Educational Sensing Car

“Driving Growth in Young Minds”

Team 6

|  |  |
| --- | --- |
| Revision | Description |
| 1 | Updated section 6 with N-connection tests, software checks and cable connector checks |
| 2 | Figures were added to section 4, including H-Bridge schematics and descriptions of their functioning. |
| 3 | Added movement block to section 3. |
| 4 | Added definitions to section 2: N-FET Q21, R/L\_FORWARD |

Table of Contents

1. Scope 3

2. Abbreviations 3

3. Overview 3

3.1. Power System Block 3

3.2. Control Board Block 4

3.3. User Interface Block 4

3.4. Movement Block …………………………………………………………………………………………………….4

3.5. Software Block 4

4. Hardware 5

4.1. LCD/LED Push Button Interface 5

4.2. Power System 7

4.3. Microcontroller 8

4.4. Motor Control Board 10

4.5. Black Line Detection 10

6. Test Process 11

6.1. Voltmeter 11

6.2. LCD Functions 11

6.3. Testing Cable Connectors 11

6.4. Testing N-Connectors 11

6.5. Software Checks 11

6.6. Output LCD Checks 11

6.7. Interrupt Tests 11

6.8. ADC Outputs 11

6.9. IR LED Functionality 11

7. Software 5

8. Flow Chart 5

8.1. Main Blocks 5

8.2. Port Initialization Blocks 5

8.3. ADC Blocks 5

8.4. Timer Blocks 5

9. Software Listing 5

9.1. Main.c 5

9.2. TimerA0.c 5

9.3. TimerB0.c 5

9.4. Ports.c 5

9.4. ADC12\_B.c 5

Figures and Tables

Figure 3.1 Sensing Car Design Overview 3

Figure 3.2 Power System 3

Figure 3.3 Control Board 4

Figure 3.4 Input Blocks 4

Figure 3.5 Movement 4

Figure 4.1 Push Buttons and LCD Screen 5

Figure 4.2 LCD Screen and Backlite 5

Figure 4.3 Push Buttons and Reactive LEDs 6

Figure 4.4 Buck-Boost Converter Circui 7

Figure 4.5 Buck-Boost Converter 7

Figure 4.6 MSP430 Microcontroller Board 8

Figure 4.7 MSP430 Microcontroller 8

Figure 4.8 Microcontroller Pin Configuration 9

Figure 4.9 Motor Control PCB … 10

Figure 4.10 H-Bridge for Right Motor … 10

Figure 4.11 Infrared LED Emitter and Detectors… 10

Figure 4.12 Black Line Detection Hardware… 10

# Scope

This document describes the electronics of the Educational Sensing Car. The car will help kids learn their shapes, as well as interact with roads that they draw or create. Also, children can learn from the shapes that they create, making it an educational tool in many environments. This document covers the general process of developing the system created, as well as the methodology of the design. A basic overview of the design, in addition to implementations of the hardware and software, are described as well.

# Abbreviations

|  |  |
| --- | --- |
| AA | Predefined measurement of specific battery type – “50.5mm x 14.5mm” |
| ADC | Analog-to-Digital Converter |
| DC | Direct Current |
| FRAM | Ferroelectric Random Access Memory |
| J10  IR  ISR | Connection to the anode and cathode of the battery pack  Infrared Light  Interrupt Service Routine |
| LCD | Liquid Crystal Display |
| LED | Light Emitting Diode |
| MSP | Mixed Signal Processing |
| MSP430 | Model of specific Mixed Signal Processing board by Texas Instruments. |
| N-FET Q21 | A n-type field emitting transistor that drives the car’s motors forward |
| PCB | Printed Circuit Board |
| PWM | Pulse-Width Modulation |
| RAM | Random Access Memory |
| R/L\_FORWARD | A digital input/output pin on the MSP430 powering the motors’ forward drive |
| SEPIC | Single-Ended Primary-Inductor Converter |
| Wi-Fi | Trademarked term referring to a specific type of network connection |

# Overview

The major components of the car are shown below. These components allow the car to function efficiently and effectively. They are:

Power System

Software

Control Board

User Interface

Movement

Figure 3.1 Sensing Car Design Overview

## Power System Block

The car requires a distribution method of power to the rest of the system to ensure the car function correctly. The power system block includes the battery, the switch and the converter. The car uses 6V that comes from four 1.5 V AA batteries in series. There is a switch component that turn the system on and off. The SEPIC converter is used to regulate the voltage by its ability to change the output voltage to be different from the input voltage.

Battery

Switch

Converter

Figure 3.2 Power System

## Control Board Block

The control board is used to control the operation of the car. The control board includes the power/ LCD board, H-Bridge board, Wi-Fi board, and the emitter/detector board.

Power/LCD

Wi-Fi

Emitter/ detector

H-Bridge

Figure 3.1 Control Board

## User Interface Block

The user interface allows interaction between the car and the users. The user interface block includes LCD screen, LED(s) and the two push buttons. The LCD screen displays information of the car’s operation status. The LEDs are used to show output signals from the car. The two push buttons are used to switch between the different information that is displayed on the LCD screen and are used as inputs.

LED(s)

LCD Screen

Button 1

Button 2

Figure 3.4 User Interface

## Movement Block

The motor block is control by the FRAM board. It has two electric motors connected with wheels and a caster wheel in the back of the car. The FRAM board sends out instructions that are executed by the motors. The motors allow the car to go forward and make different shapes such as circle, figure 8 and triangle. The caster wheel is attached to the back of the car to keep the car level and easy to navigate.

Caster wheel

FRAM board

Left Motor

Right Motor

Figure 3.5 Movement

## Software Block

The system is programmed with the C language. The code is stored in the control board which controls how the devices act.

# Hardware

This section describes the peripheral components and microcontroller that make up the autonomous car.

