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

Team Members
Duy Dang
Christian Fischer
Rajwal Gautam
Sam Messick

Originator: Sam Messick
Checked: 3-19-2018 | Released: 3-19.2016
Filename: Project_06.doc
Title: **Educational Sensing Car Team 6**

Date:	Document Number:	Rev:	Sheet:
03/19/2018	**03-19-2018-P6**	**A**	**1 of 36**

Table of Contents

Figures and Tables

1. 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.

2. 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	Connection to the anode and cathode of the battery pack
IR	Infrared Light
ISR	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

3. Overview

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

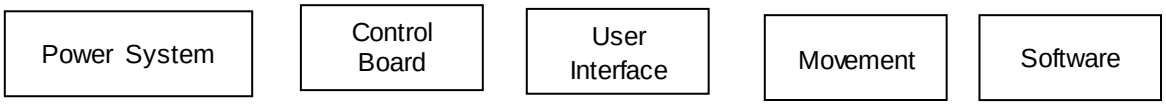


Figure 3.1 Sensing Car Design Overview

3.1. 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.

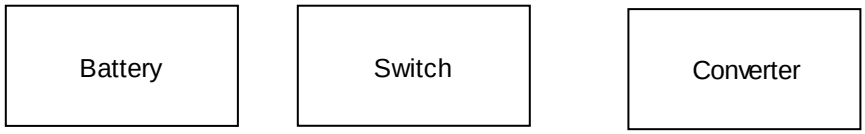


Figure 3.2 Power System

3.2. 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.

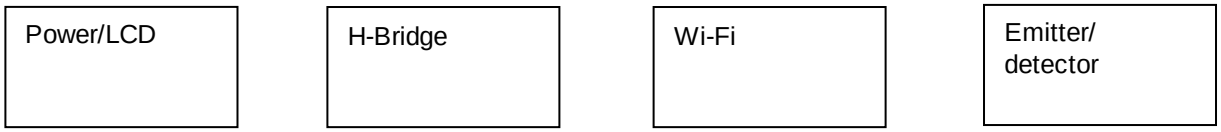


Figure 3.3 Control Board

3.3. 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.

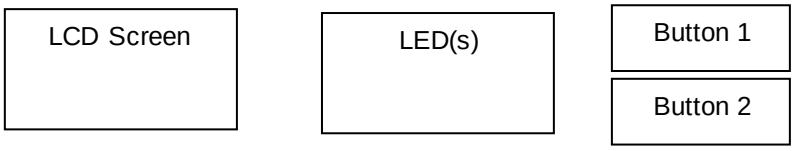


Figure 3.4 User Interface

3.4. 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.

