Auto-Drone Extreme 2015 Hyper Lightning Edition

Driving into the future

Team 5

Revision	Description
1A	Initial revision. Added scope and power system information.
4A	Initial revision for Project 4. Changed the scope for wording. Changed hardware to reflect LCD addition. Added code and explanation to the software section. Added information to the flow chart, software listing, and conclusion. – Tyler Cheek
4B	Further revisions for Project 4. Added more subsystem blocks. Also reworded the conclusion.
5A	Added more prudent diagrams for all sections where they would be helpful. Updated the text sections for all subsections of the Overview (3) block diagrams and Hardware (4). Updated the project testing descriptions (5). – Tyler Cheek
6A	Serial communication added to test process and software listing added- Samuel Eddy

Team Members
Jason Zicherman
Samuel Eddy
Tyler Cheek
George Puzo

Originator Tyler Cheek
Checked: | Released:
Filename: Project Writeup - 6.docx

Table of Contents

1. Scope

The Auto-Drone Extreme 2015 Hyper Lightning Edition is an advanced, revolutionary children's toy designed to entertain kids from ages 4 and up. It operates in 3 different modes: remote controlled, GPS navigation, and black line detection. Each mode offers a fresh experience. Remote controlled mode is the traditional RC experience in which the child directly operates the vehicle. In this mode, the child can navigate this advanced vehicle at extreme speeds. In GPS navigation mode, the car will navigate a predefined GPS course created by the child using a simple, child friendly user interface. In black line mode, the car will autonomously follow a black line course set up to the child's liking. The Auto-Drone Extreme 2015 Hyper Lightning Edition will revolutionize the children's RC industry by offering modes more advanced than anything currently offered and will drive your kids right into the future.

2. Abbreviations

ADE – Auto Drone Extreme

LCD – Liquid Crystal Display

USB – Universal Serial Bus

FRAM - Ferroelectric Random Access Memory

IR LED – Infrared Light Emitting Diode

3. Overview

The ADE is built around a 4 AA battery-driven power supply, a control board, the MSP430 FRAM board, the bi-directional motor bridge, and an LCD display module. See Figure 1 below.

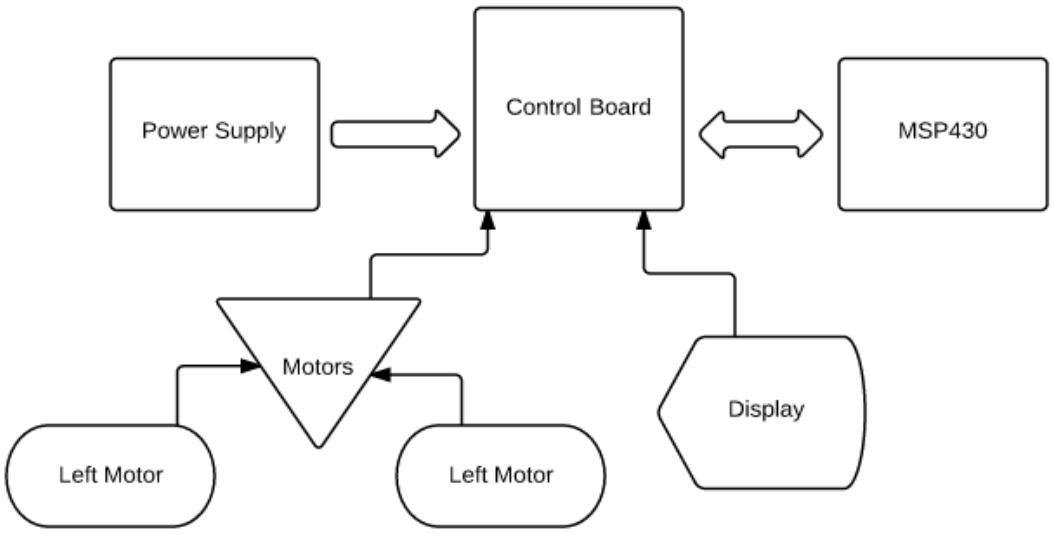


Figure 1 Overview

3.1. Power

The Power system consists of 4 AA batteries, the battery pack, the control board, which regulates power to the MSP430, a switch to turn the board on or off, a 3.3V circuit, and a 1.8V circuit. See Figure 2 below.

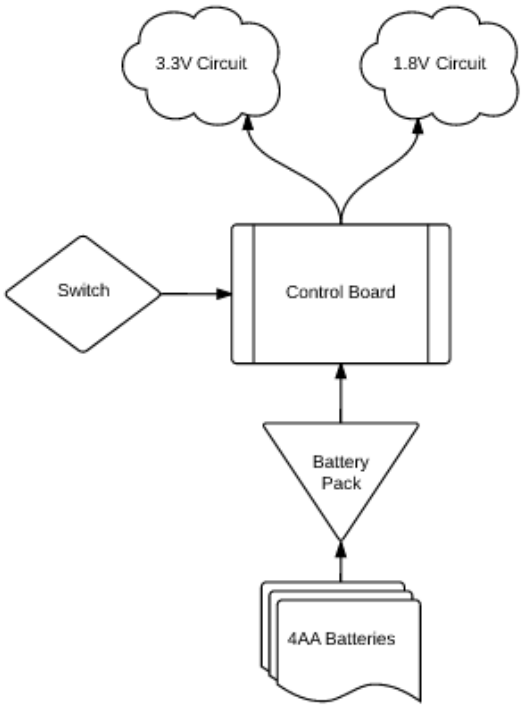


Figure 2 Power System Blocks

3.2. Motors

The two motors are connected to the H-bridge through J9 and J4. Each allow the motors to be provided a positive or negative voltage to drive the motors forward or backward, respectively. See Figure 3 below.

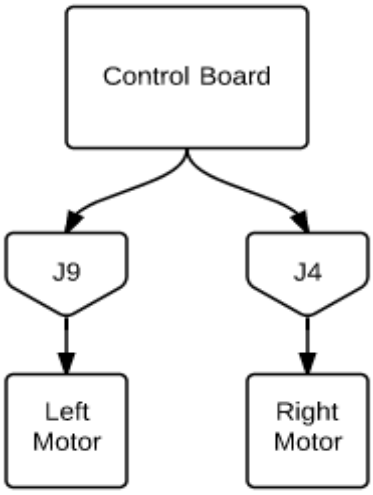


Figure 3 Motor Blocks

3.3. MSP430

The MSP430 FRAM board includes a USB port for software manipulation and debugging, 8 LEDs for itemized user feedback, external jumpers for control board merging, and two switches for asynchronous inputs. See Figure 4 below.

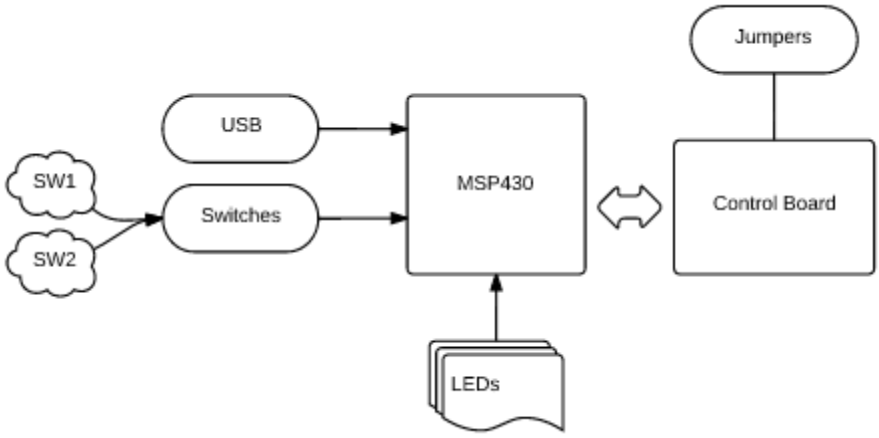


Figure 4 MSP430 Blocks

3.4. User Interface

The user interface for the ADE is through both the control board and the MSP430 with the LCD, the thumbwheel, the LEDs, and the switches.

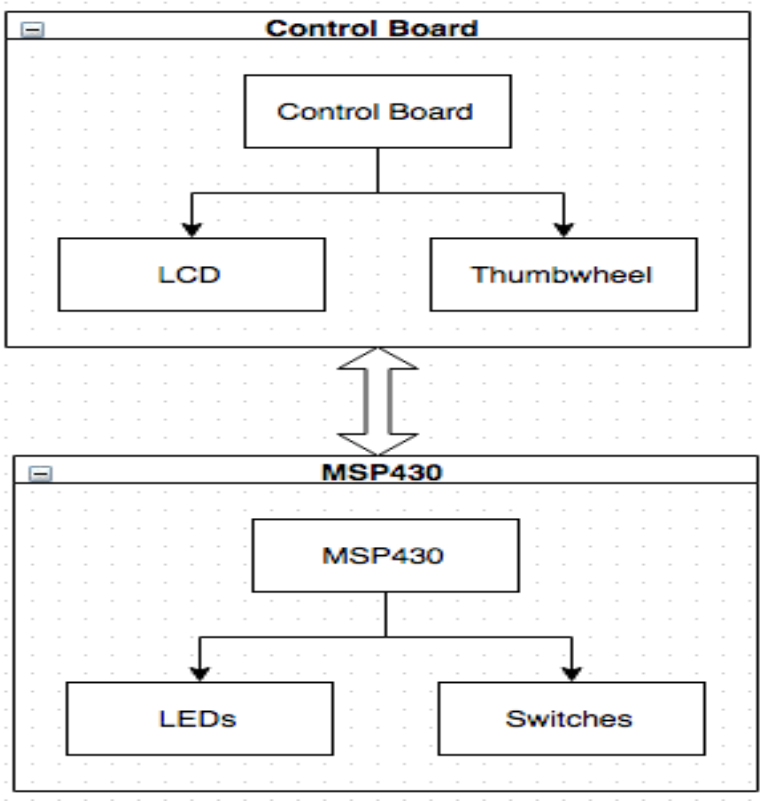


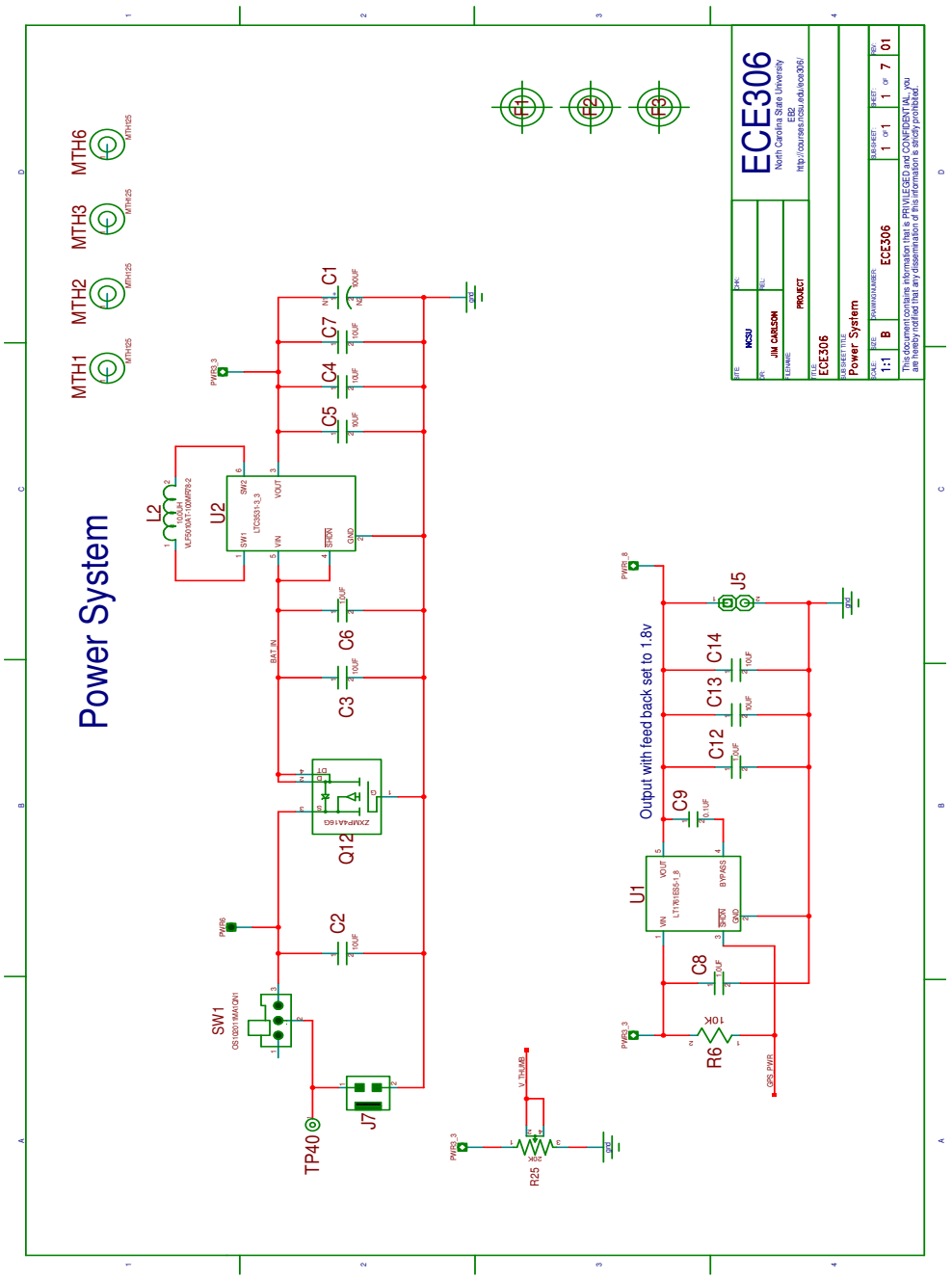
Figure 5 User Interface Blocks

4. Hardware

The ADE is built around a Texas Instruments MSP-430FR5739 Experimenters Board.