## LCD/LED Push Button Interface

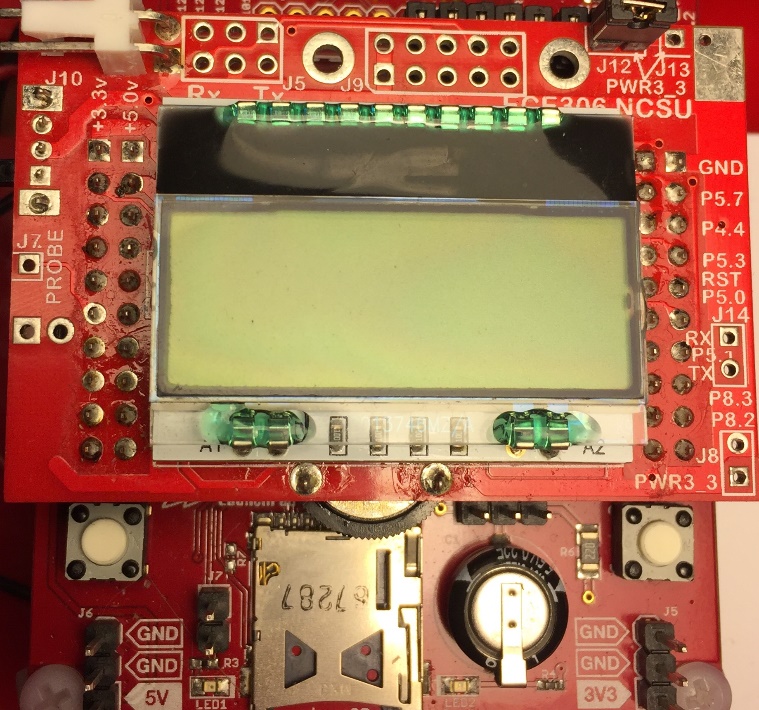


Figure 4.1 Pushbuttons and LCD Screen

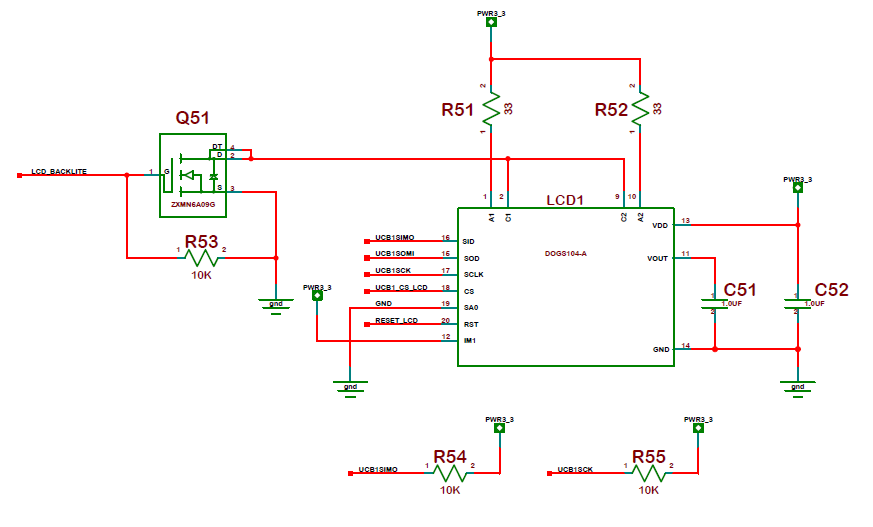
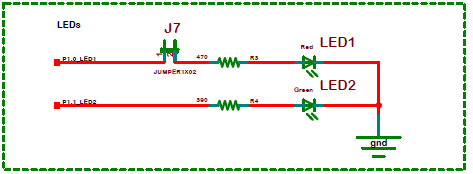
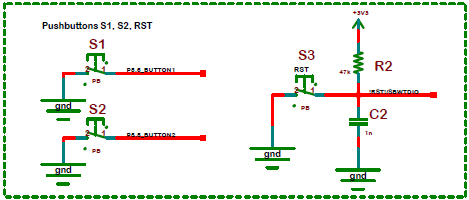


Figure 4.2 LCD Screen and Backlite



**Figure 4.3 Push Buttons and Reactive LEDs**

Users can modify the car's behavior through a responsive LCD screen and two LEDs. The screen switches between two messages to inform the user that a button has been pressed. To understand when input is received, the two primary switches are connected to pull-up resistors and “de-bounced” programmatically to respond accurately to each press. In addition, the LEDs alternate on/off to inform the user when the car is “thinking,” or that the car’s operating system is running in its uninterrupted routine. The two LEDs are connected to resistors to reduce their brightness to a tolerable scale and extend playtime.