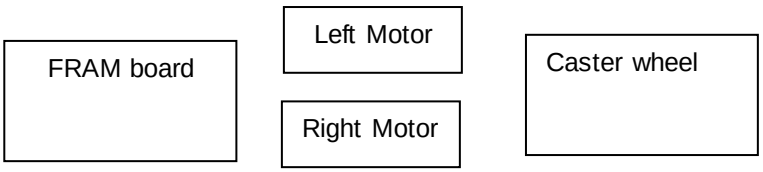


Figure 3.5 Movement

3.5. Software Block

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

4. Hardware

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

4.1. LCD/LED Push Button Interface

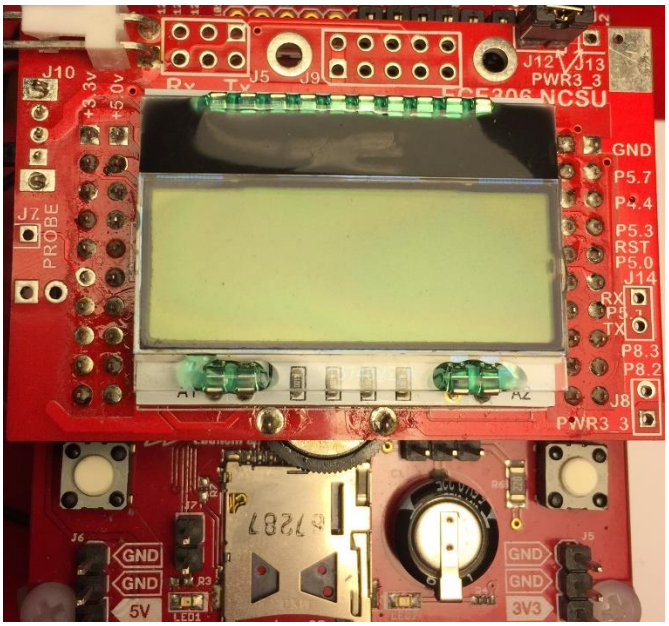


Figure 4.1 Pushbuttons and LCD Screen

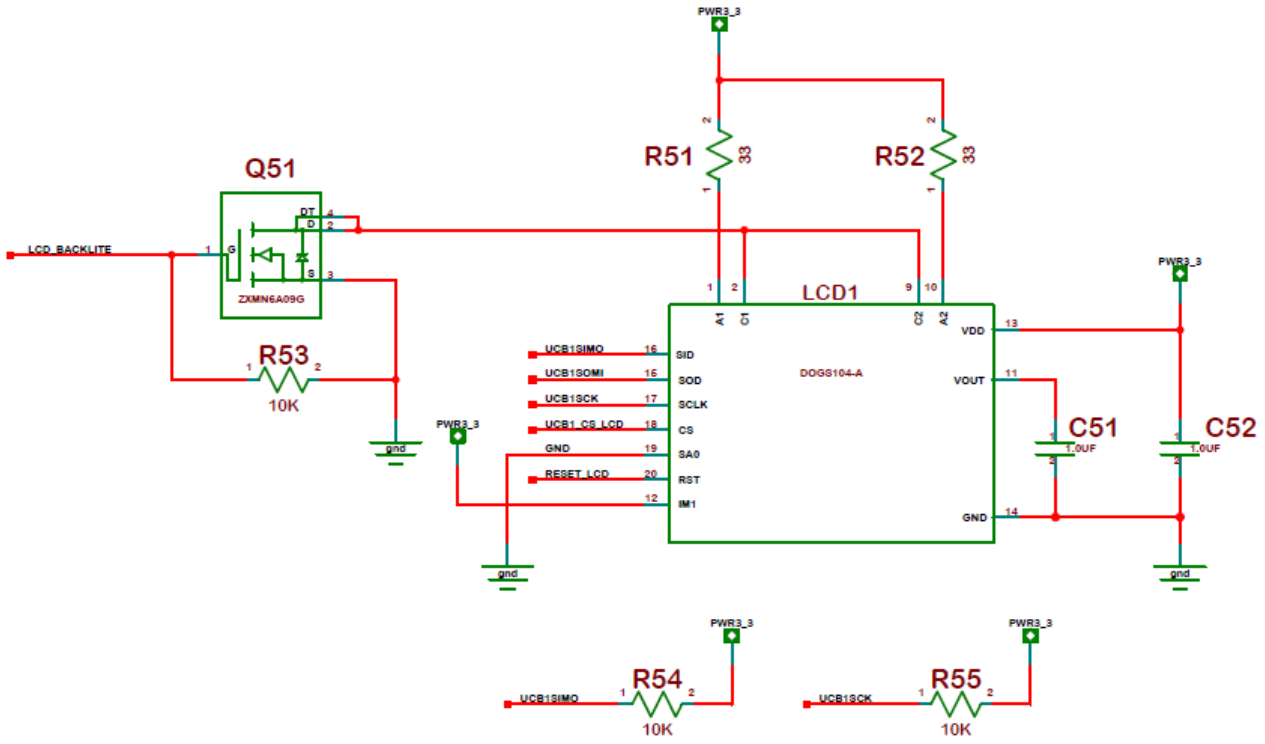


Figure 4.2 LCD Screen and Backlite

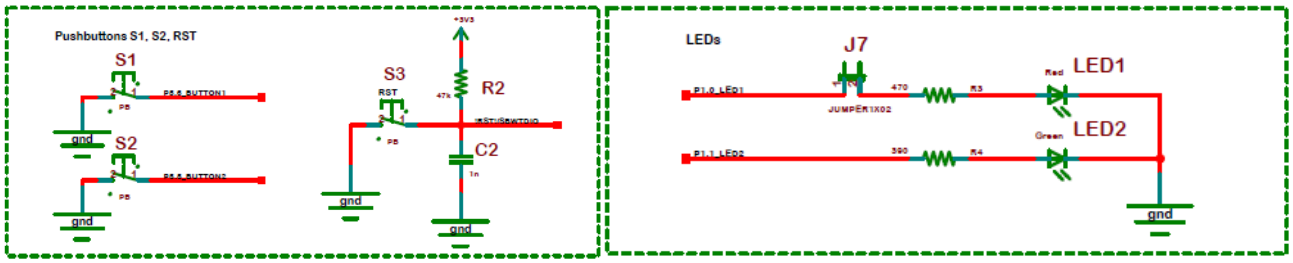


Figure 4.3 Push Buttons and Reactive LEDs

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.

4.2. Power System

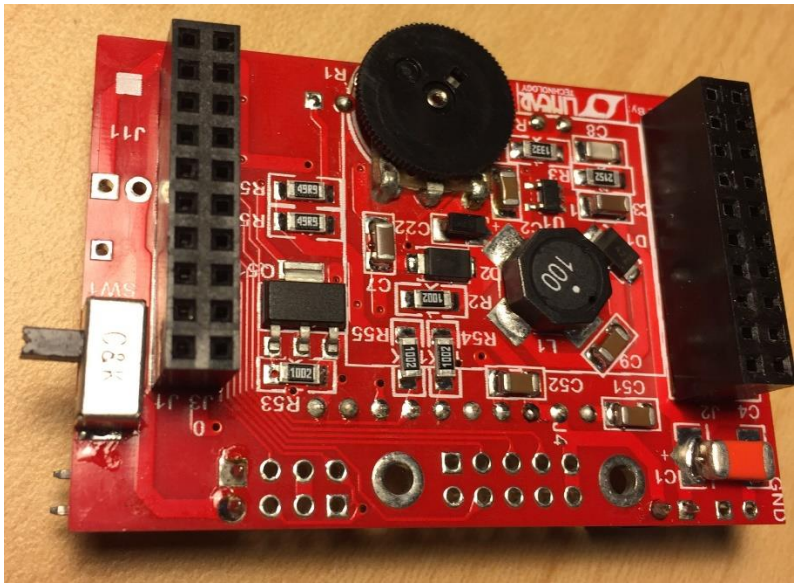


Figure 4.4 Buck-Boost Converter Circuit

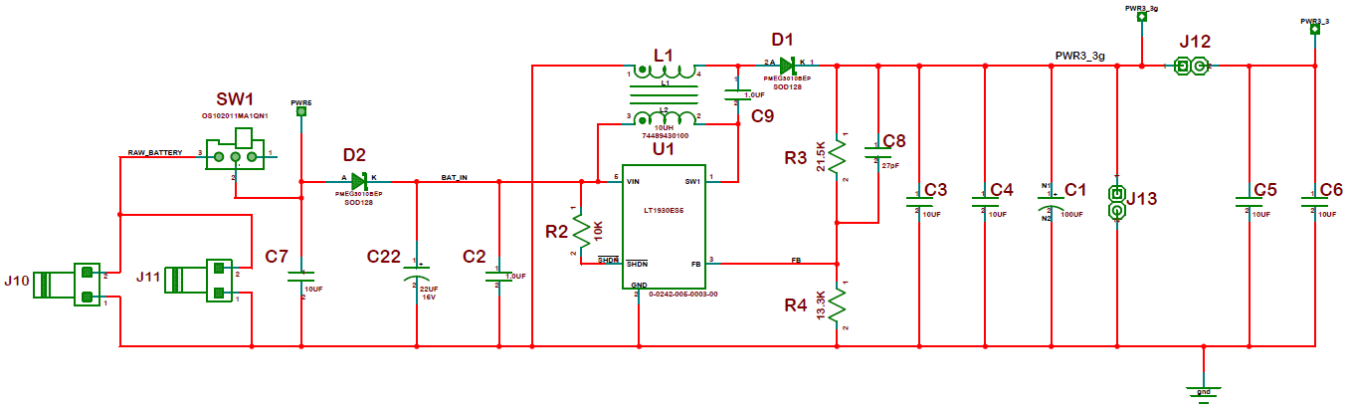


Figure 4.5 Buck-Boost Converter (DC Batteries to Car Microcontroller)

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.