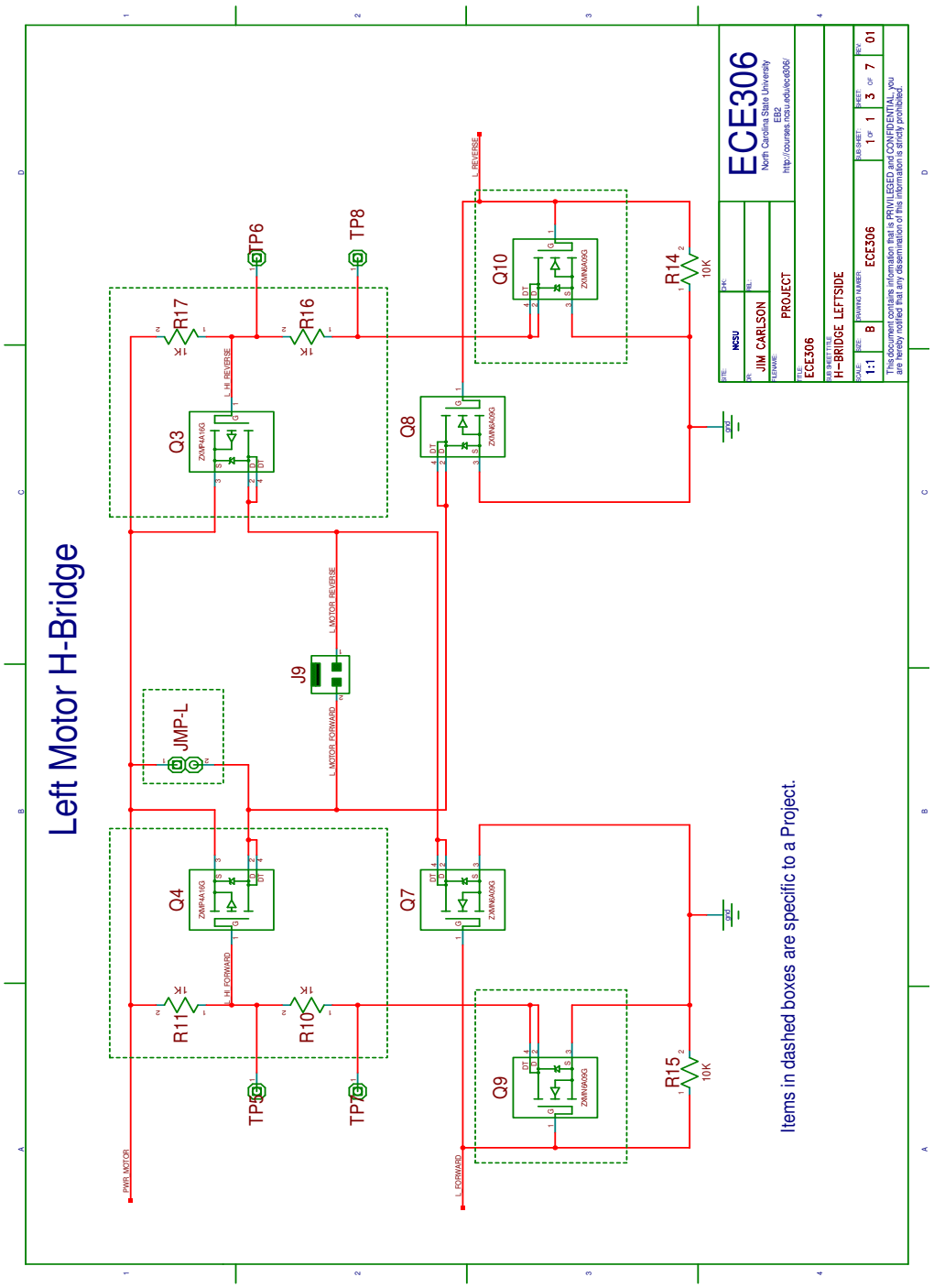
4.1. Power

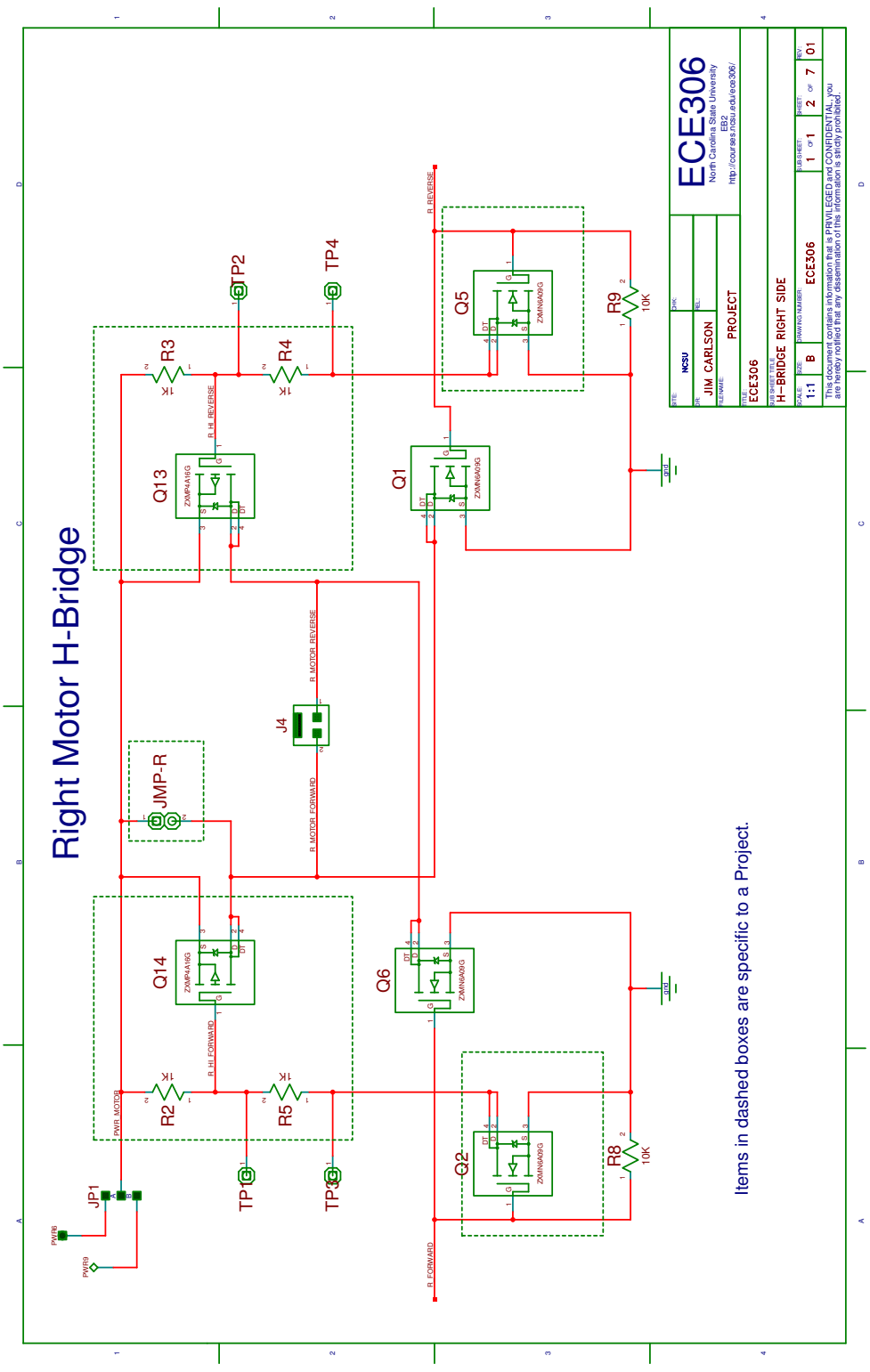
We designed the power system to maximize battery life and minimize wasted power. The buck/boost circuit that powers the MSP430 is up to 90% efficient, wastes as little power as possible, and provides short circuit protection. The ADE also has a switch installed on the control board to minimize battery drain when the car is not in use. The car can also be powered by USB for use while batteries charge.



4.2. Motors

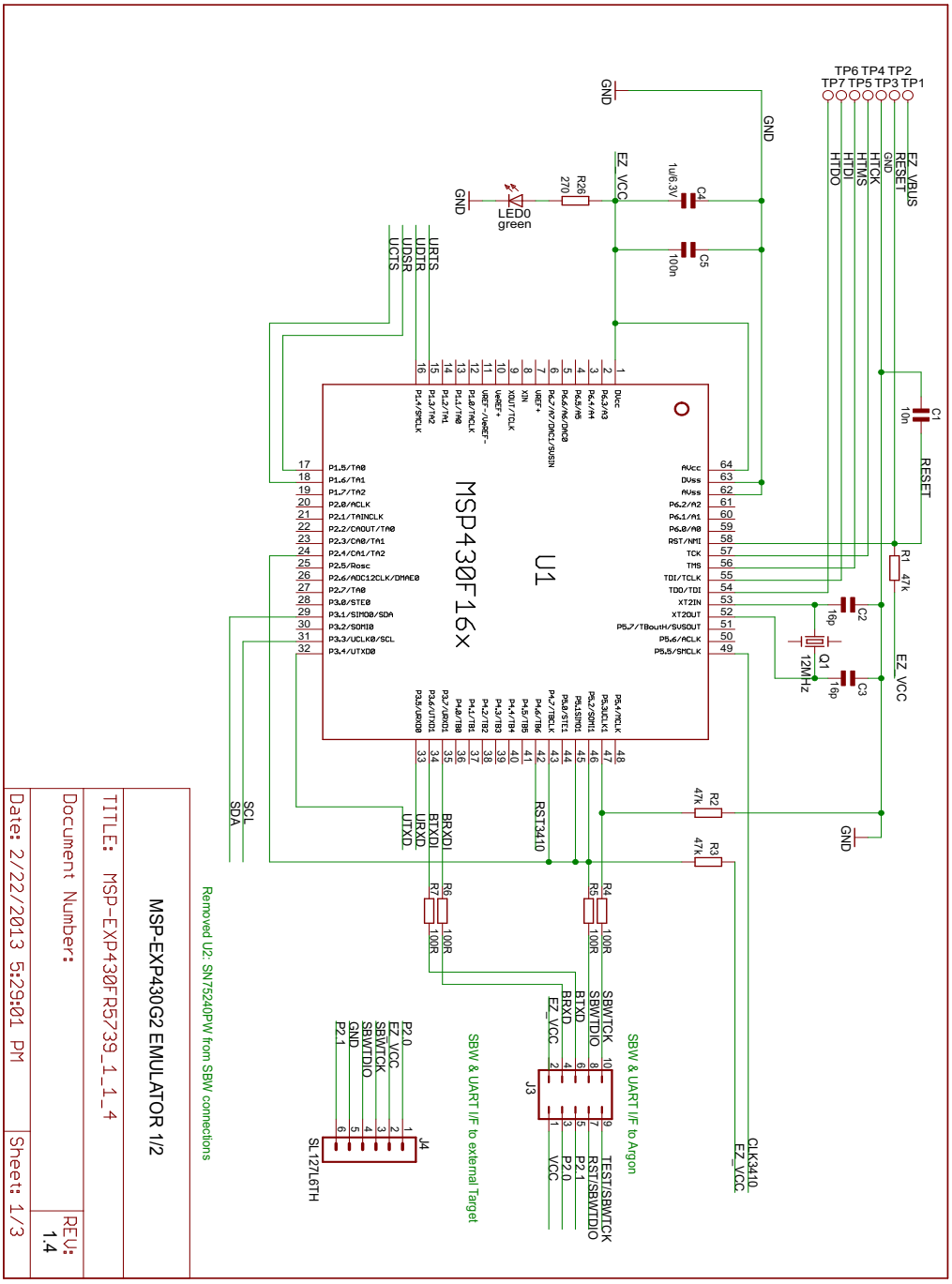
The ADE has two motors that both retain the capability to provide forward and reverse directionality. This is accomplished by the H-bridge composition of J4 and J9, two modules that each provide the ability to drive a positive (forward) or negative (reverse) voltage to the motor system. J4 and J9 separate one motor from the other, allowing more creative movements through targeted motor direction manipulation.