## Power System

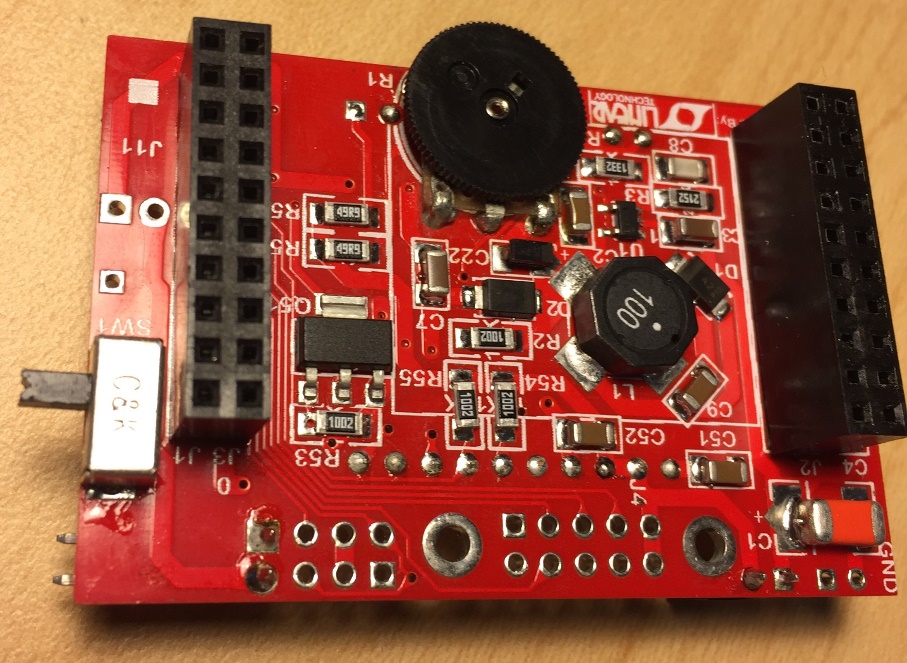


Figure 4.4 Buck-Boost Converter Circuit

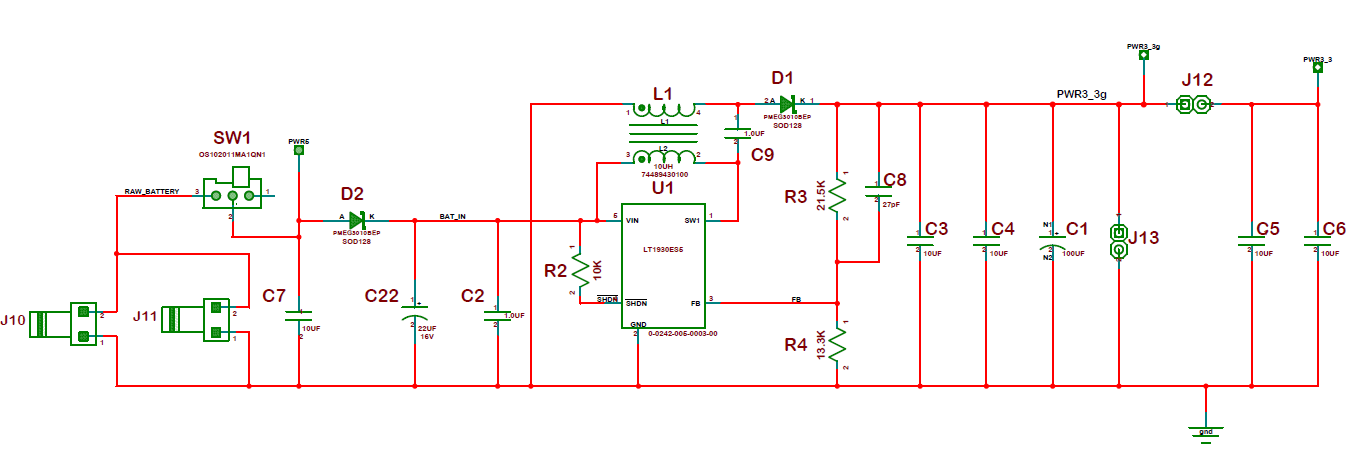


Figure 4.5 Buck-Boost Converter (DC Batteries to Car Microcontroller)

Power is received by 4 AA batteries connected to J10 . Since power can fluctuate between 2.8-6 Volts during regular use, a buck-boost converter adjusts the input voltage to 5 Volts. The MSP430 microcontroller then distributes the received power to the peripherals according to its processor, the user’s input and the car’s environment.