4.3. Microcontroller

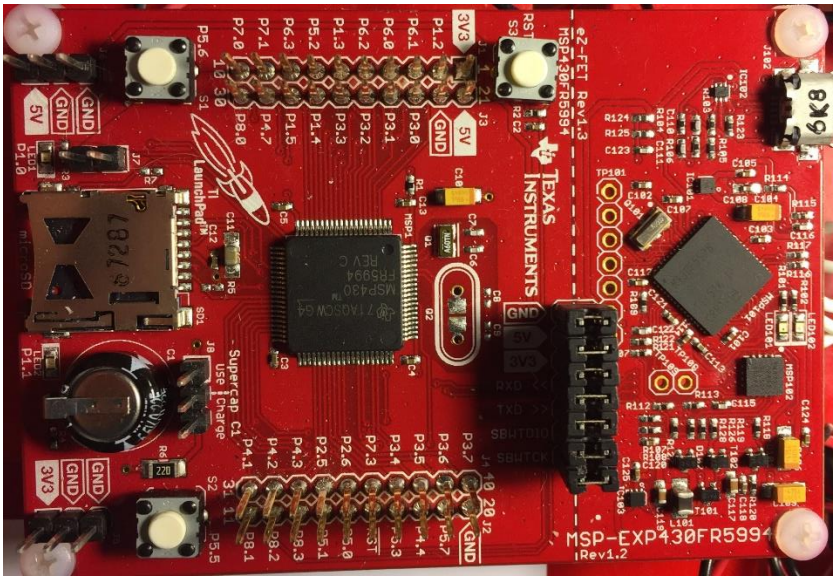


Figure 4.6 MSP430 Microcontroller Board

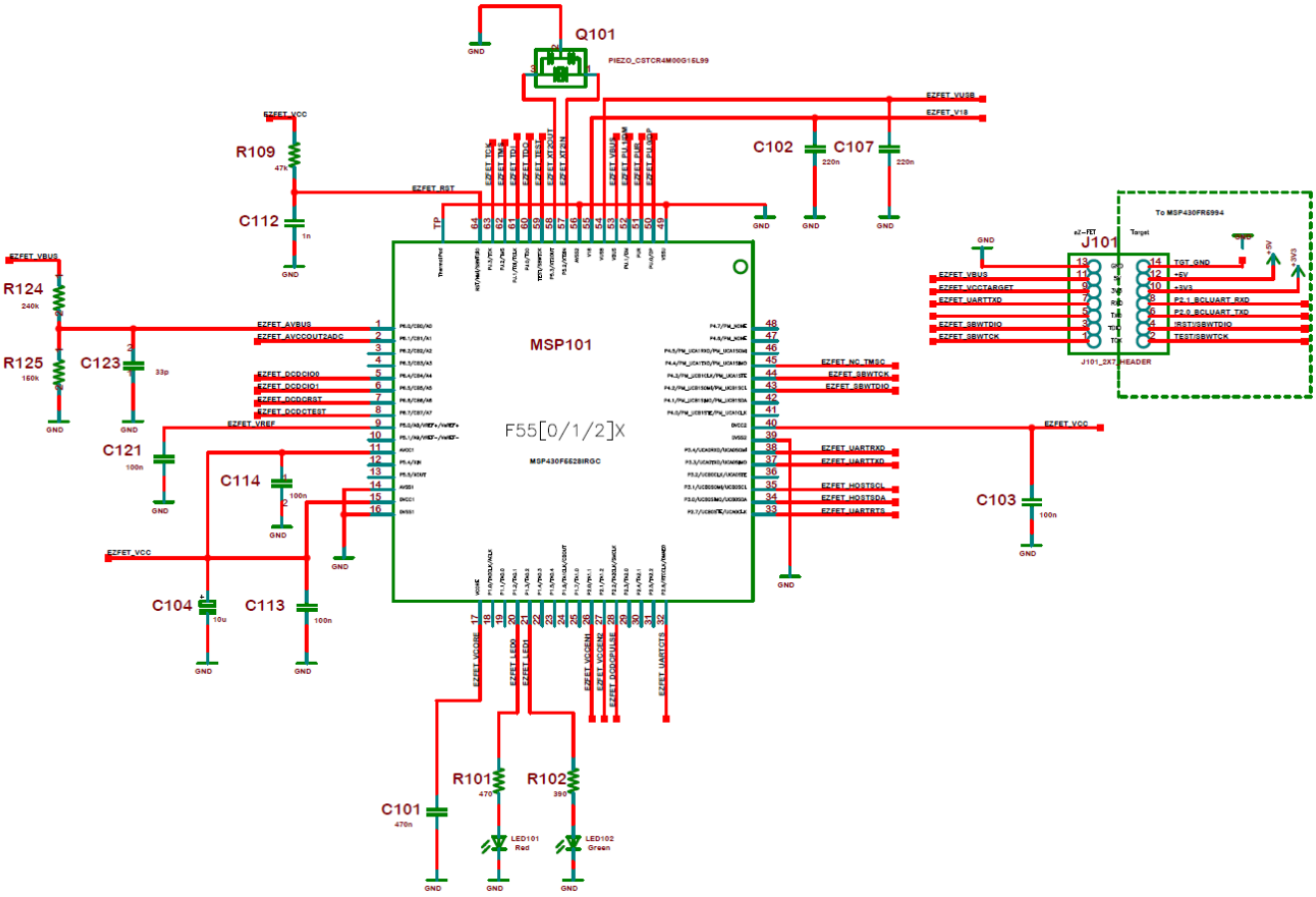


Figure 4.7 MSP430 Microcontroller

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.