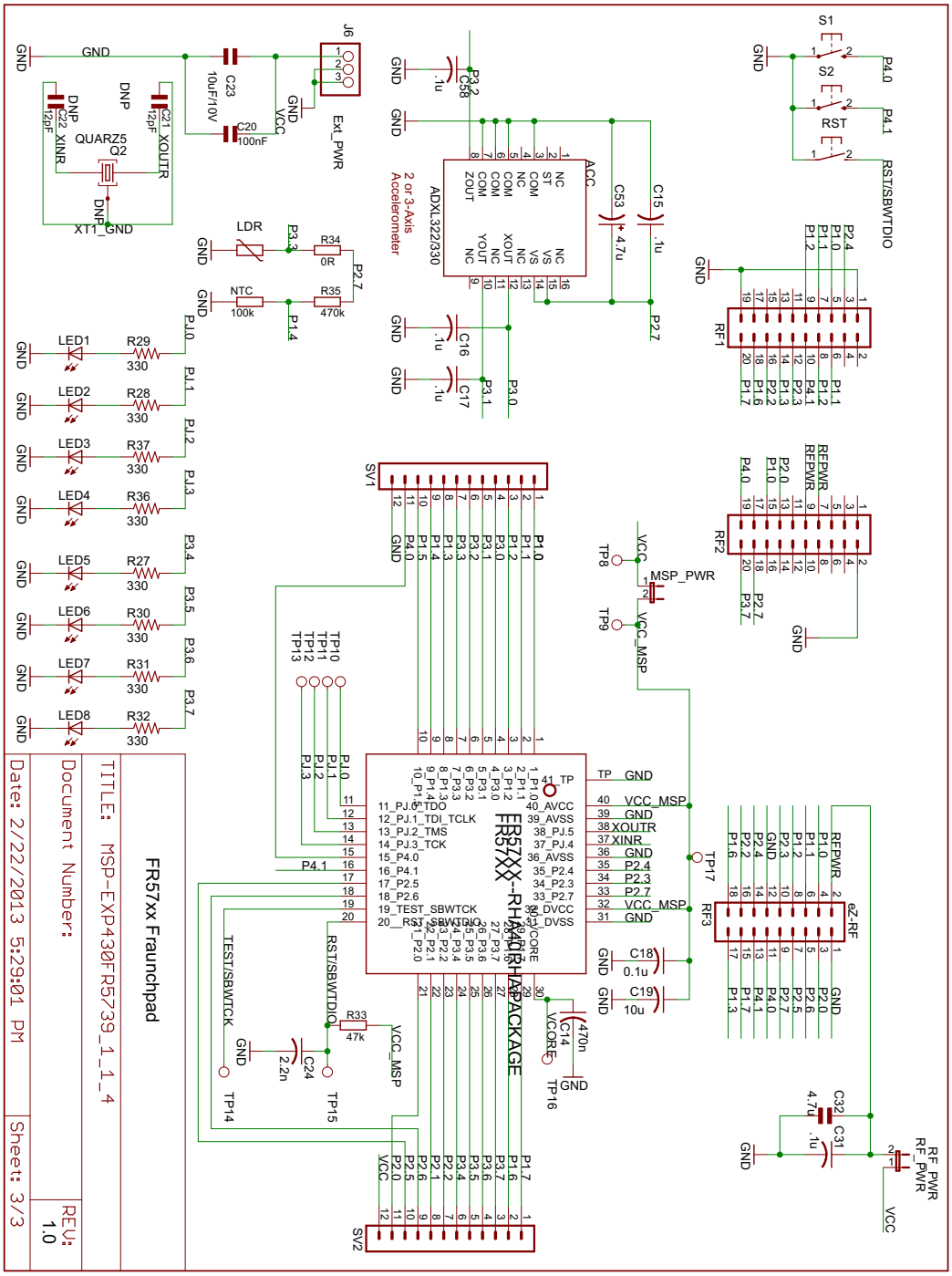


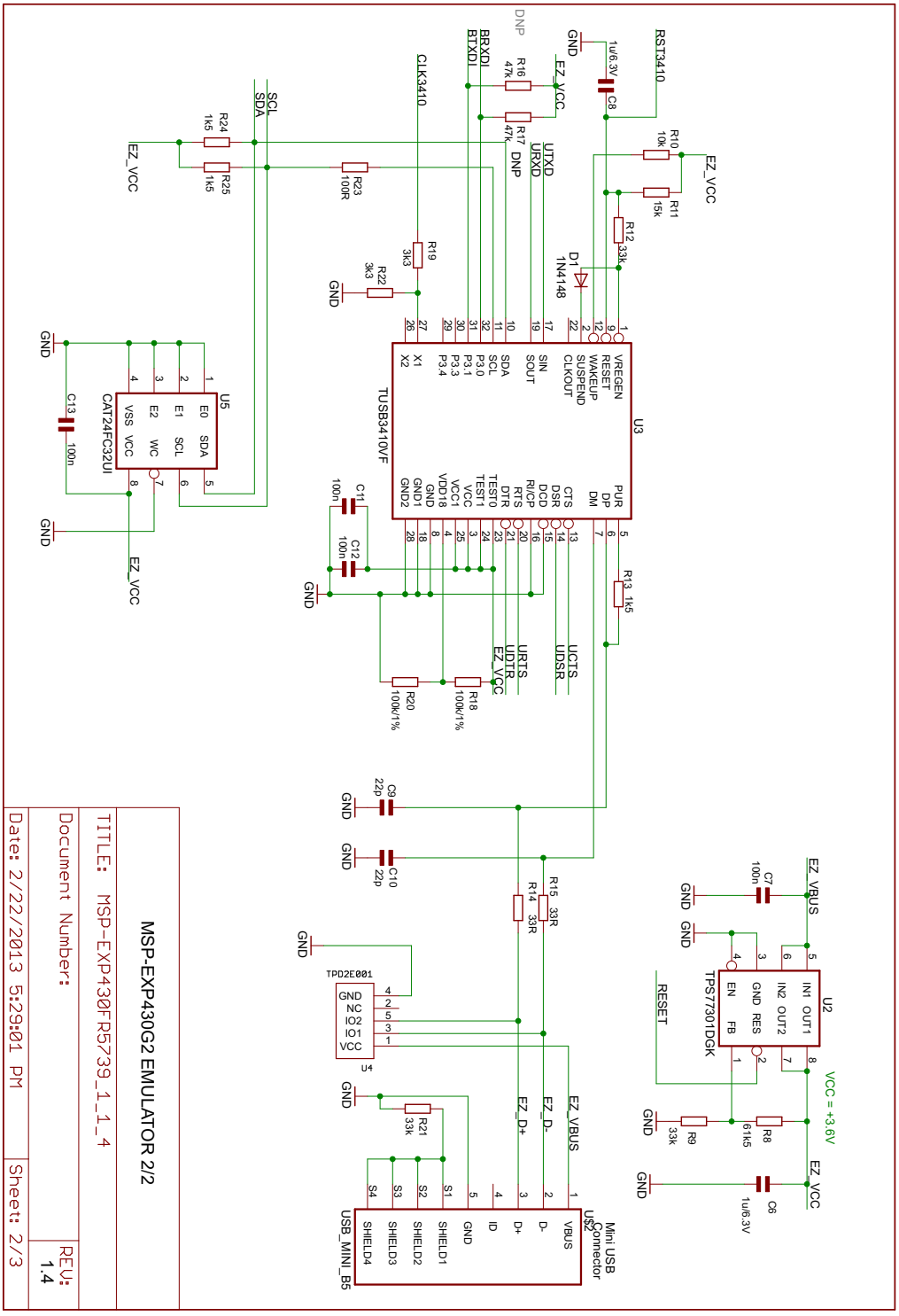

Right Motor H-Bridge
ECE306
North Carolina State University
ECE306
http://courses.ncsu.edu/ece306/
PROJECT
ECE306
H-BRIDGE RIGHT SIDE
JIM CARLSON
NCSU
B
1:1
1 of 1
2 of 7
01
This document contains information that is PRIVILEGED and CONFIDENTIAL; you are hereby notified that any dissemination of this information is strictly prohibited.
Items in dashed boxes are specific to a Project.
R_REVERSE
R_FORWARD
R_HI_REVERSE
R_HI_FORWARD
R_MOTOR_REVERSE
R_MOTOR_FORWARD
PWR_MOTOR
R2
R3
R4
R5
R8
R9
10K
1K
Q1
Q2
Q5
Q6
Q13
Q14
ZXMP6A16G
ZXMN6A09G
J4
JMP-R
JP1
TP1
TP2
TP3
TP4
PWR8
PWR9
gnd

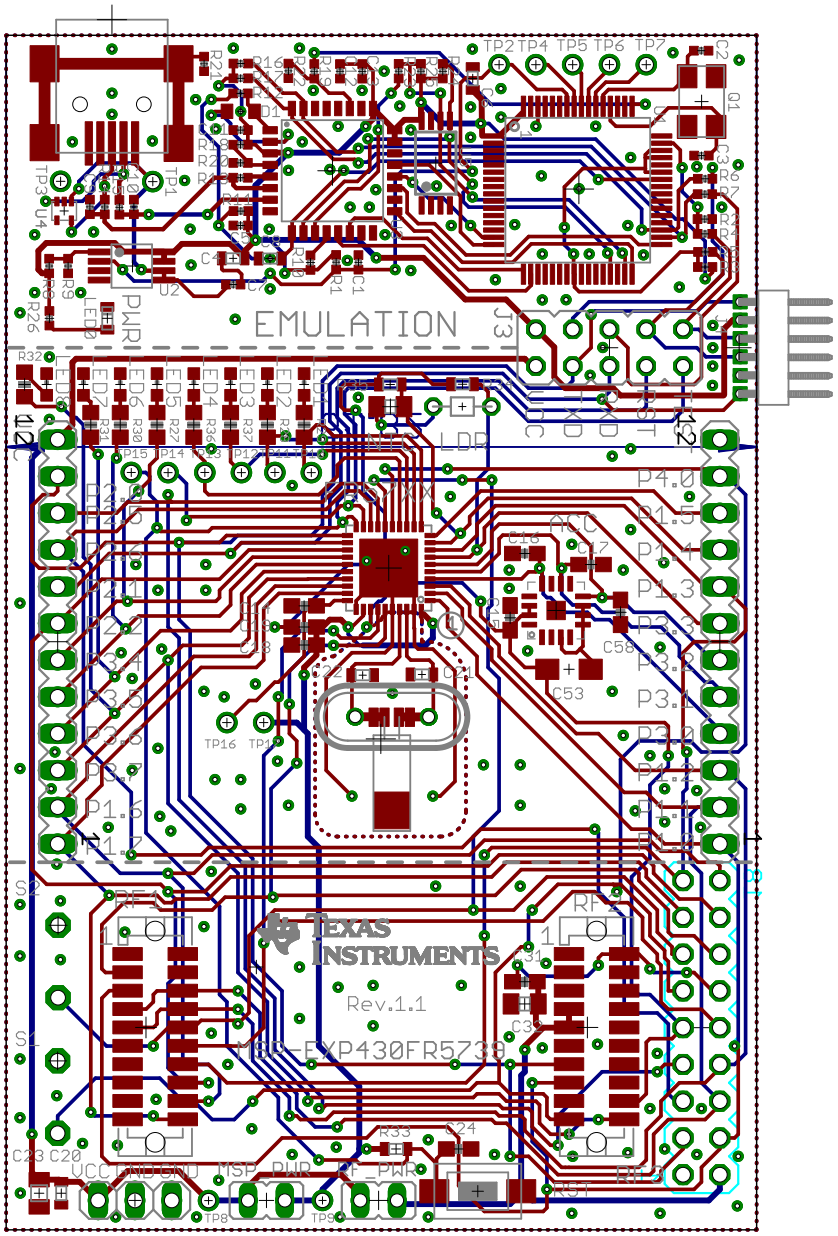
4.3. MSP430

The MSP430 Experimenters Board is driven by the highly-efficient buck-boost power system as described above. The board includes 8 independent LEDs that can be manipulated through software and two switches that can actuate unique, asynchronous inputs.