## Microcontroller

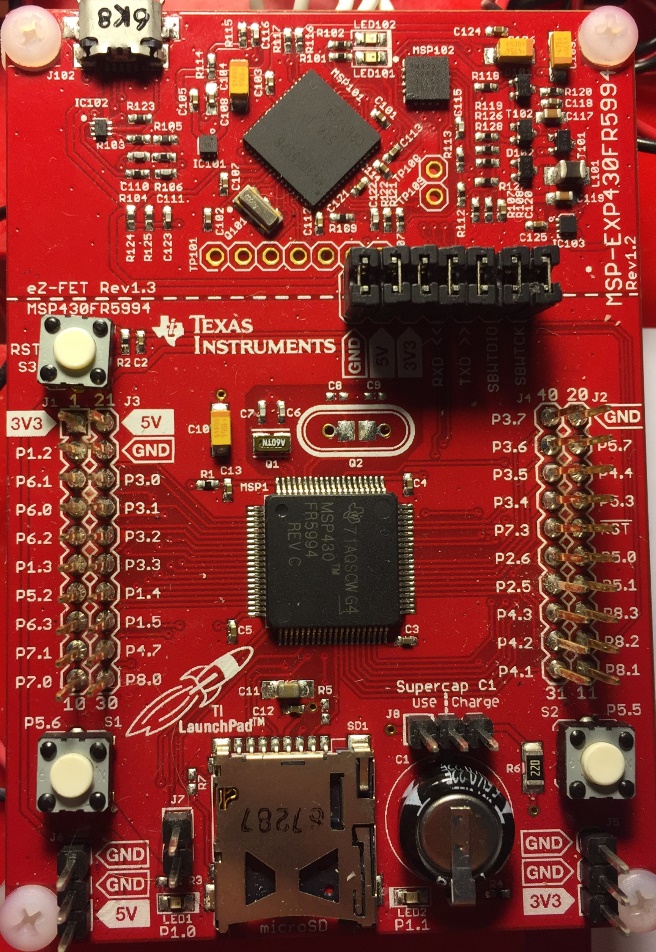


Figure 4.6 MSP430 Microcontroller Board

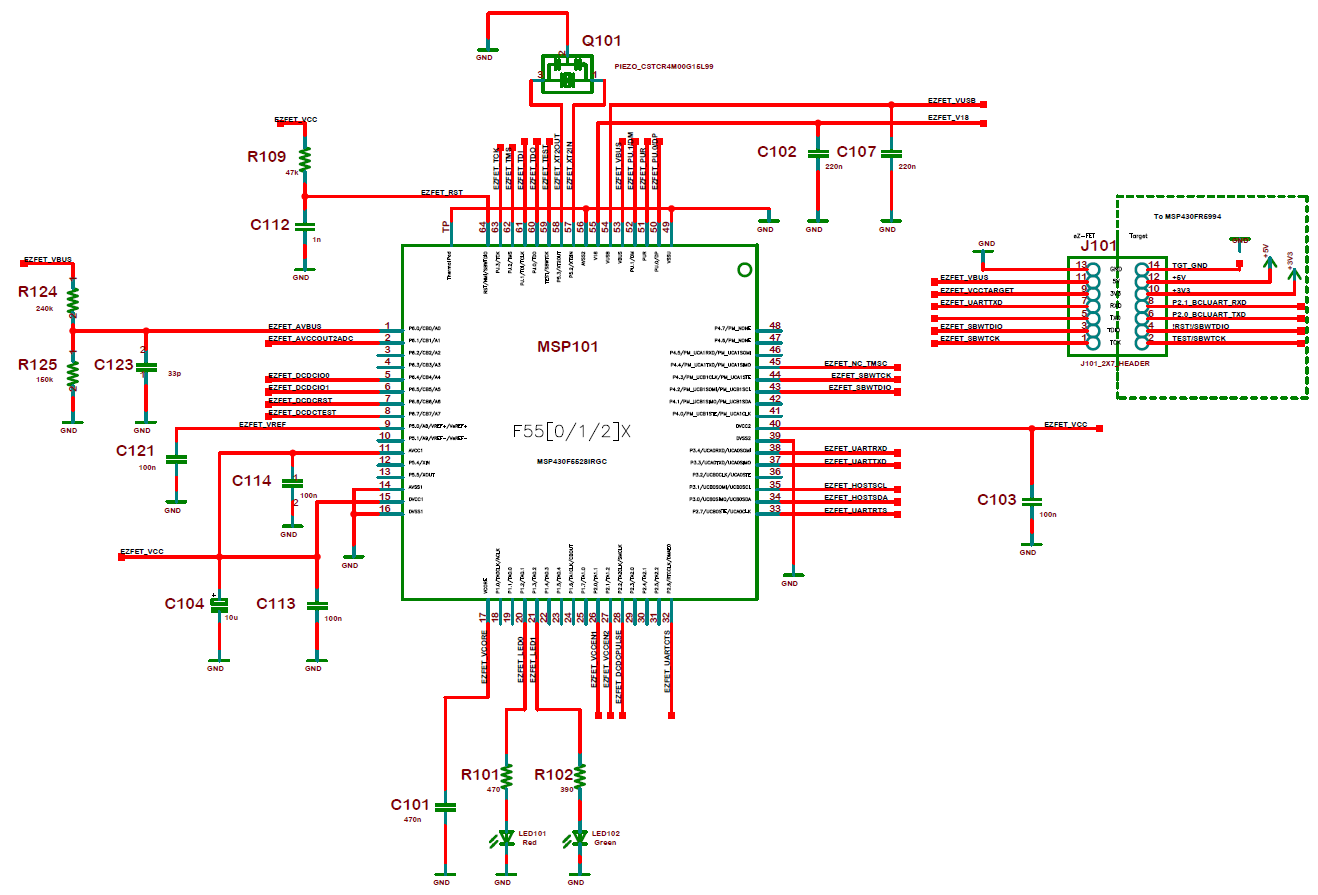


Figure 4.7 MSP430 Microcontroller

The Educational Sensing Car’s peripherals respond to stimuli using a programmed MSP430 microcontroller. For example, the microcontroller receives a low signal from a button and displays a new message to the LCD screen in response. The programmable pins connecting the listed peripherals to the microcontroller’s processor are listed below as proof of concept.