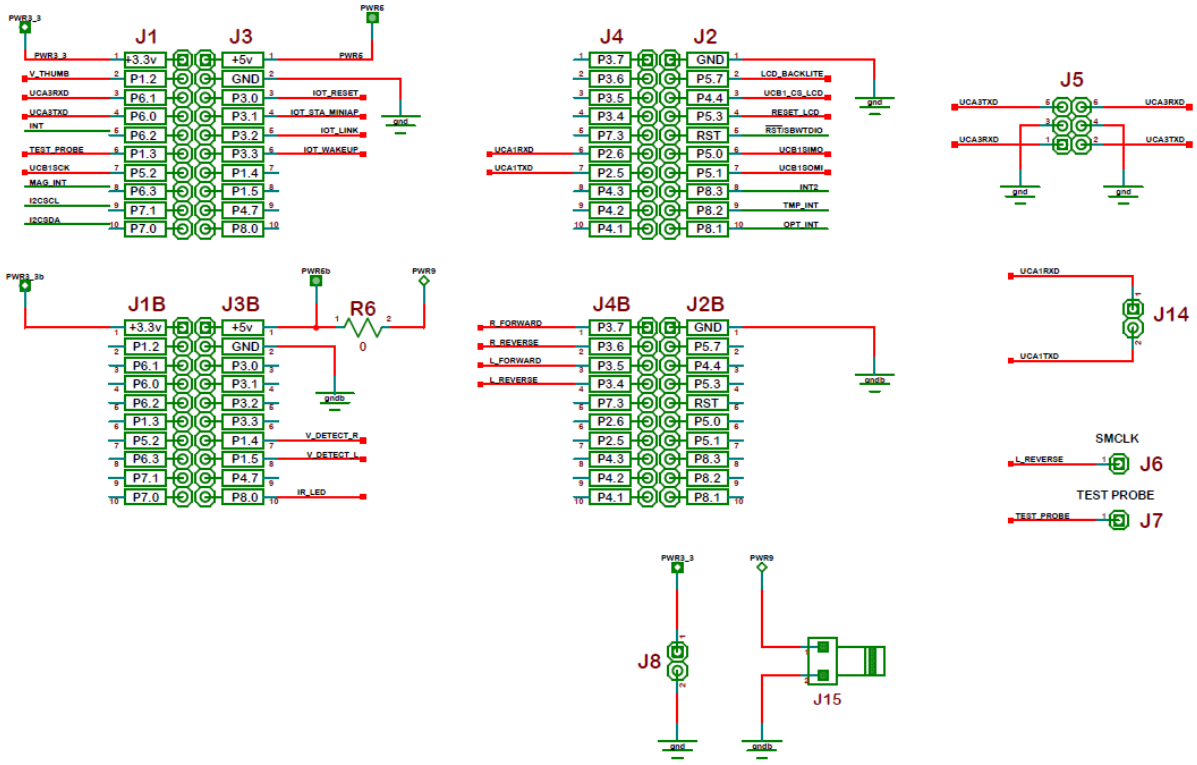


Figure 4.8 Microcontroller Pin Configuration

4.4. Motor Control Board

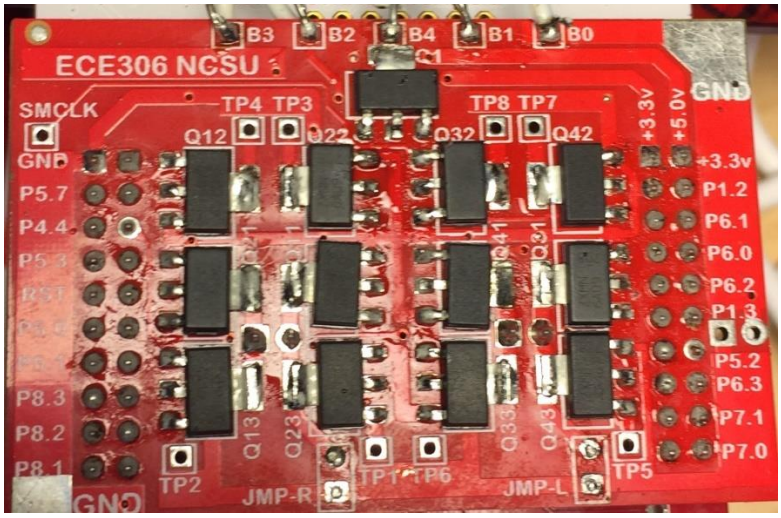


Figure 4.9 Motor Control PCB

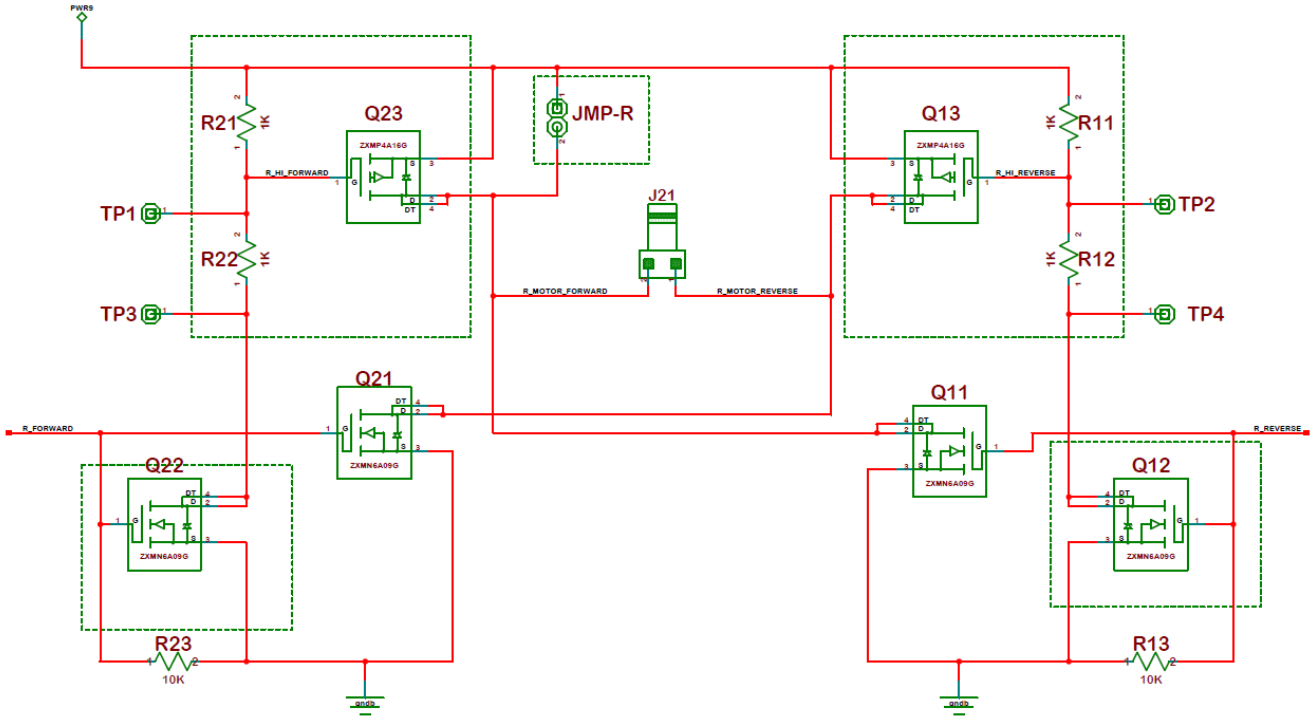


Figure 4.10 H-Bridge for Right Motor (mirror left H-Bridge schematic)

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.

4.5. Black Line Detection

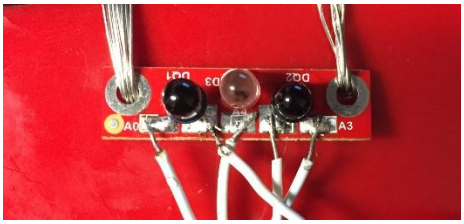


Figure 4.11 Infrared LED Emitter (center) and Detectors (left and right)