FR57xx Fraunchpad
TITLE: MSP-EXP430FR5739_1_1_4
Document Number:
Date: 2/22/2013 5:29:01 PM
Sheet: 3/3
REV: 1.0
FR57XX-RHA40RHA PACKAGE
ADXL322/330
2 or 3-Axis Accelerometer
QUARZ5
Ext_PWR
LDR
NTC
MSP_PWR
RST/SBWTDIO
TEST/SBWTCK
RF_PWR

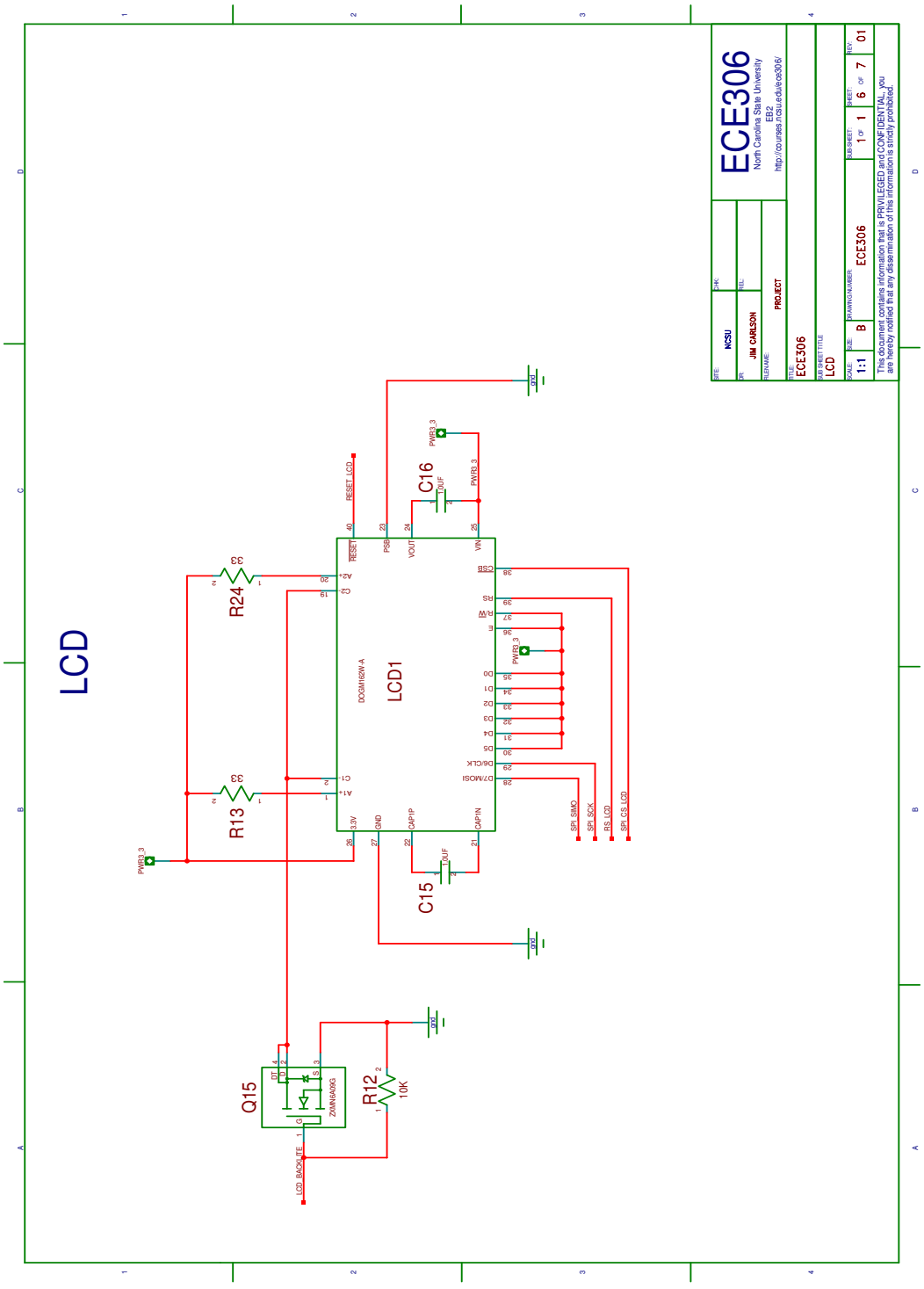

MSP-EXP430G2 EMULATOR 2/2
TITLE: MSP-EXP430FR5739_1_1_4
Document Number:
Date: 2/22/2013 5:29:01 PM
Sheet: 2/3
REV: 1.4
TUSB3410V/F
U3
CAT24FC32UI
U5
TPS77301DGK
U2
TPD2E001
U4
USB_MINI_B5
Mini USB Connector
U$2
RST3410
CLK3410
BRXDI
BTXDI
UTXD
URXD
URTS
UDTR
UCTS
UDSR
EZ_VCC
EZ_VBUS
EZ_D-
EZ_D+
SCL
SDA
RESET
VCC = +3.6V
GND
DNP



Date:	Document Number:	Rev:	Sheet:
04/15/2015	0-0000-000-0000-06	**6A**	**12 of 57**

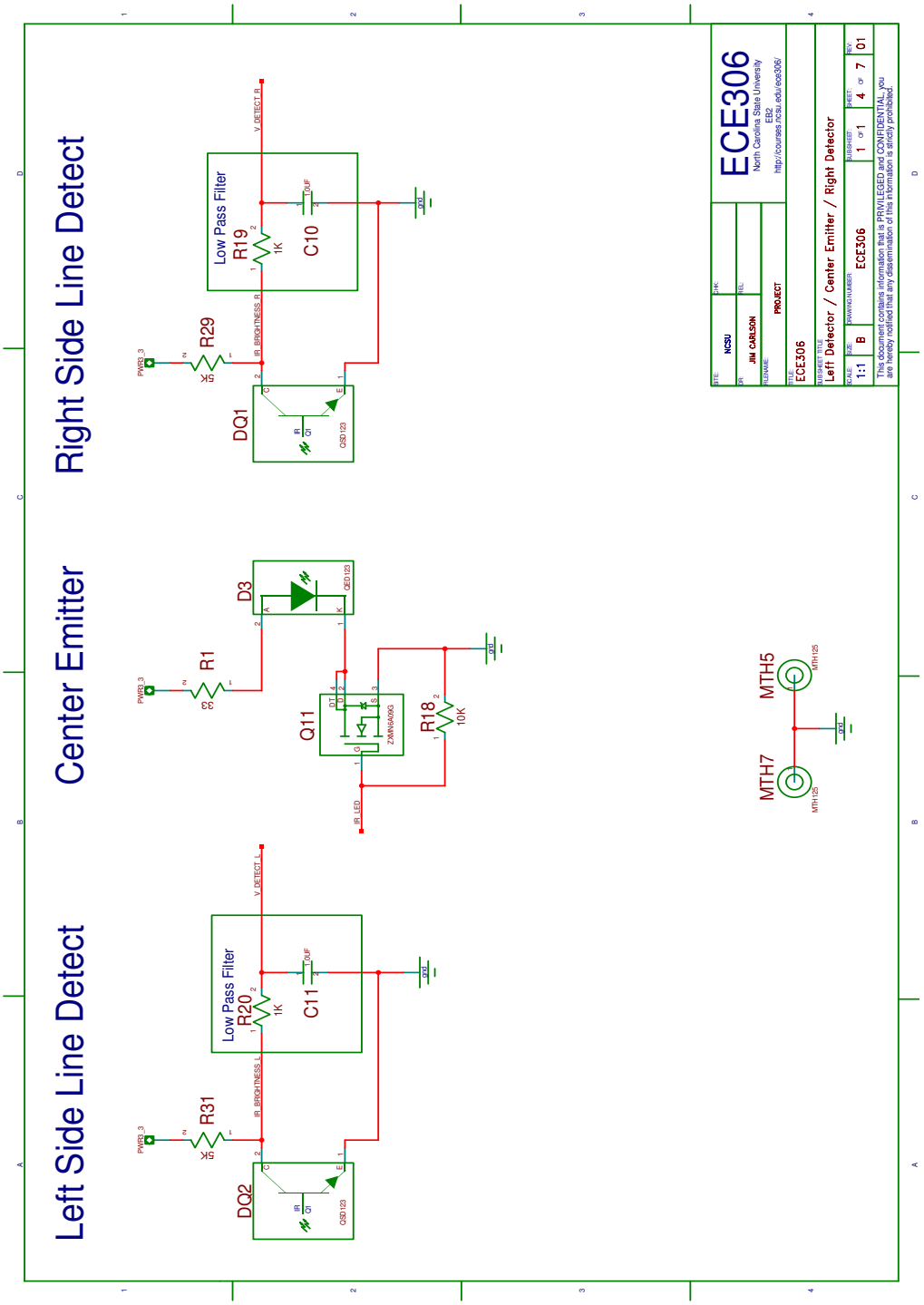
4.4. LCD Display

The ADE can display its power is on and its current mode through the LCD display. The display will light up and display its mode on startup.



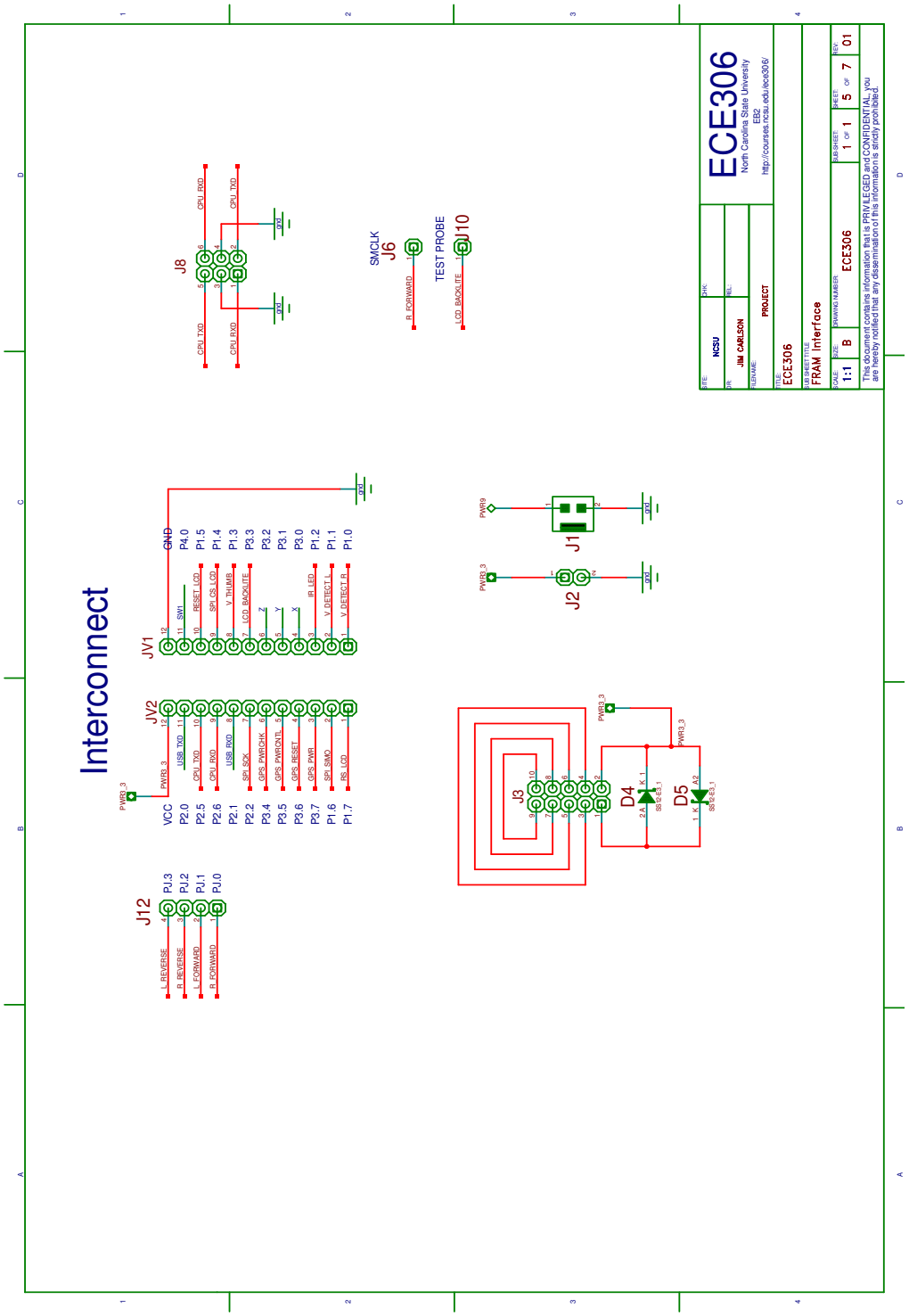
4.5. Diodes

One emitting diode and two detecting diodes are attached to the ADE and can be used to detect changes in light. For example, the ADE can be programmed to turn around or follow a black electrical tape line based off the values detected by the diodes.



4.6. Control Board Ports

The ports connect the control board to the MSP430 and are configured to control all power and functionality for the ADE.



5. Test Process

5.1. Project 1

In order to verify that the power system was functioning properly, we used a voltmeter to test the voltage across two nodes, Vin and Vout, on the control board. We verified that the voltage supplied to the processor was 3.3V and the voltage supplied for use in the GPS was 1.8V. We ensured that the board was fully functioning by loading the Hardware_Test code and making sure the LEDs functioned.

5.2. Project 2

The LCD was mounted onto the control board and certain capacitors were carefully installed. The LCD was tested using Hardware_Test and shown to be editable.

5.3. Project 3

To verify the motors were both functioning properly, several files were used to make the vehicle drive in specified shapes. Timers were instituted to replace certain functions of the software.

5.4. Project 4

Reverse functionality was added through the completion of the H-bridge. Extensive testing of certain test points on the control board verified the integrity of the H-bridge construction. Reverse functionality was demonstrated used shapes made by the device through driving.

5.5. Project 5

Black line detection was implemented using two detectors and an IRLED emitter. Test samples of the ground below the car denoted the color of the surface directly under the detectors. Continuous sampling was tested against a buffer value of what could be considered “black”, “white”, and “ambient”, which were calibrated prior to testing.

5.6. Project 6

Serial communication was tested using the LCD display and switches. Switch 1 initiated a transfer from one car to the next connected car in a ring, incrementing the number on their LCDs until they reached a certain value. Intermittent testing was performed to ensure the serial communication stopped after the serial link was broken.

6. Software

The code that the ADE is built on is modular in design. Upon power up, the software initiates the ports that the processor uses to interface with the motors, LCD, switches, on-board LEDs, and other functions, taking care to give each function ample time to initialize properly before starting the next module. The first LED will turn on during boot to indicate the power system is working properly. Turning on the right and left motors will also turn on the third and fourth LEDs, respectively.