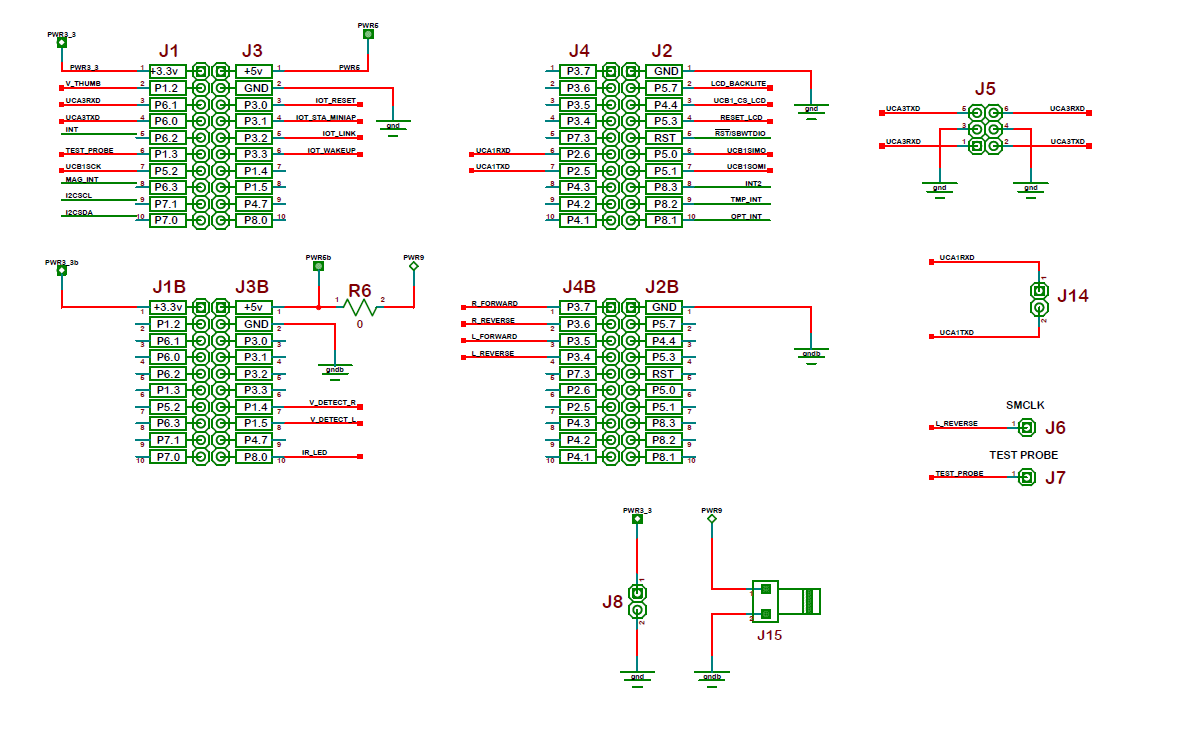


Figure 4.8 Microcontroller Pin Configuration

## Motor Control Board

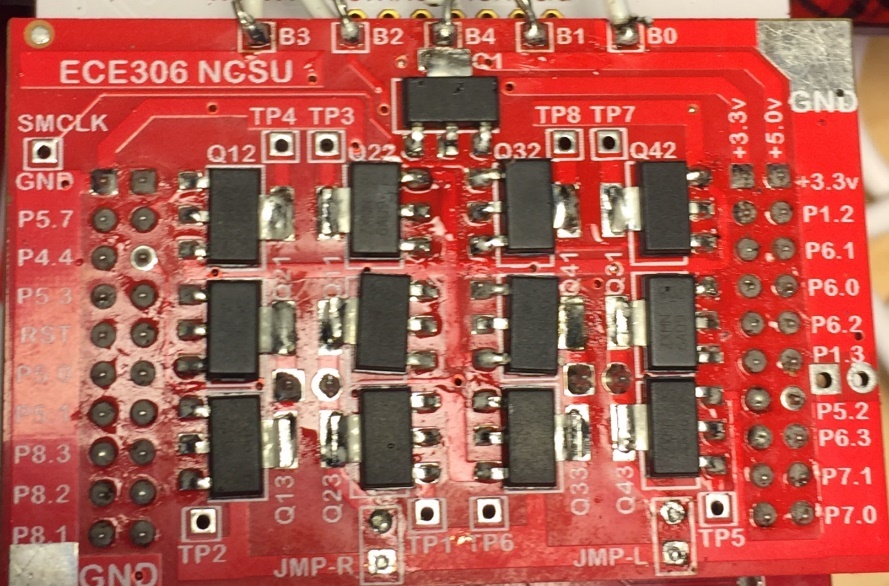


Figure 4.9 Motor Control PCB

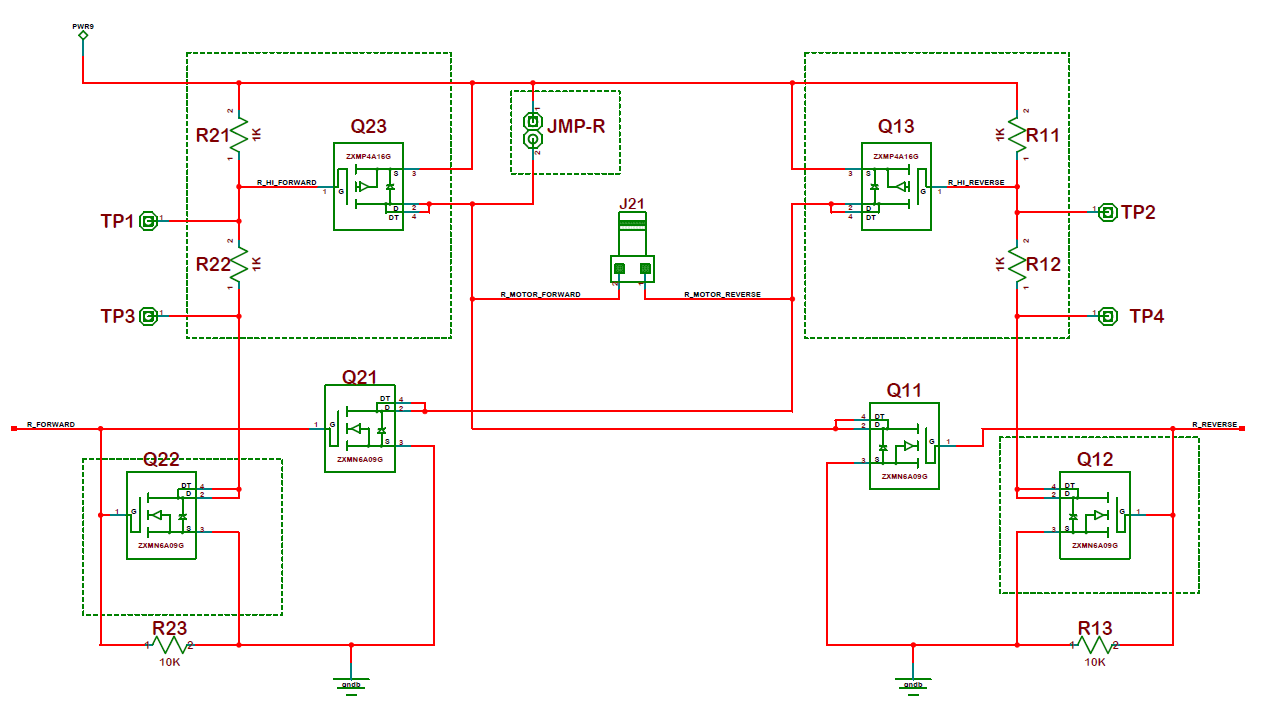


Figure 4.10 H-Bridge for Right Motor (mirror left H-Bridge schematic)

The Educational Sensing Car uses an h-bridge to drive each motor based on digital I/O signals from the MSP430. Although the above images display a fully-assembled h-bridge for each motor, the current implementation consists of only two half-h-bridges. The two half-h-bridges bridges each consist of the N-FET Q21 and can only drive the motors in the forward direction based on the digital signal received from R\_FORWARD and L\_FORWARD.