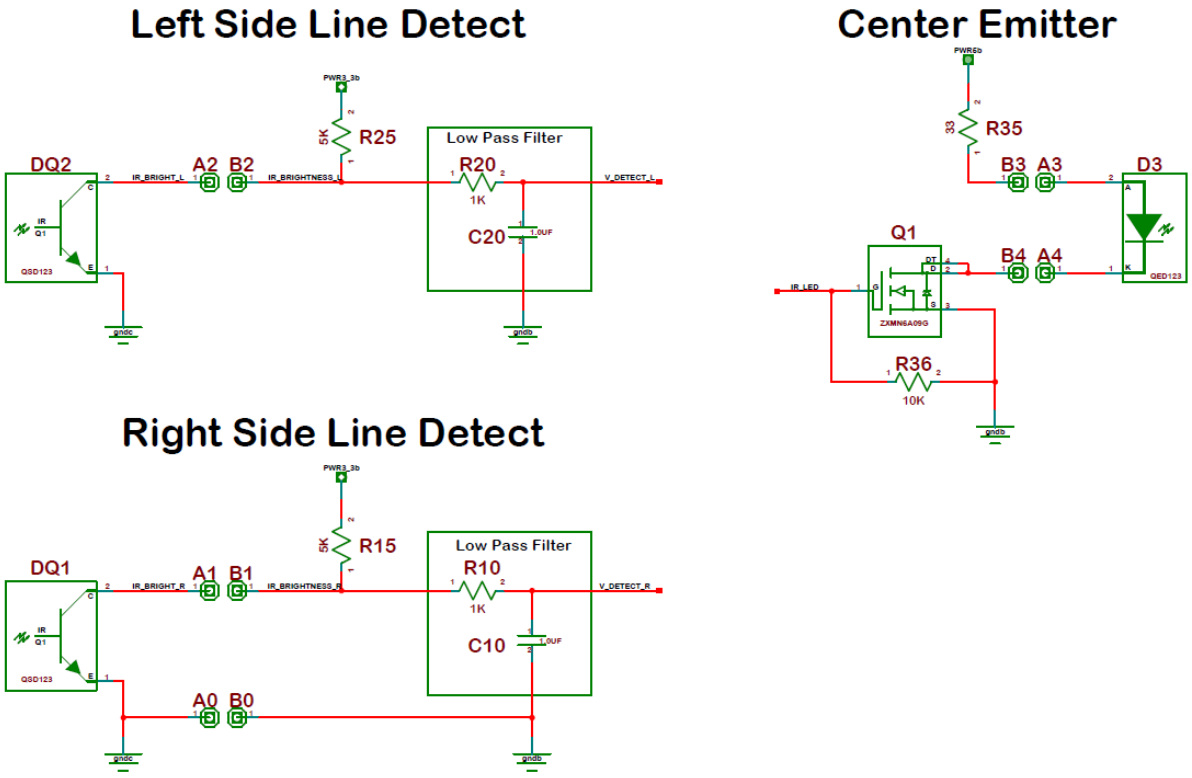


Figure 4.12 Black Line Detection Hardware

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.

6. 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.

6.1. 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.

6.2. 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.

6.3. 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.

6.4. 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.

6.5. 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.

6.6. 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.

6.7. 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.

6.8. 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.

6.9. 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.

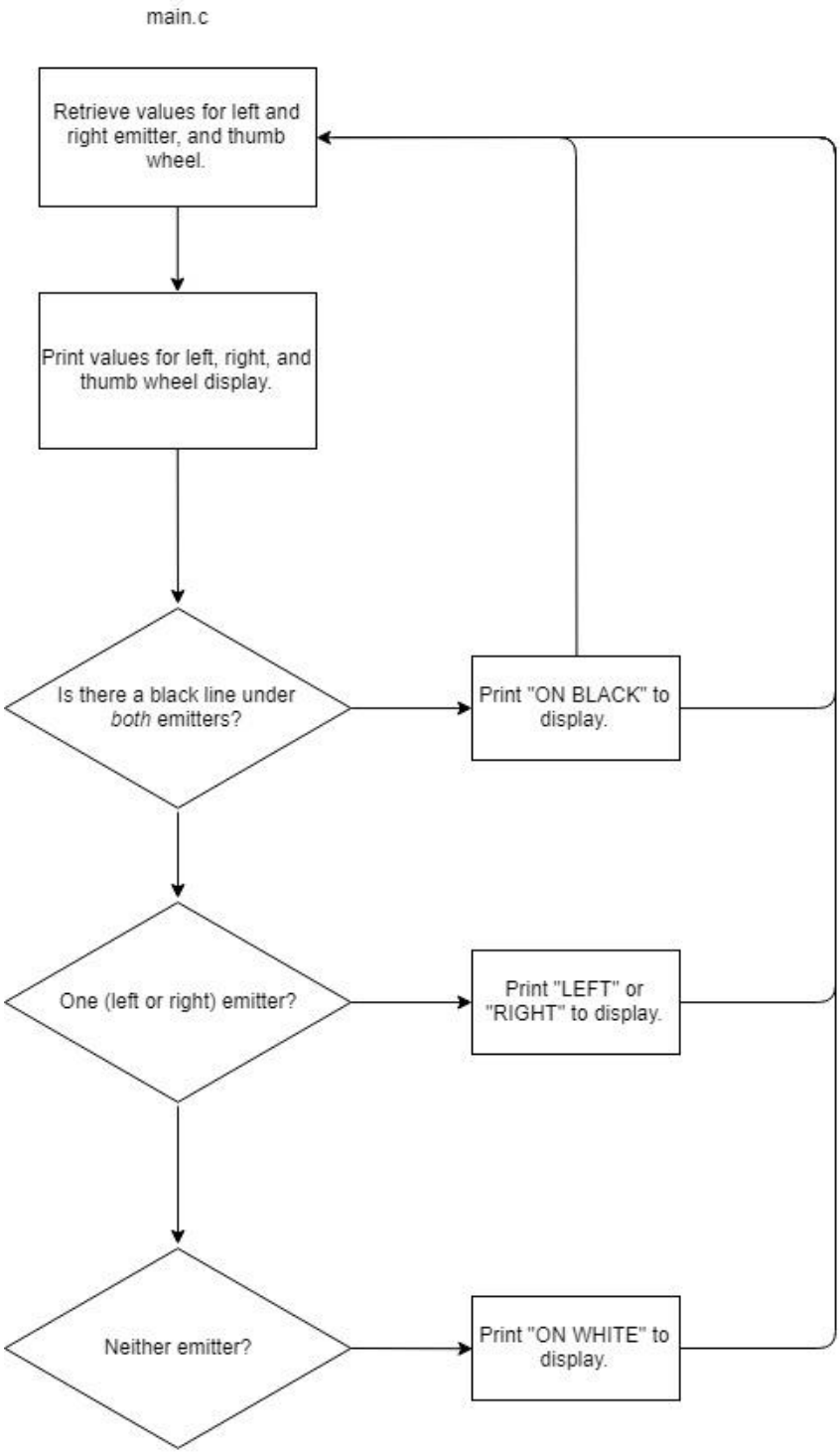
7. Software

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 – its 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.