7. Block Progression

The following shows the progression of the code as it operates the ADE.

7.1. Main Blocks

The main block initiates first after the ADE powers on. Main is not fed any parameters. First, the main function calls other functions to initialize the ports, clocks, timers, LCD, interrupts, and LEDs, and sets the initial state of the LCD. Then the main function enters a while loop that runs continuously, allowing interrupts to run modules when appropriate.

7.2. Initialization Blocks

Init_Ports() initializes the port configurations to certain input, output, or function states.

Init_Clocks() initializes the clock module in the processor that defines the initial clock sources and speeds.

Init_Timers() initializes timers A0, A1, and B2 by defining their clock sources and timer intervals.

Init_LCD() initializes the LCD backlight and state.

Init_LEDs() initializes the LED port configurations and allows LED manipulation by other areas of the code.

7.3. Interrupt Blocks

Interrupt_timers directs the LCD backlight to toggle twice per second, defines the debounce cycle loop variables, and increments the counter that is used by our five millisecond sleep function called five_more_msec_mom(). 4

Interrupt_ports directs the switch debounce module that disallows rapid switch activation during a small time interval. The debounce time interval is serviced by Timer A0.

8. Software Listing

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

8.1. Main.c

```
//------------------------------------------------------------------------------
// Description: This file contains the Main Routine - "While" Operating System
//
// Tyler Cheek
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//------------------------------------------------------------------------------

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

// Global Variables
volatile unsigned char control_state[CNTL_STATE_INDEX];
volatile unsigned int Time_Sequence;
volatile char one_time;
volatile unsigned int five_msec_count;
volatile unsigned int SW1_PRESSED = FALSE;
volatile unsigned int SW2_PRESSED = FALSE;
volatile unsigned int SW1_DEBOUNCED = FALSE;
volatile unsigned int SW2_DEBOUNCED = FALSE;
volatile unsigned int SW1_DEBOUNCE_COUNT = INIT;
volatile unsigned int SW2_DEBOUNCE_COUNT = INIT;
```

```
extern char *display_2;


char *display_NCSU;
char *display_P3;


//---------------------------------------------------------------------------


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.
//
//---------------------------------------------------------------------------
  Init_Ports();                       // Initialize Ports
  Init_Clocks();                      // Initialize Clock System
  Init_Conditions();
  //PJOUT |= LED1;                    // Turn LED 1 on to indicate boot
  Time_Sequence = INIT;               //
  Init_Timers();                      // Initialize Timers
  five_more_msec_mom(TWOHUNDREDFIFTY_MSEC_DELAY);          // 250 msec delay for the clock to settle
  Init_LCD();                         // Initialize LCD
  five_more_msec_mom(THREEHUNDREDSEVENTYFIVE_MSEC_DELAY);  // 375 msec delay for the clock to settle
  Init_LEDs();                        // Initialize LEDs
  display_NCSU = "   N.C. State   ";
  display_P3 =  "   Homework_06   ";
  display_2 = "";
  SW1_PRESSED = FALSE;                // No switches have been pressed at start
  SW2_PRESSED = FALSE;


//  lcd_command(CLEAR_DISPLAY);
  five_more_msec_mom(TEN_CYCLES);
  lcd_out(display_NCSU,LCD_LINE_1);     // 16 characters max between quotes - line 1
  lcd_out(display_P3,LCD_LINE_2);       // 16 characters max between quotes - line 2


//---------------------------------------------------------------------------
// Begining of the "While" Operating System
//---------------------------------------------------------------------------
// while(ALWAYS) {                     // Can the Operating system run
//  switch(Time_Sequence){
//    case TS_DAMN_DIS_IS_HUGE:                   // 1000 msec
//     if(one_time){
//       Init_LEDs();                   // Initialize LEDs
```

```
//      one_time = INIT;
//    }
//    Time_Sequence = INIT;              //
//  case TS_PRETTY_BIG:                  // 1000 msec
//    if(one_time){
//      PJOUT |= LED4;                 // Change State of LED 4
//      P3OUT |= LED5;                 // Change State of LED 5
//      one_time = INIT;
//    }
//  case TS_LARGE:                       // 750 msec
//    if(one_time){
//      PJOUT |= LED3;                 // Change State of LED 3
//      P3OUT |= LED6;                 // Change State of LED 6
//      one_time = INIT;
//    }
//  case TS_MEDIUM:                      // 500 msec
//    if(one_time){
//      PJOUT |= LED2;                 // Change State of LED 2
//      P3OUT |= LED7;                 // Change State of LED 7
//      one_time = INIT;
//    }
//  case  TS_KIND_OF_SMALL:              // 250 msec
//    if(one_time){
//      PJOUT |= LED1;                 // Change State of LED 1
//      P3OUT |= LED8;                 // Change State of LED 8
//      one_time = INIT;
//    }
//    break;
//  default: break;
// }
// if(Time_Sequence > TS_DAMN_DIS_IS_HUGE){
//    Time_Sequence = INIT;
// }
  while(ALWAYS) {
  Switches_Process();
  lcd_out(display_NCSU,LCD_LINE_1);
  lcd_out(display_2,LCD_LINE_2);
  }
}
```

8.2. Init.c

```
//-------------------------------------------------------------------------------
// Description: This file contains the Initial Conditions routine
```

```
// Tyler Cheek
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------


//-------------------------------------------------------------------------------


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

void Init_Conditions(void){
//============================================================================
// Function name: Init_LEDs
//
// Description: This function enalbes interrupts because they are disabled
// by default.
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
 enable_interrupts();


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

8.3. Interrupt_ports.c

```
//-------------------------------------------------------------------------------
// Description: This file contains the interrupt instructions relating to the
// ports in the device.
//
// Tyler Cheek
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------


#include  "msp430.h"
```

```
#include  "macros.h"


extern char *display_1;
extern char *display_2;
volatile char switch_char;
extern volatile unsigned int SW1_PRESSED;
extern volatile unsigned int SW2_PRESSED;
extern volatile unsigned int SW1_DEBOUNCED;
extern volatile unsigned int SW2_DEBOUNCED;
extern volatile unsigned int SW1_DEBOUNCE_COUNT;
extern volatile unsigned int SW2_DEBOUNCE_COUNT;


#pragma vector=PORT4_VECTOR
__interrupt void switch_interrupt(void) {
//=============================================================================
// Function name: switch_interrupt
//
// Description: This is function that interupts the CPU to take action when
// switches are pressed.
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: display_1, display_2, SW1_PRESSED, SW2_PRESSED, SW1_DEBOUNCE_COUNT
//         SW2_DEBOUNCE_COUNT, SW1_DEBOUNCED, SW2_DEBOUNCED
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
// Switch 1
 if (P4IFG & SW1) {
   TA0CCTL1 |= CCIE;
   TA1CCTL0 |= CCIE;
// Turn LED 3 on to indicate boot ISR working
   PJOUT |= LED4;
// Set a variable to identify the switch has been pressed.
   SW1_PRESSED = TRUE;
// Set a variable to identify the switch is being debounced.
   SW1_DEBOUNCED = TRUE;
// Reset the count required of the debounce.
   SW1_DEBOUNCE_COUNT = INIT;
// Disable the Switch Interrupt
```

```
  P4IE &= ~SW1;
// Clear any current timer interrupt.
  P4IFG &= ~SW1;
  switch_char |= SW1;
  display_2 = "   Initiate  ";
 }
// Switch 2
 if (P4IFG & SW2) {
  TA0CCTL1 &= ~CCIE;
  TA1CCTL0 &= ~CCIE;
// Turn LED 4 on to indicate boot ISR working
  PJOUT |= LED4;
  wheels_off();
// Set a variable to identify the switch has been pressed.
  SW2_PRESSED = TRUE;
// Set a variable to identify the switch is being debounced.
  SW2_DEBOUNCED = TRUE;
// Reset the count required of the debounce.
  SW2_DEBOUNCE_COUNT = INIT;
// Disable the Switch Interrupt.
  P4IE &= ~SW2;
// Clear any current timer interrupt.
  P4IFG &= ~SW2;
  switch_char |= SW2;
  display_2 = "    Stop   ";
 }
// Enable the Timer Interrupt for the debounce.
}
```

8.4. Interrupt_timers.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the interrupt instructions for timers that
//  are dependent on interrupts in other parts of the code.
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------

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

```
unsigned int count_up;
volatile char switch_flag;
extern volatile char switch_char;
extern volatile unsigned int SW1_PRESSED;
extern volatile unsigned int SW2_PRESSED;
extern volatile unsigned int SW1_DEBOUNCED;
extern volatile unsigned int SW2_DEBOUNCED;
extern volatile unsigned int SW1_DEBOUNCE_COUNT;
extern volatile unsigned int SW2_DEBOUNCE_COUNT;
extern volatile unsigned int DRIVING_FORWARD;
extern volatile unsigned int DRIVING_BACKWARD;
volatile unsigned int FIFTY_MSEC_TIMER = INIT;
volatile unsigned int ONE_SEC_COUNTER = INIT;

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer0_A0(void){                       // 50ms timer
//=============================================================================
// Function name: Timer0_A0
//
// Description: This interrupts the CPU to take action every
// interval of 50ms.
//
// Passed : no variables passed
// Locals: switch_flag, FIFTY_MSEC_TIMER
// Returned: no values returned
// Globals: display_1, display_2, SW1_PRESSED, SW2_PRESSED, SW1_DEBOUNCE_COUNT
//        SW2_DEBOUNCE_COUNT, SW1_DEBOUNCED, SW2_DEBOUNCED
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
 FIFTY_MSEC_TIMER++;
 count_up ++;
 if(count_up == DECADE) {
   P3DIR ^= LCD_BACKLITE;
   count_up = INIT;
 }
 TA0CCR0 += TA0CCR0_INTERVAL; // Add Offset to TACCR0
}

#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer0_A1(void){                       // 5ms timer
```

```
//===============================================================================
// Function name: Timer0_A1
//
// Description: This interrupts the CPU to take action every
// interval of 5ms.
//
// Passed : no variables passed
// Locals: switch_flag, FIFTY_MSEC_TIMER
// Returned: no values returned
// Globals: display_1, display_2, SW1_PRESSED, SW2_PRESSED, SW1_DEBOUNCE_COUNT
//        SW2_DEBOUNCE_COUNT, SW1_DEBOUNCED, SW2_DEBOUNCED
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//===============================================================================
  switch(__even_in_range(TA0IV,14)){
  case 0: break; // No interrupt
  case 2: // CCR1 not used
     // Put code here for CCR1
     if(SW1_DEBOUNCED)
        SW1_DEBOUNCE_COUNT++;
     if(SW2_DEBOUNCED)
        SW2_DEBOUNCE_COUNT++;
//     if(DRIVING_FORWARD) {
//        right_wheel_off();
//        //usleep10(20);
//        left_wheel_off();
//        drive_forward();
//     }
//     if(DRIVING_BACKWARD) {
//        right_wheel_off();
//        //usleep10(20);
//        left_wheel_off();
//        drive_reverse();
//     }
     if(SW2_PRESSED)
        wheels_off();
     TA0CCR1 += TA0CCR1_INTERVAL; // Add Offset to TACCR1
  break;
  case 4: // CCR2 not used
     // Put code here for CCR2
     // TA0CCR2 += TA0CCR2_INTERVAL; // Add Offset to TACCR2
```

```
  break;
  case 6: break; // reserved
  case 8: break; // reserved
  case 10: break; // reserved
  case 12: break; // reserved
  case 14: // overflow
     // Put code here for overflow
  break;
  default: break;
  }
}


#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer1_A0(void){                              // 1sec timer
//=============================================================================
// Function name: Timer1_A0
//
// Description: This interrupts the CPU to take action every
// interval of 1sec.
//
// Passed : no variables passed
// Locals: ONE_SEC_COUNTER
// Returned: no values returned
// Globals: display_1, display_2, SW1_PRESSED, SW2_PRESSED, SW1_DEBOUNCE_COUNT
//         SW2_DEBOUNCE_COUNT, SW1_DEBOUNCED, SW2_DEBOUNCED
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
  switch(ONE_SEC_COUNTER) {
  case 0: // Do nothing
     PJOUT |= LED2;
     break;
  case 1: // Do nothing
     PJOUT |= LED3;
     break;
  case 2: // Do nothing
     PJOUT |= LED4;
     break;
  case 3: // Do nothing
     P3OUT |= LED5;
     break;
```

```
  case 4: // Drive forward for 1 second
     P3OUT |= LED6;
     wheels_off();
     drive_forward();
     break;
  case 5: // Drive backward for 2 seconds
     P3OUT |= LED6;
     wheels_off();
     drive_reverse();
     break;
  case 6: // Drive backward for 2 seconds
     P3OUT |= LED6;
     wheels_off();
     drive_reverse();
     break;
  case 7: // Drive forward for 1 second
     P3OUT |= LED6;
     wheels_off();
     drive_forward();
     break;
  case 8: // Spin clockwise for 1 second
     P3OUT |= LED6;
     wheels_off();
     left_wheel_fwd();
     right_wheel_rev();
     break;
  case 9: // Spin counterclockwise for 1 second
     P3OUT |= LED6;
     wheels_off();
     right_wheel_fwd();
     left_wheel_rev();
     ONE_SEC_COUNTER = INIT;
     break;
  default:
     ONE_SEC_COUNTER = INIT;
     break;
  }
  ONE_SEC_COUNTER++;
  TA1CCR0 += TA1CCR0_INTERVAL; // Add Offset to TACCR0
}
```

8.5. Clocks.c

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