## Black Line Detection

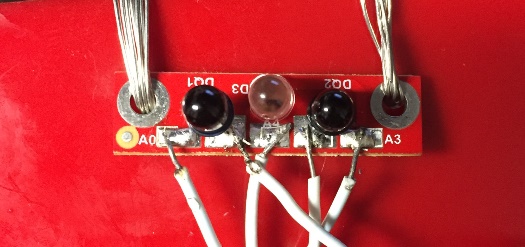


Figure 4.11 Infrared LED Emitter (center) and Detectors (left and right)

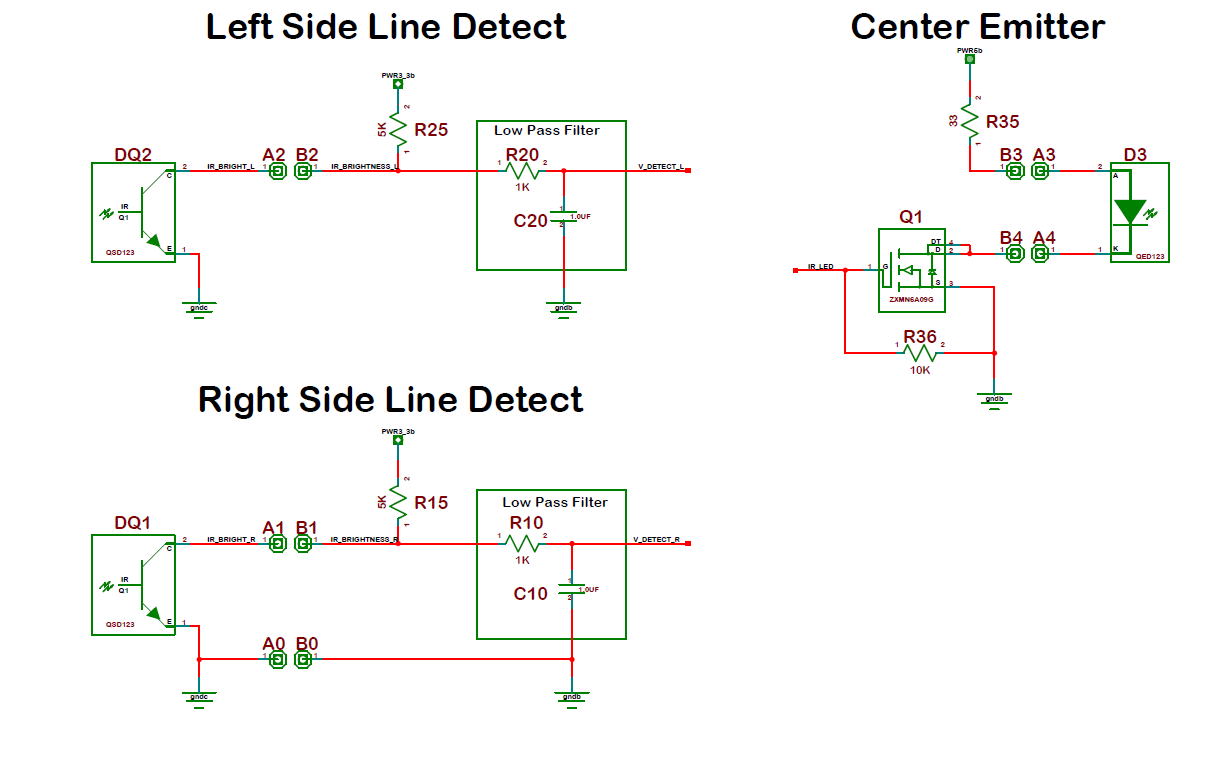


Figure 4.12 Black Line Detection Hardware

The car is configured to read the voltage across two infrared-sensitive LEDs in order to determine whether or not the car is situated over a black line. Infrared radiation is first emitted from an infrared LED, powered on and off based on user input through the button interface controlling a field emitting transistor on the emitter circuit. The radiation is then scattered off a surface under the car and intercepted via the infrared-sensitive detectors. For example, should the car be situated over a light surface, a lower voltage will be read, and if the car is situated over a black line, a higher voltage will be read. Voltage readings are sent to the analog to digital converter on the MSP430.

# Test Process

To test that the hardware was functioning properly, we performed various tests including probing the board with a volt meter to ensure the voltage across various terminals was correct and verifying that the LCD turned on and displayed the proper text.

## Volt meter

To verify that all the resistors were secured in place and connected after the assembly and reflow, we connected probes to ground and the high voltage point on the power board. With a volt meter on a test probe site, we then verified that the proper amount of voltage was present at the test probe site. This voltage should be around 3.2-3.3 volts. In addition, we used a multimeter over the pins in J12 to ensure that when the system was connected to the battery pack, the pins were again receiving the proper voltage.

## LCD Functions

The LCD is a very important part of the car as it allows the user to debug the system and therefore one must be certain that it works properly. To confirm that the LCD was connected properly, we carefully inspected all of the solder joints and checked that the backlight was snugly attached to the LCD.

## Testing Cable Connectors

The jumper cables used to power the motors for the wheels were tested with a 5-volt input to one side of the connector and ground on the other side. Then using a voltmeter, we determined whether the wires were correctly crimped into the connector by checking if the voltage reached the ends of the wire.

## Testing N-Connectors

To ensure that voltage from the batteries was reaching the newly installed N-Connectors, we plugged in the battery pack and turned it on. Then using a volt meter, we could check that the voltage was present between each terminal of the N-Connectors.