8. Flow Chart

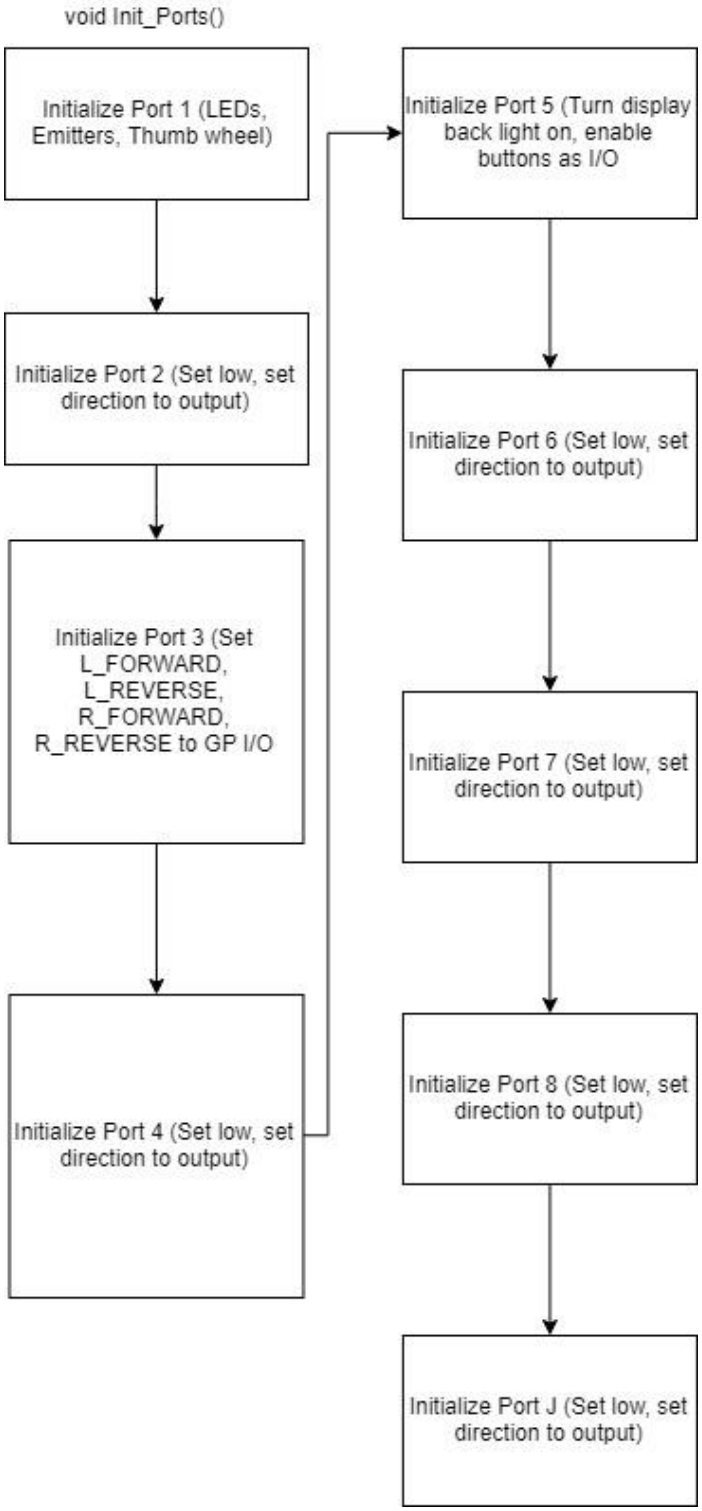
The following is a series of flow charts describing our code.

8.1. Main Blocks



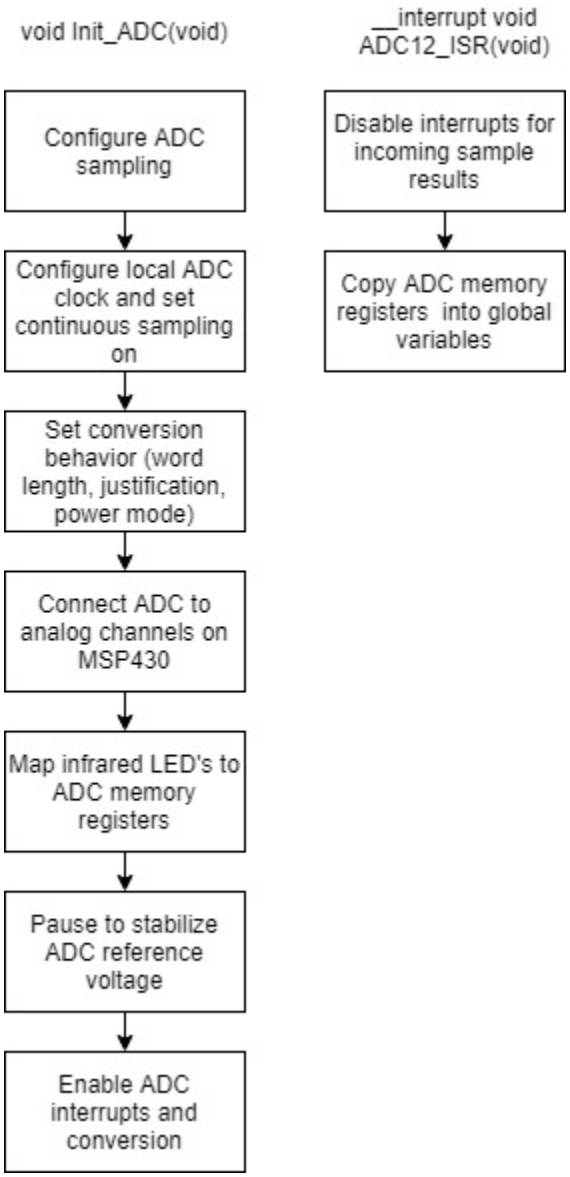
In main.c, we retrieve values from the left and right emitter, and the thumbwheel, and based on our outputs, we output the proper text to the display.

8.2. Port Initialization Blocks



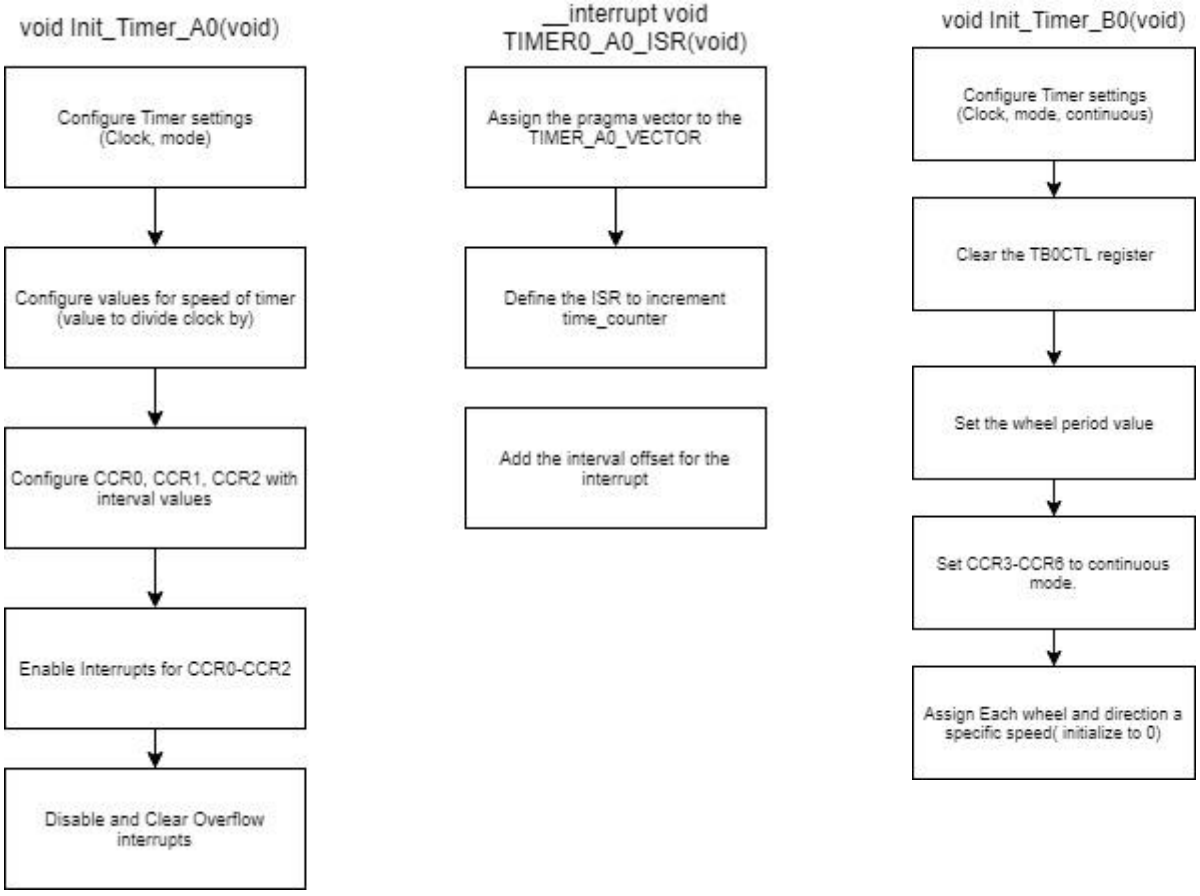
In Init_Ports(), we initialize all 9 ports (1-8, J) by assigning each individual bit of each port a specified value. Ports 2, 4, 6, 7, 8, and J are all set low.

