLED

  //PIN 3
  P6SEL1 &= ~MAG_INT;      // SET PIN AS GP I/O
  P6SEL0 &= ~MAG_INT;      // SET PIN A GP I/O
  //P6OUT |= MAG_INT;      // PULLUP RESISTOR
  P6DIR &= ~MAG_INT;       // SET PIN TO INPUT
  //P6REN |= MAG_INT;      // RESISTOR ENABLED

  //PIN 4
  P6SEL1 &= ~P6_4;         // SET PIN AS GP I/O
  P6SEL0 &= ~P6_4;         // SET PIN A GP I/O
  //P6OUT |= P6_4;         // PULLUP RESISTOR
  P6DIR &= ~P6_4;          // SET PIN TO INPUT
  //P6REN |= P6_4;         // RESISTOR ENABLED

  //PIN 5
  P6SEL1 &= ~P6_5;         // SET PIN AS GP I/O
  P6SEL0 &= ~P6_5;         // SET PIN A GP I/O
  //P6OUT |= P6_5;          // PULLUP RESISTOR
  P6DIR &= ~P6_5;          // SET PIN TO INPUT
  //P6REN |= P6_5;         // RESISTOR ENABLED

  //PIN 6
  P6SEL1 &= ~P6_6;         // SET PIN AS GP I/O
  P6SEL0 &= ~P6_6;         // SET PIN A GP I/O
  //P6OUT |= P6_6;         // PULLUP RESISTOR
  P6DIR &= ~P6_6;          // SET PIN TO INPUT
  //P6REN |= P6_6;         // RESISTOR ENABLED

  //PIN 7
  P6SEL1 &= ~P6_7;         // SET PIN AS GP I/O
  P6SEL0 &= ~P6_7;         // SET PIN A GP I/O
  //P6OUT |= P6_7;         // PULLUP RESISTOR
  P6DIR &= ~P6_7;          // SET PIN TO INPUT
```

```
  //P6REN |= P6_7;          // RESISTOR ENABLED

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

//************************************ PORT_7 *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_7
void Init_Port_7(void){

  P7SEL1 = RESET;             // RESET PORT
  P7SEL0 = RESET;             // RESET PORT

  //PIN 0
  P7SEL1 &= ~I2CSDA_J1_10;       // I2CSDA_UCB2SDA
  P7SEL0 |= I2CSDA_J1_10;       // I2CSDA_UCB2SDA

  //PIN 1
  P7SEL1 &= ~I2CSCL_J1_9;       // I2CSCL_USB2SCL
  P7SEL0 |= I2CSCL_J1_9;       // I2CSCL_USB2SCL

  //PIN 2
  P7SEL1 &= ~SD_DETECT;          // SET PIN AS GP I/O
  P7SEL0 &= ~SD_DETECT;          // SET PIN A GP I/O
  //P7OUT |= SD_DETECT;          // PULLUP RESISTOR
  P7DIR &= ~SD_DETECT;          // SET PIN TO INPUT
  //P7REN |= SD_DETECT;          // RESISTOR ENABLED

  //PIN 3
  P7SEL1 &= ~CAPTURE_J4_36;       // SET PIN AS GP I/O
  P7SEL0 &= ~CAPTURE_J4_36;       // SET PIN A GP I/O
  //P7OUT |= CAPTURE_J4_36;       // PULLUP RESISTOR
  P7DIR &= ~CAPTURE_J4_36;       // SET PIN TO INPUT
  //P7REN |= CAPTURE_J4_36;       // RESISTOR ENABLED

  //PIN 4
  P7SEL1 &= ~P7_4;          // SET PIN AS GP I/O
  P7SEL0 &= ~P7_4;          // SET PIN A GP I/O
  //P7OUT |= P7_4;          // PULLUP RESISTOR
  P7DIR &= ~P7_4;          // SET PIN TO INPUT
  //P7REN |= P7_4;          // RESISTOR ENABLED

  //PIN 5
  P7SEL1 &= ~P7_5;          // SET PIN AS GP I/O
  P7SEL0 &= ~P7_5;          // SET PIN A GP I/O
  //P7OUT |= P7_5;          // PULLUP RESISTOR
  P7DIR &= ~P7_5;          // SET PIN TO INPUT
  //P7REN |= P7_5;          // RESISTOR ENABLED

  //PIN 6
  P7SEL1 &= ~P7_6;          // SET PIN AS GP I/O
  P7SEL0 &= ~P7_6;          // SET PIN A GP I/O
  //P7OUT |= P7_6;          // PULLUP RESISTOR
  P7DIR &= ~P7_6;          // SET PIN TO INPUT
  //P7REN |= P7_6;          // RESISTOR ENABLED

  //PIN 7
```

```
  P7SEL1 &= ~P7_7;          // SET PIN AS GP I/O
  P7SEL0 &= ~P7_7;          // SET PIN A GP I/O
  //P7OUT |= P7_7;          // PULLUP RESISTOR
  P7DIR &= ~P7_7;           // SET PIN TO INPUT
  //P7REN |= P7_7;          // RESISTOR ENABLED