## Software Checks

Verification that the code we wrote could control the cars forward movement was implemented by first writing functions that updated the motor port pins and then calling these functions to observe the output of the motors. Once the vehicle could drive, we utilized the above functions to control specific movement that allowed the car to turn and curve dependent of whether the ports were on or off. From there, observations of the drawn shape could be converted to code updates that improved the car’s shapes.

## Output LCD checks

We used functions that would display the shape that the car was attempting to draw in order to ensure that we were evaluating the correct shapes and updating the code to correct for any mistakes.

## Interrupt Tests

During the testing of interrupts, we wrote code in the interrupts that toggled the state of the Backlite, Red LED, and Green LED every several hundred milliseconds. This helped us ensure that the interrupt was being called at the right frequency and the program was not being stalled anywhere.

## ADC Outputs

To ensure the ADC was correctly converting the data, we wrote code that would output to the screen all of the values of the Thumbwheel, and both detectors. This allowed us to check that each value was being updated and that the detector was working.

## IR LED Functionality

Because infrared light is not visible, it is difficult to determine whether the LED is on or off. To check this condition, we used the front facing camera on our phones which does not block infrared light to see when/if the LED was turning on.

# Software

The software was created using a modular approach, isolating functions by program based on which component they addressed. All pins on the MSP430 were configured in the program Ports.c along with user-initiated interrupts from the two buttons; pin configurations included I/O functionality for leds, PWM functionality for the left and right motor and analog input functionality for the infrared detectors. The configuration of the ADC – it’s sampling rate, timing and resolution – along with interrupts to handle ADC readings were handled in ADC12\_B.c. Timing for motor PWM was handled in TimerB0.c while timing for software delays was handled by interrupts in TimerA0.c. All software revolved around a continuous loop in Main.c in which updates to each component’s status could be requested and handled.

# Flow Chart

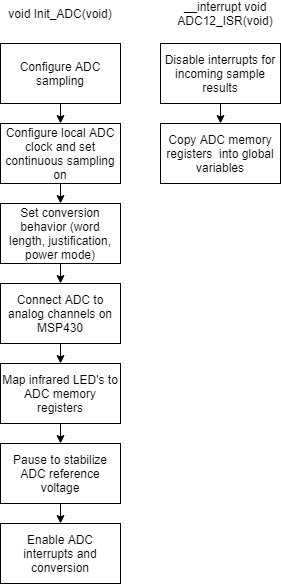
The following is a series of flow charts describing our code.

## Main Blocks

Each function should have its own section and be on a separate page. Insert page breaks when necessary. This describes the software. What calls the function? Etc. Do not just put C code statements in blocks. Also do not write novels. Be reasonable.

## Port Initialization Blocks

## ADC Blocks



The leftmost function initializes the ADC to begin performing conversions. The rightmost function is an interrupt service routine triggered by a new reading being loaded into the ADC memory registers associated with the infrared LED’s.

## A screenshot of a cell phone Description generated with high confidenceA screenshot of a social media post Description generated with very high confidenceTimer Blocks

The leftmost diagram above represents the dataflow of the initialization of Timer A0. The middle diagram above represents the interrupt service routine for Timer A0. The diagram above and to the right represents Timer B0’s initialization.

# Software Listing

This is a printout of the code. Each file is listed under its section.

## Main.c

## TimerA0.c

#define RESET\_STATE (0)

#define TA0CCR0\_INTERVAL (25000) //8,000,000/2/8/20 (50 msec)

#define TA0CCR1\_INTERVAL (25000) //8,000,000/2/8/20 (50 msec)

#define TA0CCR2\_INTERVAL (25000) //8,000,000/2/8/20 (50 msec)

extern int time\_counter =RESET\_STATE; //used to keep track of time elapsed

// Timer A0 initialization sets up both A0\_0, A0\_1, and 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; // Divide clock by 2

TA0EX0 = TAIDEX\_7; // Divide clock by an additional 8

TA0CCR0 = TA0CCR0\_INTERVAL; // CCR0 interval set

TA0CCTL0 |= CCIE; // CCR0 enable interrupt

TA0CCR1 = TA0CCR1\_INTERVAL; // CCR1 interval set

TA0CCTL1 |= CCIE; // CCR1 enable interrupt

TA0CCR2 = TA0CCR2\_INTERVAL; // CCR2 interval set

TA0CCTL2 &= ~CCIE; // CCR2 disable interrupt

TA0CTL &= ~TAIE; // Disable Overflow Interrupt

TA0CTL &= ~TAIFG; // Clear Overflow Interrupt flag

}

#pragma vector = TIMER0\_A0\_VECTOR

\_\_interrupt void Timer0\_A0\_ISR(void)

{

time\_counter++;

TA0CCR0 += TA0CCR0\_INTERVAL ;

}

## TimerB0.c

//PWM definitions

#define WHEEL\_PERIOD (10000)

#define L\_REV\_SPEED (TB0CCR3)

#define L\_FWD\_SPEED (TB0CCR4)

#define R\_REV\_SPEED (TB0CCR5)

#define R\_FWD\_SPEED (TB0CCR6)

#define WHEEL\_OFF (0)

//Timer B0 initialization sets up CCR 3-6 to control the wheel speeds as a ratio of the wheel period

void Init\_Timer\_B0(void){

TB0CTL |= TBSSEL\_\_SMCLK;//sets clock source to SMCLK

TB0CTL |= MC\_\_UP;//set to up mode

TB0CTL |=TBCLR;