```
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------


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


void Init_Clocks(void){
//=========================================================================
// Function name: Init_Clocks
//
// Description: This is the clock initialization for the program.
//
// Initial clock configuration, runs immediately after boot.
// Disables 1ms watchdog timer,
//     configure MCLK to 8MHz
//     configure SMCLK to 8MHz.
// Since the X1 oscillator is not currently connected,
// X1CLK is an unknown speed (probably ~10kHz).
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=========================================================================
  WDTCTL = WDTPW | WDTHOLD;          // Disable watchdog
// Clocks:
  CSCTL0 = CSKEY;                   // Unlock register


  CSCTL1 |= DCOFSEL0 + DCOFSEL1;      // Set max. DCO setting [8MHz]
  CSCTL2 = SELA_1 + SELS_3 + SELM_3;   // set ACLK = vlo; SMCLK = DCO; MCLK = DCO
  CSCTL3 = DIVA_0 + DIVS_0 + DIVM_0;   // set all dividers /1


  CSCTL0_H = CSLOCK;                // Lock registers
}
```

8.6. Functions.h

```
//-----------------------------------------------------------------------------
//  Description: This file contains a list of all functions used in this project
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------


// Function prototypes main
void main(void);
void Init_Conditions(void);
void Switches_Process(void);
void Init_Timers(void);
void Init_LEDs(void);



// Function prototypes clocks
void Init_Clocks(void);



// Function prototypes systems
void enable_interrupts(void);

// Function prototypes
__interrupt void Timer2_B0_ISR(void);
 __interrupt void TIMER2_B1_ISR(void);
void Init_Timer_B2(void);
void TimerB0code(void);
void usleep10(unsigned int usec);
void five_more_msec_mom(unsigned int fivemsec);
void five_msec_sleep(unsigned int fivemsec);
void Init_Timer_A0(void);
void Init_Timer_A1(void);

// Function prototypes ports
void Init_Ports(void);
void Init_Port1(void);
void Init_Port2(void);
void Init_Port3(void);
```

```
//void Init_PortJ(char gps_state);
void Init_PortJ(char clock_state);

// LCD Prototypes
void Init_LCD(void);
void lcd_clear(void);
void lcd_putc(char c);
void lcd_puts(char *s);
void lcd_out(char *s, char line);
void lcd_write(char data, char command);
void lcd_command( char data);

// Wheels
void wheels_off(void);
void right_wheel_fwd(void);
void right_wheel_rev(void);
void right_wheel_off(void);
void left_wheel_fwd(void);
void left_wheel_rev(void);
void left_wheel_off(void);
void drive_forward(void);
void drive_reverse(void);
```

8.7. LED.c

```
//-------------------------------------------------------------------------
//  Description: This file contains the LED routine
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

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

void Init_LEDs(void){
//=============================================================================
// Function name: Init_LEDs
//
```

```
// Description: This function ensures all LEDs are initiallized to be off.
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
  PJOUT &= ~LED1;
  PJOUT &= ~LED2;
  PJOUT &= ~LED3;
  PJOUT &= ~LED4;
  P3OUT &= ~LED5;
  P3OUT &= ~LED6;
  P3OUT &= ~LED7;
  P3OUT &= ~LED8;
}
```

8.8. Macros.h

```
//-----------------------------------------------------------------------------
//  Description: This file contains the macros used in all other files
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------


// Required defines
// In the event the universe no longer exists, this bit will reset
#define ALWAYS                  (1)
#define TRUE                    (1)
#define INCREMENT               (1)
#define FALSE                   (0)
#define INIT                    (0)
#define CLEAR_REGISTER          (0x00)
#define FUNCTION_REGISTER       (0x01)
#define CNTL_STATE_INDEX        (3) // Control States
#define LCD_LINE_1              (0x80) // Position Cursor at Character 01 of Line 1
#define LCD_LINE_2              (0xC0) // Position Cursor at Character 01 of Line 2
```

```
#define INPUT_SET           (0x00)
#define OUTPUT_SET          (0x01)
#define CIRCLE              (1)
#define TRIANGLE            (2)
#define FIGURE_EIGHT        (3)
#define HALF_SECOND_MODULO  (8)
#define DECADE              (10)

//Timer delays
#define TIME_DELAY                      (10)
#define TEN_CYCLES                      (5)
#define TWOHUNDREDFIFTY_MSEC_DELAY      (50)
#define DEBOUNCE_DELAY                  (5)
#define THREEHUNDREDSEVENTYFIVE_MSEC_DELAY  (75)
#define WTF_TOO_LARGE_DELAY             (1000)
#define TS_DAMN_DIS_IS_HUGE             (250)
#define TS_PRETTY_BIG                   (200)
#define TS_LARGE                        (150)
#define TS_MEDIUM                       (100)
#define TS_KIND_OF_SMALL                (50)
#define TA0CCR0_INTERVAL                (25000) // 8,000,000 / 8 / 2 / 25000 = 20Hz => 50msec
#define TA0CCR1_INTERVAL                (2500)  // 8,000,000 / 8 / 2 / 2500 = 200Hz => 5msec
#define TA1CCR0_INTERVAL                (1250)  // 10,000 / 1 / 8 / 1250 = 1Hz => 1sec
#define USECDELAY                       (14)
//LEDs
#define LED1            (0x01) // LED 1
#define LED2            (0x02) // LED 2
#define LED3            (0x04) // LED 3
#define LED4            (0x08) // LED 4
#define LED5            (0x10) // LED 5
#define LED6            (0x20) // LED 6
#define LED7            (0x40) // LED 7
#define LED8            (0x80) // LED 8

//Switches
#define SW1             (0x01) // Switch 1
#define SW2             (0x02) // Switch 2

//Port 1
#define V_DETECT_R          (0x01)
#define V_DETECT_L          (0x02)
#define IR_LED              (0x04)
#define V_THUMB             (0x08)
```

```
#define SPI_CS_LCD          (0x10) // LCD Chip Select
#define LCD_RESET           (0x20) // GPS Power Check from GPS
#define SPI_SIMO            (0x40) // SPI mode - slave in/master out of USCI_B0
#define RS_LCD              (0x80) // RS_LCD Command / Data selection
#define SOMI_B              (0x80) // SPI mode - slave out/master in of USCI_B0

//Port 2
#define USB_TXD             (0x01) //
#define USB_RXD             (0x02) //
#define SPI_SCK             (0x04) //
#define CPU_TXD             (0x08) // GPS Power Check from GPS
#define CPU_RXD             (0x10) // SPI mode - slave in/master out of USCI_B0
#define GPS_PWRCHK          (0x10)
#define GPS_PWRCNTL         (0x20)
#define GPS_RESET           (0x40)
#define GPS_PWR             (0x80)

//Port 3
#define LCD_BACKLITE        (0x08)

//Clocks
#define CSLOCK              (0x01) // Any incorrect password locks registers
#define MCLK_FREQ           (8000000L)
#define SMCLK_FREQ          (8000000L)
#define SMCLK_DIV           (64)
#define USE_R_FORWARD       (0x00)
#define USE_SMCLK           (0x01)

//directions
#define R_FORWARD           (0x01)
#define L_FORWARD           (0x02)
#define R_REVERSE           (0x04)
#define L_REVERSE           (0x08)
```

8.9. Port1.c

```
//-----------------------------------------------------------------------------
// Description: This file contains the routine for Port 1
//
// Tyler Cheek
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-----------------------------------------------------------------------------
```

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

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

//=============================================================================
// Init_Port1
// Purpose: Initialize Port 1
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA0 CCR1 capture: CCI1A input
// 0 1 1 - TA0 CCR1 compare: Out1
// 1 0 0 - External DMA trigger
// 1 0 1 - RTC clock calibration output
// 1 1 X - Analog input A0 - ADC, Comparator_D input CD0, Vref- External applied reference
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA0 CCR2 capture: CCI2A input,
// 0 1 1 - TA0 CCR2 compare: Out2
// 1 0 0 - TA1 input clock
// 1 0 1 - Comparator_D output
// 1 1 X - Analog input A1 - ADC, Comparator_D input CD1, Input for an external reference voltage to the ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA1 CCR1 capture: CCI1A input
// 0 1 1 - TA1 CCR1 compare: Out1
// 1 0 0 - TA0 input clock
// 1 0 1 - Comparator_D output
// 1 1 X - Analog input A2 - ADC, Comparator_D input CD2
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA1 CCR2 capture: CCI2A input
// 0 1 1 - TA1 CCR2 compare: Out2
```

```
// 1 0 X - Slave transmit enable - eUSCI_B0 SPI mode
// 1 1 X - Analog input A3 - ADC, Comparator_D input CD3
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB0 CCR1 capture: CCI1A input
// 0 1 1 - TB0 CCR1 compare: Out1
// 1 0 X - Slave transmit enable - eUSCI_A0 SPI mode// 1 1 X - Analog input A4 - ADC, Comparator_D input CD4
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB0 CCR2 capture: CCI2A input
// 0 1 1 - TBO CCR2 compare: Out2
// 1 0 X - Clock signal input - eUSCI_A0 SPI slave, Clock signal output - eUSCI_A0 SPI master
// 1 1 X - Analog input A5 - ADC, Comparator_D input CD5
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1 CCR1 capture: CCI1A input
// 0 1 1 - TB1 CCR1 compare: Out1
// 1 0 X - *Slave in, master out - eUSCI_B0 SPI mode, I2C data - eUSCI_B0 I2C mode
// 1 1 0 - TA0 CCR0 capture: CCI0A input
// 1 1 1 - TA0 CCR0 compare: Out0
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1 CCR2 capture: CCI2A input
// 0 1 1 - TB1 CCR2 compare: Out2 (not available on devices without TB1)
// 1 0 X - Slave out, master in - eUSCI_B0 SPI mode, I2C clock - eUSCI_B0 I2C mode
// 1 1 0 - TA1 CCR0 capture: CCI0A input
// 1 1 1 - TA1 CCR0 compare: Out0
//
// Passed: No variables passed
// Locals: No local variables
// Returned: No values returned
//
// Author: John Doe
// Date: Sept 2013
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
```

```
// Version: 1.0
//-------------------------------------------------------------------------
void Init_Port1(void){
// Set Selections bits
P1SEL0 = CLEAR_REGISTER; // P1 set default as I/0
P1SEL1 = CLEAR_REGISTER; // P1 set default as I/0
P1SEL0 &= ~SPI_SIMO; // SPI Function SIMO_B selected
P1SEL1 |= SPI_SIMO; // SPI Function SIMO_B selected
P1SELC |= V_DETECT_R; // Analog V_DETECT_R selected
P1SELC |= V_DETECT_L; // Analog V_DETECT_L selected
P1SELC |= V_THUMB; // Analog V_THUMB selected//Set Pin Direction bits
P1SEL0 &= ~SPI_CS_LCD; //
P1SEL1 &= ~SPI_CS_LCD;
P1OUT = CLEAR_REGISTER; // Set all outputs low for safety
P1DIR = INPUT_SET; // Set P1 default direction to input
P1DIR |= SPI_CS_LCD;
P1DIR |= SPI_SIMO; // SIMO_B set to Output
P1DIR |= RS_LCD; // Set P1 RS_LCD direction to output
P1DIR |= LCD_RESET; // Set P1 LCD_RESET direction to output
P1DIR |= IR_LED; // Set P1 IR_LED direction to output

//Initialize outputs
P1OUT &= ~LCD_RESET; // Set LCD_RESET low
P1OUT &= ~IR_LED; // Set IR_LED low
P1OUT |= SPI_SIMO; // Set IR_LED high Configure for pullup resistor
P1OUT |= RS_LCD; // Set IR_LED high Configure for pullup resistor

//Enable resistors
P1REN |= RS_LCD; // Enable pullup resistor
P1REN |= SPI_SIMO; // Enable pullup resistor
}
```

8.10. Port2.c

```
//-------------------------------------------------------------------------
//  Description: This file contains the routine for Port 2
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------

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

```
#include  "functions.h"
#include  "macros.h"

//=============================================================================
// Init_Port2
// Purpose: Initialize Port 2
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB2 CCIOA capture: CCI0A input
// 0 1 1 - TB2 CCIOA compare: Out0
// 1 0 0 - Transmit Data - eUSCI_A0 UART mode
// 1 0 1 - Slave in, master out - eUSCI_A0 SPI mode
// 1 1 X - TB0 clock input
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB2 CCR1 capture: CCI1A
// 0 1 1 - TA0 CCR2 compare: Out1
// 1 0 0 - Receive data ? eUSCI_A0 UART mode
// 1 0 1 - Slave out, master in ? eUSCI_A0 SPI mode
// 1 1 X - Analog input A1 - ADC, Comparator_D input CD1, Input for an external reference voltage to the ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB2 CCR2 capture: CCI2A input
// 0 1 1 - Tb2 CCR2 compare: Out2
// 1 0 0 - Clock signal input ? eUSCI_B0 SPI slave mod
// 1 0 1 - Clock signal output ? eUSCI_B0 SPI master mode
// 1 1 X - Analog input A2 - ADC, Comparator_D input CD2
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA0 CCR0 capture: CCI0B
// 0 1 1 - TA0 CCR0 compare: Out0
// 1 0 X - Slave transmit enable ? eUSCI_A1 SPI mode
// 1 1 X - Analog input A6 - ADC, Comparator_D input CD10
//
// Various options for Pin 4
```