8.3. 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.

8.4. Timer 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.

9. Software Listing

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

9.1. Main.c

```
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
  Button_Enable();
  PM5CTL0 &= ~LOCKLPM5;
  P1OUT |= RED_LED;
  P1OUT |= GRN_LED;
  Init_Clocks();                  // Initialize Clock System
  Init_Conditions();              // Initialize Variables and Initial Conditions
  Init_Timers();
  Init_LCD();                     // Initialize LCD
  Init_ADC(); // Initialize ADC

  strcpy(display_line[DISPLAY_ARRA_0], "  NCSU  ");
  update_string(display_line[DISPLAY_ARRA_0], DISPLAY_ARRA_0);
  strcpy(display_line[DISPLAY_ARRA_1], " WOLFPACK ");
  update_string(display_line[DISPLAY_ARRA_1], DISPLAY_ARRA_1);
  strcpy(display_line[DISPLAY_ARRA_2], " ECE-306 ");
  update_string(display_line[DISPLAY_ARRA_3], DISPLAY_ARRA_3);
  Display_Update(DISPLAY_ARRA_3,DISPLAY_ARRA_1,DISPLAY_ARRA_0,DISPLAY_ARRA_0);

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

    if(ADC_Left_Detector > BLACK_LINE && ADC_Right_Detector > BLACK_LINE)
```

```
      display_black_LR(BOTH);

    else if(ADC_Left_Detector  > BLACK_LINE  && ADC_Right_Detector  < BLACK_LINE)
      display_black_LR(LEFT_HIGH);

    else if (ADC_Left_Detector  < BLACK_LINE  && ADC_Right_Detector  > BLACK_LINE)
      display_black_LR(RIGHT_HIGH);

    else if(ADC_Left_Detector  < BLACK_LINE  && ADC_Right_Detector  < BLACK_LINE)
      display_black_LR(NONE_DETECTED);

    Display_Process();
}
//-------------------------------------------------------------------------
}
```

9.2. 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 ;
}
```

9.3. 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

}
```