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

//************************************ PORT_8 *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_8
void Init_Port_8(void){

  P8SEL1 = RESET;           // RESET PORT
  P8SEL0 = RESET;           // RESET PORT

  //PIN 0
  P8SEL1 &= ~IR_LED;        // SET PIN AS GP I/O
  P8SEL0 &= ~IR_LED;        // SET PIN A GP I/O
  P8OUT &= ~IR_LED;         // SET PIN low
  P8DIR |= IR_LED;          // SET PIN TO OUTPUT

  //PIN 1
  P8SEL1 &= ~OPT_INT;       // SET PIN AS GP I/O
  P8SEL0 &= ~OPT_INT;       // SET PIN A GP I/O
  //P8OUT |= OPT_INT;       // PULLUP RESISTOR
  P8DIR &= ~OPT_INT;        // SET PIN TO INPUT
  //P8REN |= OPT_INT;       // RESISTOR ENABLED

  //PIN 2
  P8SEL1 &= ~TMP_INT;       // SET PIN AS GP I/O
  P8SEL0 &= ~TMP_INT;       // SET PIN A GP I/O
  //P8OUT |= TMP_INT;       // PULLUP RESISTOR
  P8DIR &= ~TMP_INT;        // SET PIN TO INPUT
  //P8REN |= TMP_INT;       // RESISTOR ENABLED

  //PIN 3
  P8SEL1 &= ~INT2;          // SET PIN AS GP I/O
  P8SEL0 &= ~INT2;          // SET PIN A GP I/O
  //P8OUT |= INT2;          // PULLUP RESISTOR
  P8DIR &= ~INT2;           // SET PIN TO INPUT
  //P8REN |= INT2;          // RESISTOR ENABLED

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

//************************************ PORT_J *********************************
//-------------------------------------------------------------------------
// THIS FUNCTION INITIALIZES PORT_J
void Init_Port_J(void){

  PJSEL1 = RESET;           // RESET PORT
  PJSEL0 = RESET;           // RESET PORT

  // PIN_0
  PJSEL1 &= ~PJ_0;          // SET PIN AS GP I/O
```

```
  PJSEL0 &= ~PJ_0;          // SET PIN A GP I/O
  //PJOUT |= PJ_0;          // PULLUP RESISTOR
  PJDIR &= ~PJ_0;           // SET PIN TO INPUT
  //PJREN |= PJ_0;          // RESISTOR ENABLED

  // PIN_1
  PJSEL1 &= ~PJ_1;          // SET PIN AS GP I/O
  PJSEL0 &= ~PJ_1;          // SET PIN A GP I/O
  //PJOUT |= PJ_1;          // PULLUP RESISTOR
  PJDIR &= ~PJ_1;           // SET PIN TO INPUT
  //PJREN |= PJ_1;          // RESISTOR ENABLED

  // PIN_2
  PJSEL1 &= ~PJ_2;          // SET PIN AS GP I/O
  PJSEL0 &= ~PJ_2;          // SET PIN A GP I/O
  //PJOUT |= PJ_2;          // PULLUP RESISTOR
  PJDIR &= ~PJ_2;           // SET PIN TO INPUT
  //PJREN |= PJ_2;          // RESISTOR ENABLED

  // PIN_3
  PJSEL1 &= ~PJ_3;          // SET PIN AS GP I/O
  PJSEL0 &= ~PJ_3;          // SET PIN A GP I/O
  //PJOUT |= PJ_3;          // PULLUP RESISTOR
  PJDIR &= ~PJ_3;           // SET PIN TO INPUT
  //PJREN |= PJ_3;          // RESISTOR ENABLED

  // PIN_4
  PJSEL1 &= ~LFXIN;         // LOW FREQUENCY CRYSTAL IN
  PJSEL0 |= LFXIN;          // LOW FREQUENCY CRYSTAL IN

  // PIN_5
  PJSEL1 &= ~LFXOUT;        // LOW FREQUENCY CRYSTAL OUT
  PJSEL0 |= LFXOUT;         // LOW FREQUENCY CRYSTAL OUT

  // PIN_6
  PJSEL1 &= ~HFXIN;         // HIGH FREQUENCY CRYSTAL IN
  PJSEL0 |= HFXIN;          // HIGH FREQUENCY CRYSTAL IN

  // PIN_7
  PJSEL1 &= ~HFXOUT;        // HIGH FREQUENCY CRYSTAL OUT
  PJSEL0 |= HFXOUT;         // HIGH FREQUENCY CRYSTAL OUT

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

//********************************* INIT PORTS *********************************
//-------------------------------------------------------------------------------
// this function calls all functions needed to initialize all the pins
// on each port
void Init_Ports(char port_3_functionality){
  Init_Port_1();
  Init_Port_2();
  Init_Port_3(port_3_functionality);
  Init_Port_4();
  Init_Port_5();
  Init_Port_6();
  Init_Port_7();
```

```
  Init_Port_8();
  Init_Port_J();