```
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TA1 CCR0 capture: CCI0B input
// 0 1 1 - TA1 CCR1 compare: Out0
// 1 0 X - Clock signal input ? eUSCI_A1 SPI slave mode, Clock signal output ? eUSCI_A1 SPI master mode (not available on deviceswithout eUSCI_A1)
// 1 1 X - Analog input A4 - ADC, Comparator_D input CD4
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB0 CCR0 capture: CCI0B input
// 0 1 1 - TBO CCR0 compare: Out0
// 1 0 X - Clock signal input - eUSCI_A0 SPI slave, Clock signal output - eUSCI_A0 SPI master
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1 CCR1 capture: CCI1B input
// 0 1 1 - TB1 CCR1 compare: Out1
// 1 0 X - Transmit data ? eUSCI_A1 UART mode
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
//
// Passed: No variables passed
// Locals: No local variables
// Returned: No values returned
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
// Version: 1.0
//-------------------------------------------------------------------------
void Init_Port2(void){
// Set Selections bits
P2SEL0 = CLEAR_REGISTER; // P2 set default as I/0
P2SEL1 = CLEAR_REGISTER; // P2 set default as I/0
P2SEL0 &= ~USB_TXD;
P2SEL1 |= USB_TXD;
P2SEL0 &= ~USB_RXD;
P2SEL1 |= USB_RXD;
P2SEL0 &= ~SPI_SCK;
```

```
P2SEL1 |= SPI_SCK;
P2SEL0 &= ~CPU_TXD;
P2SEL1 |= CPU_TXD;
P2SEL0 &= ~CPU_RXD;
P2SEL1 |= CPU_RXD;

//Enable resistors
P2REN |= SPI_SCK; // Enable pullup resistor
}
```

8.11. Port3.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the routine for Port 3
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------

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

//=============================================================================
// Init_Port3
// Purpose: Initialize Port 3
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A12 ? ADC, Comparator_D input CD12, Vref- External applied reference
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A13 ? ADC, Comparator_D input CD13, Input for an external reference voltage to the ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A14 - ADC, Comparator_D input CD14
```

```
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A15 - ADC, Comparator_D input CD15
//
// Various options for Pin 4
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1 CCR1 capture: CCI1B input
// 0 1 1 - TB1 CCR1 compare: Out1
// 1 1 X - TB2 clock input, SMCLK output
//
// Various options for Pin 5
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB1 CCR2 capture: CCI2B input
// 0 1 1 - TB1 CCR2 compare: Out2
// 1 1 X - Comparator_D output
//
// Various options for Pin 6
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB2 CCR1 capture: CCI1B input
// 0 1 1 - TB2 CCR1 compare: Out1
// 1 1 1 - TB1 clock input
//
// Various options for Pin 7
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 0 1 0 - TB2 CCR2 capture: CCI2B input
// 0 1 1 - TB2 CCR2 compare: Out2 (not available on devices without TB2)
//
// Passed: No variables passed
// Locals: No local variables
// Returned: No values returned
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
// Version: 1.0
//-------------------------------------------------------------------------
void Init_Port3(void){
  // Set Selections bits
```

```
  P3SEL0 = CLEAR_REGISTER; // P1 set default as I/0
  P3SEL1 = CLEAR_REGISTER; // P1 set default as I/0
  P3SELC &= ~LCD_BACKLITE;
  P3SEL0 &= ~LED5;
  P3SEL1 &= ~LED5;
  P3SEL0 &= ~LED6;
  P3SEL1 &= ~LED6;
  P3SEL0 &= ~LED7;
  P3SEL1 &= ~LED7;
  P3SEL0 &= ~LED8;
  P3SEL1 &= ~LED8;

  //Initialize outputs
  P3DIR |= LCD_BACKLITE;
  P3OUT = LCD_BACKLITE;
  P3OUT &= ~LED5;
  P3OUT &= ~LED6;
  P3OUT &= ~LED7;
  P3OUT &= ~LED8;

  //Set pin directions
  P3DIR |= LED5;       // Set P3 direction to output
  P3DIR |= LED6;       // Set P3 direction to output
  P3DIR |= LED7;       // Set P3 direction to output
  P3DIR |= LED8;       // Set P3 direction to output
}
```

8.12. Port4.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the routine for Port 4
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------

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

 //------------------------------------------------------------------
// Configure PORT4
```

```
//
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
//
// Passed: No variables passed
// Locals: No local variables
// Returned: No values returned
//
// Author: Tyler Cheek
// Date: Febrary 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
// Version: 1.0
//-------------------------------------------------------------------------------
void Init_Port4(void){
//  P4SEL0 = INPUT_SET;       // P4 set as I/0
//  P4SEL1 = INPUT_SET;       // P4 set as I/0
//  P4DIR = CLEAR_REGISTER;    // Set P4 direction to input
//  P4DIR &= ~(SW1 | SW2);     // Direction = input
//  P4OUT = CLEAR_REGISTER;    // Clear the output register for P4
//  P4OUT |= SW1 | SW2;       // Configure pullup resistor
//  P4REN |= SW1 | SW2;       // Enable pullup resistor


   P4SEL0 = CLEAR_REGISTER;   // P4 set as I/0
   P4SEL1 = CLEAR_REGISTER;   // P4 set as I/0
   P4DIR = CLEAR_REGISTER;    // Set P4 direction to input
   P4DIR &= ~(SW1 | SW2);     // Direction = input
   P4OUT = CLEAR_REGISTER;
   P4OUT |= SW1;              // Configure pullup resistor
   P4OUT |= SW2;              // Configure pullup resistor
   P4REN |= SW1;              // Enable pullup resistor
   P4REN |= SW2;              // Enable pullup resistor
   // Configure the interrupt
   P4IES |= SW1;             // P4.0 Hi/Lo edge interrupt
   P4IES |= SW2;             // P4.0 Hi/Lo edge interrupt
   P4IFG &= ~SW1;             // P4 IFG SW1 cleared
   P4IFG &= ~SW2;             // P4 IFG SW2 cleared
   // Start/Enable Interrupt
```

```
  P4IE |= SW1;            // P4.0 SW1 interrupt Enabled
  P4IE |= SW2;            // P4.0 SW2 interrupt enabled
}
```

8.13. PortJ.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the routine for Port 4
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------


//-------------------------------------------------------------------------------

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

 //-------------------------------------------------------------------------------
// Port J Pins
// Init_Port3
// Purpose: Initialize Port J
//
// Various options for Pin 0
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A12 ? ADC, Comparator_D input CD12, Vref- External applied reference
//
// Various options for Pin 1
// SEL0 SEL1 DIR
// 0 0 I:0 0:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A13 ? ADC, Comparator_D input CD13, Input for an external reference voltage to the ADC
//
// Various options for Pin 2
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A14 - ADC, Comparator_D input CD14
//
// Various options for Pin 3
// SEL0 SEL1 DIR
// 0 0 I:0 O:1 - *General-purpose digital I/O with port interrupt and wake up from LPMx.5
// 1 1 X - Analog input A15 - ADC, Comparator_D input CD15
```

```
// Passed: No variables passed
// Locals: No local variables
// Returned: No values returned
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
// Version: 1.0
//-------------------------------------------------------------------------------
void Init_PortJ(char clock_state){
 if (clock_state==USE_R_FORWARD) {
   // Set Selections bits
   PJSEL0 = CLEAR_REGISTER;    // PJ set as I/0
   PJSEL1 = CLEAR_REGISTER;    // PJ set as I/0
   PJSEL0 &= ~LED1;
   PJSEL1 &= ~LED1;
   PJSEL0 &= ~LED2;
   PJSEL1 &= ~LED2;
   PJSEL0 &= ~LED3;
   PJSEL1 &= ~LED3;
   PJSEL0 &= ~LED4;
   PJSEL1 &= ~LED4;

   //Initialize outputs
   PJOUT &= ~LED1;
   PJOUT &= ~LED2;
   PJOUT &= ~LED3;
   PJOUT &= ~LED4;

   //Set pin directions
   PJDIR = OUTPUT_SET;        // Set PJ direction to output
   PJDIR |= LED1;
   PJDIR |= LED2;
   PJDIR |= LED3;
   PJDIR |= LED4;
 }
 if (clock_state==USE_SMCLK) {
   P3SEL0 = FUNCTION_REGISTER; // set default as SMCLK
   P3SEL1 = FUNCTION_REGISTER;
   P3DIR |= clock_state;
 }
}
```

8.14. Ports.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the Ports routines
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------


//-------------------------------------------------------------------------------


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


void Init_Ports(void) {
//=========================================================================
// Function name: Init_Ports
//
// Description: This function calls all other port intitialization functions.
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=========================================================================
  Init_Port1();
  Init_Port2();
  Init_Port3();
  Init_Port4();
  Init_PortJ(USE_R_FORWARD);
}
```

8.15. Switch.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the Switch routine
//
//  Tyler Cheek
//  February 2015
```

```
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------


//-------------------------------------------------------------------------


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


extern volatile unsigned int SW1_PRESSED;
extern volatile unsigned int SW2_PRESSED;
extern volatile unsigned int SW1_DEBOUNCED;
extern volatile unsigned int SW2_DEBOUNCED;
extern volatile unsigned int SW1_DEBOUNCE_COUNT;
extern volatile unsigned int SW2_DEBOUNCE_COUNT;


void Switches_Process(void) {
//=============================================================================
// Function name: switch_interrupt
//
// Description: This is function that interupts the CPU to take action when
// switches are pressed.
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
// Switch 1
 if(SW1_DEBOUNCE_COUNT >= DEBOUNCE_DELAY){
    P4IE |= SW1;
    SW1_DEBOUNCED = FALSE;
    SW1_PRESSED = FALSE;
 }
 if(SW2_DEBOUNCE_COUNT >= DEBOUNCE_DELAY) {
    P4IE |= SW2;
    SW2_DEBOUNCED = FALSE;
    SW2_PRESSED = FALSE;
 }
```

```
 if (!(SW1_DEBOUNCED) && !(SW2_DEBOUNCED)) {
    TA0CCTL1 &= ~CCIE;          // CCR1 enable interrupt
    PJOUT &= ~LED4;
 }
}
```

8.16. System.c

```
//-------------------------------------------------------------------------------
//  Description: System Configurations
//  Tells the compiler to provide the value in reg as an input to an
//  inline assembly block.  Even though the block contains no instructions,
//  the compiler may not optimize it away, and is told that the value
//  may change and should not be relied upon.
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------

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

//-------------------------------------------------------------------------------

void enable_interrupts(void){
//=============================================================================
// Function name: enable_interrupts
//
// Description: This function contains the code enable interupts
//
// Passed : no variables passed
// Locals: no locals declared
// Returned: no values returned
// Globals: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
  __bis_SR_register(GIE);     // enable interrupts
}
```

8.17. Timers.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the Timers routines
//
//  Tyler Cheek
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------

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

// Global Variables
extern volatile unsigned char control_state[CNTL_STATE_INDEX];
extern volatile unsigned int Time_Sequence;
extern volatile char one_time;
extern volatile unsigned int five_msec_count;
extern volatile unsigned int FIFTY_MSEC_TIMER;

void Init_Timers(void){
//=============================================================================
// Function name: Init_Timers
//
// Description: This function initializes the timers held in the timersB2.r43
// file.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: no globals used
//
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================

  Init_Timer_A0(); //
  Init_Timer_A1(); //
//  Init_Timer_B0(); //
//  Init_Timer_B1(); //
```

```
 Init_Timer_B2();  //  Required for provided compiled code to work
}


void usleep10(unsigned int usec){
//============================================================================
// Function name: usleep10
//
// Description: This function forces the processor to sleep (perform counting)
// for a certain duration (in microseconds).
//
// At the current clock rate this yields about 10usec per value passed
// A value of  0 yields 1.37 uSec
// A value of  1 yields 1.87 uSec
// A value of  2 yields 2.36 uSec
// A value of  3 yields 2.86 uSec
// A value of  4 yields 3.36 uSec
// A value of  5 yields 3.86 uSec
// A value of  6 yields 4.36 uSec
// A value of  7 yields 4.85 uSec
// A value of  8 yields 5.354 uSec
// A value of  9 yields 5.854 uSec
// A value of 10 yields 6.353 uSec
// A value of 11 yields 6.851 uSec
// A value of 12 yields 7.351 uSec
// A value of 13 yields 7.849 uSec
// A value of 14 yields 8.350 uSec
// A value of 15 yields 8.849 uSec
// A value of 16 yields 9.342 uSec
// A value of 17 yields 9.84 uSec
// A value of 18 yields 10.34 uSec
// A value of 19 yields 10.84 uSec
//
// Passed : unsigned int usec
// Locals: int i, int j
// Returned: no globals used
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
 int i,j;
 for(j=INIT;j<usec;j++){
   for(i=INIT;i<USECDELAY;i++);
```