TB0CCR0 = WHEEL\_PERIOD; //determines the interval for one period

TB0CCTL3 =OUTMOD\_7; //sets to continuous mode

L\_REV\_SPEED =WHEEL\_OFF;//sets the value of TB0CCR3 to the desired interval

TB0CCTL4 =OUTMOD\_7;//sets to continuous mode

L\_FWD\_SPEED =WHEEL\_OFF;//sets the value of TB0CCR4 to the desired interval

TB0CCTL5 =OUTMOD\_7;//sets to continuous mode

R\_REV\_SPEED =WHEEL\_OFF;//sets the value of TB0CCR5 to the desired interval

TB0CCTL6 =OUTMOD\_7;//sets to continuous mode

R\_FWD\_SPEED =WHEEL\_OFF;//sets the value of TB0CCR6 to the desired interval

}

## Ports.c

## ADC12\_B.c

#define ADC\_RESET\_STATE (0)

#define STABILIZE\_REFERENCE {\_\_delay\_cycles(10000);}

volatile uint16\_t ADC\_Thumb;

volatile uint16\_t ADC\_Right\_Detector;

volatile uint16\_t ADC\_Left\_Detector;

void Init\_ADC(void){

ADC12CTL0 = ADC\_RESET\_STATE;

/\* Configure ADC sampling; power on

\* -----------------

\* ADC12SHT0\_2: 16 ADC12CLK cycles in sampling period

\* ADC12SHT1\_2: 16 ADC12CLK cycles in sample-and-hold time

\* (ADC12MEM0 to ADC12MEM7 || ADC12MEM24 to ADC12MEM31)

\* ADC12MSC : First rising edge of SHI signal triggers sampling timer

\* ADC12ON : ADC12\_B powered ON

\*/

ADC12CTL0 |= (ADC12SHT0\_2 |

ADC12SHT1\_2 |

ADC12MSC |

ADC12ON);

/////////////////////////////////////////////////////////////////////////////

ADC12CTL1 = ADC\_RESET\_STATE;

/\* Configure ADC clocking

\* -----------------

\* ADC12PDIV\_0 : Predivide ADC12CLK by 1

\* ADC12SHS\_0 : ADC12SC as sample-and-hold source

\* ADC12SHP : SMPCON signal sourced from sampling timer

\* ADC12ISSH\_0 : Sample-input signal is not inverted

\* ADC12DIV\_0 : ADC12CLK divided by 1

\* ADC12SSEL0 : ADC12\_B clock source select (MODOSC)

\* ADC12CONSEQ\_3: Sequence-of-channels conversion sequence mode

\*/

ADC12CTL1 |= (ADC12PDIV\_0 |

ADC12SHS\_0 |

ADC12SHP |

ADC12ISSH\_0 |

ADC12DIV\_0 |

ADC12SSEL0 |

ADC12CONSEQ\_3);

/////////////////////////////////////////////////////////////////////////////

ADC12CTL2 = ADC\_RESET\_STATE;

/\* Configure conversion settings

\* -----------------

\* ADC12RES\_2 : 12-bit conversion result resolution (14 clock cycle conv.)

\* ADC12DF\_0 : Result data stored as binary unsigned, right justified

\* ADC12PWRMD\_0 : Regular power mode (not LPM) where sample rate

\* not restricted

\*/

ADC12CTL2 |= (ADC12RES\_2 |

ADC12DF\_0 |

ADC12PWRMD\_0);

/////////////////////////////////////////////////////////////////////////////

ADC12CTL3 = ADC\_RESET\_STATE;

/\* Configure ADC input channels

\* -----------------

\* ADC12ICH3MAP\_0 : External pin selected for ADC input channel A26

\* ADC12ICH2MAP\_0 : External pin selected for ADC input channel A27

\* ADC12ICH1MAP\_0 : External pin selected for ADC input channel A28

\* ADC12ICH0MAP\_0 : External pin selected for ADC input channel A29

\* ADC12TCMAP\_1 : Internal temperature sensor for ADC input channel A30

\* ADC12BATMAP\_1 : 1/2 AVCC channel sel. for ADC input channel A31

\* ADC12CSTARTADD0: ADC12MEM0 set as conversion start address (in sequence)

\*/

ADC12CTL3 |= (ADC12ICH3MAP\_0 |

ADC12ICH2MAP\_0 |

ADC12ICH1MAP\_0 |

ADC12ICH0MAP\_0 |

ADC12TCMAP\_1 |

ADC12BATMAP\_1 |

ADC12CSTARTADD\_0);

/////////////////////////////////////////////////////////////////////////////

ADC12MCTL0 = ADC12MCTL1

= ADC12MCTL2

= ADC\_RESET\_STATE;

/\* Configure ADC input channels

\* -----------------

\* ADC12WINC\_0 : Comparator window disabled

\* ADC12DIF\_0 : Single-ended mode enabled

\* ADC12VRSEL\_0: VR+ = AVCC, VR- = AVSS

\* ADC12INCH\_x : channel = Ax

\* ADC12EOS : End of sequence

\*/

ADC12MCTL0 |= (ADC12WINC\_0 |

ADC12DIF\_0 |

ADC12VRSEL\_0 |

ADC12INCH\_2);

ADC12MCTL1 |= (ADC12WINC\_0 |

ADC12DIF\_0 |

ADC12VRSEL\_0 |

ADC12INCH\_4);

ADC12MCTL2 |= (ADC12WINC\_0 |

ADC12DIF\_0 |

ADC12VRSEL\_0 |

ADC12INCH\_5 |

ADC12EOS);

/////////////////////////////////////////////////////////////////////////////

STABILIZE\_REFERENCE

ADC12IER0 |= (ADC12IE2 | // Enable interrupts for new sample results

ADC12IE4 |

ADC12IE5);

ADC12CTL0 |= (ADC12ENC | // Enable Conversion

ADC12SC);

}

#pragma vector = ADC12\_B\_VECTOR

\_\_interrupt void ADC12\_ISR(void){

ADC12IER0 &= ~(ADC12IE2 | // Disable interrupts for new sample results

ADC12IE4 |

ADC12IE5);

ADC\_Thumb = ADC12MEM0;

ADC\_Right\_Detector = ADC12MEM1;

ADC\_Left\_Detector = ADC12MEM2;

}