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

9.6 Serial.c

```
 //=====================================================================
//
// File Name : serial.c
//
// Description: This file contains serial communication interrupts as well as
// usefule serial functions.
//
// Author: Timothy Davis
// Date: Oct 2018
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=====================================================================

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

extern volatile char IOT_Char_Rx[SMALL_RING_SIZE];
extern volatile char IOT_Char_Tx[LARGE_RING_SIZE];
extern volatile unsigned int iot_rx_ring_wr3;
extern volatile unsigned int iot_rx_ring_rd3;
extern volatile unsigned int iot_tx_ring_wr3;
extern volatile unsigned int iot_tx_ring_rd3;
extern volatile unsigned int UCA3_index;

extern volatile char PC_Char_Rx[SMALL_RING_SIZE];
extern volatile char PC_Char_Tx[LARGE_RING_SIZE];
extern volatile unsigned int pc_rx_ring_wr0;
extern volatile unsigned int pc_rx_ring_rd0;
extern volatile unsigned int pc_tx_ring_wr0;
extern volatile unsigned int pc_tx_ring_rd0;
extern volatile unsigned int UCA0_index;

extern volatile unsigned int test_command[NUMTEN];
extern volatile unsigned int item_recieved;

#pragma vector=USCI_A3_VECTOR
__interrupt void USCI_A3_ISR(void){
unsigned int temp;
   switch(__even_in_range(UCA3IV,ENDOFVECTORS)){
   case VECT0: // Vector 0 - no interrupt
    break;
   case VECT2: // Vector 2 - RXIFG
     temp = iot_rx_ring_wr3;
     IOT_Char_Rx[temp] = UCA3RXBUF; // RX -> USB_Char_Rx character
```

```
    if (++iot_rx_ring_wr3 >= (SMALL_RING_SIZE)){
      iot_rx_ring_wr3 = BEGINNING; // Circular buffer back to beginning
    }
   break;
  case VECT4: // Vector 4 – TXIFG
   switch(UCA3_index++){
    case MOVSTE0: //
    case MOVSTE1: //
    case MOVSTE2: //
    case MOVSTE3: //
    case MOVSTE4: //
    case MOVSTE5: //
    case MOVSTE6: //
    case MOVSTE7: //
    case MOVSTE8: //
      UCA3TXBUF = test_command[UCA3_index];
      break;
    case MOVSTE9: //
      UCA3TXBUF = SENDTOTX13;
    break;
    case MOVSTE10: // Vector 0 - no interrupt
      UCA3TXBUF = SENDTOTX10;
      break;
    default:
      UCA3IE &= ~UCTXIE; // Disable TX interrupt
      break;
   }
   break;
  default: break;
  }
}

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void){
  unsigned int temp2;
  switch(__even_in_range(UCA0IV,ENDOFVECTORS)){
  case VECT0: // Vector 0 - no interrupt
   break;
  case VECT2: // Vector 2 - RXIFG
   temp2 = UCA0RXBUF;
   UCA0TXBUF = temp2;
   temp2 = pc_rx_ring_wr0;
   PC_Char_Rx[temp2] = UCA0RXBUF; // RX -> USB_Char_Rx character
   if (++pc_rx_ring_wr0 >= (SMALL_RING_SIZE)){
     pc_rx_ring_wr0 = BEGINNING; // Circular buffer back to beginning
   }
   if(UCA0RXBUF == SENDTOTX13) item_recieved = ITEMRECIEVED;
   break;
  case VECT4: // Vector 4 – TXIFG
   switch(UCA0_index++){
    case MOVSTE0: //
    case MOVSTE1: //
    case MOVSTE2: //
    case MOVSTE3: //
    case MOVSTE4: //
    case MOVSTE5: //
    case MOVSTE6: //
```

```
      case MOVSTE7: //
      case MOVSTE8: //
        UCA0TXBUF = test_command[UCA0_index];
        break;
      case MOVSTE9: //
        UCA0TXBUF = SENDTOTX13;
      break;
      case MOVSTE10: // Vector 0 - no interrupt
        UCA0TXBUF = SENDTOTX10;
        break;
      default:
        UCA0IE &= ~UCTXIE; // Disable TX interrupt
        break;
    }
    break;
   default: break;
   }
}



void out_character(char character){
  while (!(UCA3IFG & UCTXIFG)); // USCI_A3 TX buffer ready?
    UCA3TXBUF = character;
}

void out_characterA0(char character){
  while (!(UCA0IFG & UCTXIFG)); // USCI_A0 TX buffer ready?
    UCA0TXBUF = character;

}

void setbaudrate(char baudrate){
  if(baudrate == BAUD1){
     UCA3BRW = BRW;
     UCA3MCTLW = BAUDRATECONCAT;  //460800  which is primary
     UCA3CTL1 &= ~UCSWRST; // Release from reset
     UCA3IE |= UCRXIE; // Enable RX interrupt
  }
  if(baudrate == BAUD2){
     UCA3BRW = BRW2;
     UCA3MCTLW = BAUDRATECONCAT2;   //115200 which is alternate
     UCA3CTL1 &= ~UCSWRST; // Release from reset
     UCA3IE |= UCRXIE; // Enable RX interrupt
  }
}
```