```
 }
}

void five_more_msec_mom(unsigned int fivemsec){
//============================================================================
// Function name: five_more_msec_mom
//
// Description: This function forces the processor to sleep (perform counting)
// for a certain duration (in 5 millisecond intervals).
//
// Passed : unsigned int fivemsec
// Locals: no variables declared
// Returned: no values returned
// Globals: five_msec_count
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
//Each count passed is at least x times 5 milliseconds
 FIFTY_MSEC_TIMER = INIT;
 while(FIFTY_MSEC_TIMER < fivemsec);
}

void five_msec_sleep(unsigned int fivemsec){
//============================================================================
// Function name: five_msec_sleep
//
// Description: This function forces the processor to sleep (perform counting)
// for a certain duration (in 5 millisecond intervals).
//
// Passed : unsigned int fivemsec
// Locals: no variables declared
// Returned: no values returned
// Globals: five_msec_count
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
//Each count passed is at least x times 5 milliseconds
 five_msec_count = INIT;
 while(fivemsec > (five_msec_count+INCREMENT));
```

```
}


void Init_Timer_A0(void) {
//=============================================================================
// Function name: Init_Timer_A0
//
// Description: This function contains the code to initialize Timer A0
// Timer A0 initialization sets up both A0_0 and A0_1-A0_2
//
// Passed : no variables passed
// Locals: one_time
// Returned: no values returned
// Globals: Time_Sequence, five_msec_count
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//=============================================================================
 TA0CTL = TASSEL__SMCLK;       // SMCLK source
 TA0CTL |= TACLR;              // Resets TA0R, clock divider, count direction
 TA0CTL |= MC__CONTINUOUS;     // Continuous up
 TA0CTL |= ID__2;              // Divide clock by 2
 TA0CTL &= ~TAIE;              // Disable Overflow Interrupt
 TA0CTL &= ~TAIFG;             // Clear Overflow Interrupt flag

 TA0EX0 = TAIDEX_7;            // Divide clock by an additional 8

 TA0CCR0 = TA0CCR0_INTERVAL;   // CCR0
 TA0CCTL0 |= CCIE;             // CCR0 enable interrupt
 TA0CCR1 = TA0CCR1_INTERVAL;   // CCR1
// TA0CCR2 = TA0CCR2_INTERVAL;  // CCR2
// TA0CCTL2 |= CCIE;           // CCR2 enable interrupt
}


void Init_Timer_A1(void) {
//=============================================================================
// Function name: Init_Timer_A1
//
// Description: This function contains the code to initialize Timer A1
// Timer A0 initialization sets up both A1_0 and A1_1-A1_2
//
// Passed : no variables passed
// Locals: one_time
```

```
// Returned: no values returned
// Globals: Time_Sequence, five_msec_count
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
 TA1CTL = TASSEL__ACLK;        // ACLK source
 TA1CTL |= TACLR;              // Resets TA0R, clock divider, count direction
 TA1CTL |= MC__CONTINUOUS;     // Continuous up
 TA1CTL |= ID__1;              // Divide clock by 1
 TA1CTL &= ~TAIE;              // Disable Overflow Interrupt
 TA1CTL &= ~TAIFG;             // Clear Overflow Interrupt flag

 TA1EX0 = TAIDEX_7;            // Divide clock by an additional 8

 TA1CCR0 = TA1CCR0_INTERVAL;   // CCR0
// TA1CCR1 = TA1CCR1_INTERVAL;  // CCR1
// TA0CCR2 = TA0CCR2_INTERVAL;  // CCR2
// TA0CCTL2 |= CCIE;            // CCR2 enable interrupt

}



void TimerB0code(void){
//============================================================================
// Function name: TimerB0code
//
// Description: This function contains the code to interrupt Timer B0
//
// Passed : no variables passed
// Locals: one_time
// Returned: no values returned
// Globals: Time_Sequence, five_msec_count
//
// Author: Tyler Cheek
// Date: February 2015
// Compiler: Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//============================================================================
  Time_Sequence++;
  one_time = TRUE;
  if (five_msec_count < WTF_TOO_LARGE_DELAY){
    five_msec_count++;
```

```
 }
}
```

8.18. Wheels.c

```
//-------------------------------------------------------------------------------
// Description: System Configurations
// Tells the compiler to provide the value in reg as an input to an
// inline assembly block.  Even though the block contains no instructions,
// the compiler may not optimize it away, and is told that the value
// may change and should not be relied upon.
// Tyler Cheek
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------
#include  "msp430.h"
#include  "functions.h"
#include  "macros.h"
volatile unsigned int DRIVING_FORWARD = FALSE;
volatile unsigned int DRIVING_BACKWARD = FALSE;
void wheels_off(void) {
   PJOUT &= ~(R_FORWARD);
   PJOUT &= ~(L_FORWARD);
   PJOUT &= ~(R_REVERSE);
   PJOUT &= ~(L_REVERSE);
}
void right_wheel_fwd(void) {
   PJOUT |= R_FORWARD;
}
void right_wheel_rev(void) {
   PJOUT |= R_REVERSE;
}
void right_wheel_off(void) {
   PJOUT &= ~(R_FORWARD);
   PJOUT &= ~(R_REVERSE);
}
void left_wheel_fwd(void) {
   PJOUT |= L_FORWARD;
}
void left_wheel_rev(void) {

   PJOUT |= L_REVERSE;
}
void left_wheel_off(void) {
```

```
    PJOUT &= ~(L_REVERSE);
}
void drive_forward(void) {
    DRIVING_FORWARD = TRUE;
    DRIVING_BACKWARD = FALSE;
    right_wheel_fwd();
    left_wheel_fwd();
}
void drive_reverse(void) {
    DRIVING_FORWARD = FALSE;
    DRIVING_BACKWARD = TRUE;
    right_wheel_rev();
    left_wheel_rev();
}
```

8.19. Serial.c

```
//-------------------------------------------------------------------------------
//  Description: This file contains the functions that handle serial
//  communications.
//
//  Jason Zicherman
//  February 2015
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------

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

volatile unsigned int usb_rx_ring_wr;
volatile unsigned int usb_rx_ring_rd;
char *USB_Char_Rx;
char *USB_Char_Tx;
unsigned int temp = RESET;
extern int sw2state;
int start = RESET;
int increment = RESET;
unsigned int temptx = RESET;
unsigned int firsttime = INCREMENT;
unsigned int i = RESET;
volatile unsigned char GPS_Char_Rx[LARGE_RING_SIZE];
volatile unsigned char NEMA_BUF[LARGE_RING_SIZE];
volatile unsigned int gps_rx_ring_rd = RESET;
volatile unsigned int gps_rx_ring_wr = RESET;
volatile unsigned int parsenow = RESET;

//-------------------------------------------------------------------------------
// Function name: Init_Serial_UCA0
//
// Description: This function initializes serial communications.
//
// Passed : no variables passed
// Locals: i
// Returned: no values returned
// Globals: no values manipulated
//
// Jason Zicherman
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------
void Init_Serial_UCA1(void){
  USB_Char_Rx = "              ";    // USB Character
  USB_Char_Tx = "1     ";
  temp = RESET;
  usb_rx_ring_wr = RESET;
  usb_rx_ring_rd = RESET;
  // Configure UART 0
  UCA1CTLW0 = RESET;                    // Use word register
  UCA1CTLW0 |= UCSSEL__SMCLK;           // Set SMCLK as fBRCLK
  UCA1CTLW0 |= UCSWRST;                 // Set Software reset enable
  // 9,600 Baud Rate
  // 1. Calculate N = fBRCLK/Baudrate [if N > 16 continue with step 3, otherwise with step 2]
  // N = SMCLK / 4,800 => 8,000,000 / 4,800 = 1666.667
  // 2. OS16 = 0, UCBRx = INT(N) [continue with step 4]
  // 3. OS16 = 1, UCx = INT(N/16), UCFx = INT([(N/16) – INT(N/16)] × 16)
  // UCx = INT(N/16) = 1666.667/16 => 104
  // UCFx = INT([(N/16) – INT(N/16)] × 16) = ([1666.667/16-INT(1666.667/16)]*16) =>
  // (104.1666875-104)*16=>0.167*16=2
  // 4. UCSx can be found by looking up the fractional part of N ( = N - INT(N) ) in Table 18-4
  // Decicmal of SMCLK / 8,000,000 / 4,800 = 1666.667 => 0.667 yields 0xD6
  // 5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed
  // TX error (%) RX error (%)
  // BRCLK Baudrate UCOS16 UCBRx UCFx UCSx  neg   pos  neg  pos
  // 8000000 4800     1     104    2  0xD6 -0.08 0.04 -0.10 0.14
  UCA1BRW = BAUD;                       // 4,800 baud
```

```
 // UCA0MCTLW = UCSx concatenate UCFx concatenate UCOS16;
  // UCA0MCTLW = 0x49 concatenate 1 concatenate 1;
  UCA1MCTLW = IDK;
  UCA1CTL1 &= ~UCSWRST;                 // Release from reset
  UCA1IE |= UCRXIE;                     // Enable RX interrupt
}

//-------------------------------------------------------------------------------
// Function name: USCI_A0_ISR
//
// Description: This function handles serial interrupts.
//
// Passed : no variables passed
// Locals: temp
// Returned: no values returned
// Globals: usb_rx_ring_wr, USB_Char_Rx
//
// Jason Zicherman
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------------
#pragma vector=USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA1IV,INTRANGE)){
  case RESET: break;                    // Vector 0 - no interrupt
  case RXIFG:                           // Vector 2 - RXIFG
    temp = gps_rx_ring_wr++;
    GPS_Char_Rx[temp] = UCA1RXBUF;      // RX -> USB_Char_Rx character
    if(GPS_Char_Rx[temp] == LF){
      for(i = RESET; i < LARGE_RING_SIZE; i++){
        NEMA_BUF[i] = GPS_Char_Rx[i];
        GPS_Char_Rx[i] = RESET;
      }
      gps_rx_ring_wr = BEGINNING;
      parsenow = TRUE;
    }
    if(gps_rx_ring_wr >= LARGE_RING_SIZE){
      gps_rx_ring_wr = BEGINNING;       // Circular buffer back to beginning
    }
    break;
  case TXIFG: break;                    // Vector 4 - TXIFG
  default: break;
 }
}
```

```
//-------------------------------------------------------------------------
// Function name: Project6
//
// Description: This function passes the transfer pointer to the buffer.
//
// Passed : no variables passed
// Locals: no variables declared
// Returned: no values returned
// Globals: temptx, start
//
// Jason Zicherman
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------
void Project6(void){
  if(start == PRESSED){
    while(ALWAYS){
      UCA1TXBUF = *(USB_Char_Tx + temptx);
      temptx++;
      five_msec_delay(INCREMENT);
      if(temptx > MAXPLACES){
        start = RESET;
        temptx = RESET;
        break;
      }
    }
  }
}


//-------------------------------------------------------------------------
// Function name: increment_Rx
//
// Description: This function increments the received value.
//
// Passed : no variables passed
// Locals: places, placeinit, exponent, Rx_num
// Returned: no values returned
// Globals: firsttime, increment, start, USB_Char_Rx, USB_Char_Tx
//
// Jason Zicherman
// February 2015
// Built with IAR Embedded Workbench Version: V4.10A/W32 (5.40.1)
//-------------------------------------------------------------------------
void increment_Rx(void){
  if(increment == PRESSED){
    if(firsttime == INCREMENT){
      UCA1TXBUF = RESET;
      for(i = RESET; i < MAXPLACES; i++) *(USB_Char_Rx + i) = SPACECHAR;
      *USB_Char_Rx = ZEROASCII + INCREMENT;
      for(i = RESET; i < MAXPLACES; i++) *(USB_Char_Tx + i) = SPACECHAR;
      firsttime = RESET;
    }
    lcd_out("              ",LCD_LINE_1);
    lcd_out(USB_Char_Rx,LCD_LINE_1);
    lcd_out("              ",LCD_LINE_2);
    int places = RESET;
    int placeinit = RESET;
    int exponent = PRESSED;
    int Rx_num = RESET;
    while(*(USB_Char_Rx + places) >= ZEROASCII && *(USB_Char_Rx + places) <= NINEASCII){
      places++;
    }
    placeinit = places;
    places--;
    while(places >= RESET){
      Rx_num += (*(USB_Char_Rx + places) - ZEROASCII)*exponent;
      exponent *= MASK;
      places--;
    }
```

```
places = placeinit;
   Rx_num++;
   if((Rx_num == RESET) || (Rx_num == MASK) || (Rx_num == SHIFTTWO) || (Rx_num == SHIFTTHREE) || (Rx_num == SHIFTFOUR)){
     places++;
     placeinit++;
   }
   else exponent /= MASK;
   while(places > RESET){
     *(USB_Char_Tx + (placeinit-places)) = (((Rx_num/exponent)%MASK)|ASCIICONV);
     places--;
     exponent /= MASK;
   }
   increment = RESET;
   five_msec_delay(SHIFTTWO);
   start = PRESSED;
   UCA1IE |= UCRXIE;
 }
```

9. Conclusion

The Auto-Drone Extreme 2015 Hyper Lightning Edition is an advanced, revolutionary children's toy designed to entertain kids from ages 4 and up. The ADE is very easy to assemble and uses cost efficient parts. The ADE is very power efficient, and requires minimal testing. This document explains how to construct the car as well as how it works.