9.4. Ports.c

```
void Init_Ports(void) // initializes all ports
{
  Init_Ports_1();
  Init_Ports_2();
  Init_Ports_3();
  Init_Ports_4();
```

```
  Init_Ports_5();
  Init_Ports_6();
  Init_Ports_7();
  Init_Ports_8();
  Init_Ports_J();
}
void Init_Ports_1(void)
{
  //Configure Port 1
  P1SEL0 = CLEAR; // GP I/O
  P1SEL1 = CLEAR; // GP I/O
  P1DIR = CLEAR; // Set P1 direction to input
  // P1_0
  P1SEL0 &= ~RED_LED; // RED_LED as GP I/O
  P1SEL1 &= ~RED_LED; // RED_LED as GP I/O
  P1OUT |= RED_LED; // Set Red LED On
  P1DIR |= RED_LED; // Set Red LED direction to output
  // P1_1
  P1SEL0 &= ~GRN_LED; // GRN_LED as GP I/O
  P1SEL1 &= ~GRN_LED; // GRN_LED as GP I/O
  P1OUT |= GRN_LED; // Set Green LED On
  P1DIR |= GRN_LED; // Set Green LED direction to output
  // P1_2
  P1SEL0 |= V_THUMB; // ADC input for Thumbwheel
  P1SEL1 |= V_THUMB; // ADC input for Thumbwheel
  // P1_3
  P1SEL0 &= ~TEST_PROBE; // TEST_PROBE as GP I/O
  P1SEL1 &= ~TEST_PROBE; // TEST_PROBE as GP I/O
  P1OUT &= ~TEST_PROBE; // Set TEST_PROBE Off
  P1DIR |= TEST_PROBE; // Set TEST_PROBE direction to output
  // P1_4
  P1SEL0 |= V_DETECT_R; // ADC input for Right Detector
  P1SEL1 |= V_DETECT_R; // ADC input for Right Detector
  // P1_5
  P1SEL0 |= V_DETECT_L; // ADC input for Left Detector
  P1SEL1 |= V_DETECT_L; // ADC input for Left Detector
  // P1_6
  P1SEL0 &= ~SD_UCB0SIMO; // USCI_B1 MOSI pin
```

```
  P1SEL1 |= SD_UCB0SIMO;  // USCI_B1  MOSI pin
  // P1_7
  P1SEL0 &= ~SD_UCB0SOMI;  // USCI_B1  MISO pin
  P1SEL1 |= SD_UCB0SOMI;  // USCI_B1  MISO pin
}
void Init_Ports_2(void)
{
  P2OUT = CLEAR; // P2 set low
  P2DIR = SET_ALL; // set P2 direction to output

  //P2_0
  P2SEL0 &= ~UCA0TXD;
  P2SEL1 |= UCA0TXD;
  P2OUT &= ~UCA0TXD;
  P2DIR |= UCA0TXD;

  //P2_1
  P2SEL0 &= ~UCA0RXD;
  P2SEL1 |= UCA0RXD;
  P2OUT &= ~UCA0RXD;
  P2DIR |= UCA0RXD;

  //P2_2
  P2SEL0 &= ~UCB0CLK ;
  P2SEL1 |= UCB0CLK;
  P2OUT &= ~UCB0CLK;
  P2DIR |= UCB0CLK;

  //P2_3
  P2SEL0 &= ~P2_3;
  P2SEL1 &= ~P2_3;
  P2OUT &= ~P2_3;
  P2DIR |= P2_3;

  //P2_4
  P2SEL0 &= ~P2_4;
  P2SEL1 &= ~P2_4;
  P2OUT &= ~P2_4;
```

```
  P2DIR |= P2_4;

  //P2_5
  P2SEL0 &= ~UCA1TXD;
  P2SEL1 |= UCA1TXD;
  P2OUT &= ~UCA1TXD;
  P2DIR |= UCA1TXD;

  //P2_6
  P2SEL0 &= ~UCA1RXD;
  P2SEL1 |= UCA1RXD;
  P2OUT &= ~UCA1RXD;
  P2DIR |= UCA1RXD;

  //P2_7
  P2SEL0 &= ~P2_7;
  P2SEL1 &= ~P2_7;
  P2OUT &= ~P2_7;
  P2DIR |= P2_7;
}
void Init_Ports_3(void)
{
  P3DIR = SET_ALL; // Set P1 direction to output
  P3OUT = CLEAR; // P1 set Low

  P3SEL0 &= ~IOT_RESET;  // Set to GP I/O
  P3SEL1 &= ~IOT_RESET;  // Set to GP I/O
  P3OUT &= ~IOT_RESET;  // Set out value Low [active]
  P3DIR |= IOT_RESET;  // Set direction to output

  P3SEL0 &= ~IOT_STA_MINIAP;  // Set to GP I/O
  P3SEL1 &= ~IOT_STA_MINIAP;  // Set to GP I/O
  P3OUT |= IOT_STA_MINIAP;  // Set out value no Mini AP
  P3DIR |= IOT_STA_MINIAP;  // Set direction to output

  P3SEL0 &= ~IOT_WAKEUP;  // Set to GP I/O
  P3SEL1 &= ~IOT_WAKEUP;  // Set to GP I/O
  P3OUT &= ~IOT_WAKEUP;  // Set out value Low [off]
```

```
  P3DIR |= IOT_WAKEUP; // Set direction to output

  P3SEL0 &= ~IOT_FACTORY;  // Set to GP I/O
  P3SEL1 &= ~IOT_FACTORY;  // Set to GP I/O
  P3OUT &= ~IOT_FACTORY;  // Set out value Low [off]
  P3DIR |= IOT_FACTORY;  // Set direction to output

  P3SEL0 &= ~L_REVERSE; // Set to GP I/O
  P3SEL1 &= ~L_REVERSE; // Set to GP I/O
  P3OUT &= ~L_REVERSE; // Set out value Low [off]
  P3DIR |= L_REVERSE; // Set direction to output

  P3SEL0 &= ~L_FORWARD; // Set to GP I/O
  P3SEL1 &= ~L_FORWARD; // Set to GP I/O
  P3OUT &= ~L_FORWARD; // Set out value Low [off]
  P3DIR |= L_FORWARD; // Set direction to output

  P3SEL0 &= ~R_REVERSE; // Set to GP I/O
  P3SEL1 &= ~R_REVERSE; // Set to GP I/O
  P3OUT &= ~R_REVERSE; // Set out value Low [off]
  P3DIR |= R_REVERSE; // Set direction to output

  P3SEL0 &= ~R_FORWARD; // Set to GP I/O
  P3SEL1 &= ~R_FORWARD; // Set to GP I/O
  P3OUT &= ~R_FORWARD; // Set out value Low [off]
  P3DIR |= R_FORWARD; // Set direction to output
}
void Init_Ports_4(void)
{
  P4OUT = CLEAR; // P4 set low
  P4DIR = SET_ALL; // set P4 direction to output

  //P4_0
  P4SEL0 &= ~SD_CS;
  P4SEL1 &= ~SD_CS;
  P4OUT &= ~SD_CS;
  P4DIR |= SD_CS;
```

```
//P4_1
P4SEL0 &= ~J4_31;
P4SEL1 &= ~J4_31;
P4OUT  &= ~J4_31;
P4DIR  |= J4_31;

//P4_2
P4SEL0 &= ~J4_32 ;
P4SEL1 &= ~J4_32;
P4OUT  &= ~J4_32;
P4DIR  |= J4_32;

//P4_3
P4SEL0 &= ~J4_33;
P4SEL1 &= ~J4_33;
P4OUT  &= ~J4_33;
P4DIR  |= J4_33;

//P4_4
P4SEL0 &= ~UCB1_CS_LCD;
P4SEL1 &= ~UCB1_CS_LCD;
P4OUT  &= ~UCB1_CS_LCD;
P4DIR  |= UCB1_CS_LCD;

//P4_5
P4SEL0 &= ~P4_4;
P4SEL1 &= ~P4_4;
P4OUT  &= ~P4_4;
P4DIR  |= P4_4;

//P4_6
P4SEL0 &= ~P4_5;
P4SEL1 &= ~P4_5;
P4OUT  &= ~P4_5;
P4DIR  |= P4_5;

//P4_7
P4SEL0 &= ~J3_29;
```

```
  P4SEL1 &= ~J3_29;
  P4OUT  &= ~J3_29;
  P4DIR  |= J3_29;

}
void  Init_Ports_5(void)
{
  P5OUT  = CLEAR; // P5 set low
  P5DIR  = CLEAR; // set P5 direction  to output

  //P5_0
  P5SEL0 |= SPI_UCB1SIMO;
  P5SEL1 &= ~SPI_UCB1SIMO;

  //P5_1
  P5SEL0 |= SPI_UCB1SOMI;
  P5SEL1 &= ~SPI_UCB1SOMI;

  //P5_2
  P5SEL0 |= SPI_UCB1CLK  ;
  P5SEL1 &= ~SPI_UCB1CLK;

  //P5_3
  P5SEL0 &= ~RESET_LCD;
  P5SEL1 &= ~RESET_LCD;
  P5OUT  |= RESET_LCD;
  P5DIR  |= RESET_LCD;

  //P5_4
  P5SEL0 &= ~P5_4;
  P5SEL1 &= ~P5_4;
  P5DIR  &= ~P5_4;

  P5SEL0 = CLEAR; // P5 set as I/0
  P5SEL1 = CLEAR; // P5 set as I/0
  P5DIR  = CLEAR; // Set P5 direction to input
  P5SEL0 &= ~BUTTON1;  // BUTTON1  set as I/0
  P5SEL1 &= ~BUTTON1;  // BUTTON1  set as I/0
```

```
  P5DIR &= ~BUTTON1;  // BUTTON1  Direction = input
  P5OUT |= BUTTON1;  // Configure  pull-up resistor BUTTON1
  P5REN |= BUTTON1;  // Enable pull-up resistor BUTTON1

  P5SEL0 &= ~BUTTON2;  // BUTTON2  set as I/0
  P5SEL1 &= ~BUTTON2;  // BUTTON2  set as I/0
  P5DIR &= ~BUTTON2;  // BUTTON2  Direction = input
  P5OUT |= BUTTON2;  // Configure  pull-up resistor BUTTON2
  P5REN |= BUTTON2;  // Enable pull-up resistor BUTTON2

  //P5_7
  P5SEL0 &= ~LCD_BACKLITE;
  P5SEL1 &= ~LCD_BACKLITE;
  P5OUT  &= ~LCD_BACKLITE;
  P5DIR |= LCD_BACKLITE;

}

void Init_Ports_6(void)
{
  P6OUT  = CLEAR; // P6 set low
  P6DIR = SET_ALL; // set P6 direction to output

  //P6_0
  P6SEL0 &= ~UCA3TXD;
  P6SEL1 &= ~UCA3TXD;
  P6OUT  &= ~UCA3TXD;
  P6DIR |= UCA3TXD;

  //P6_1
  P6SEL0 &= ~UCA3RXD;
  P6SEL1 &= ~UCA3RXD;
  P6OUT  &= ~UCA3RXD;
  P6DIR |= UCA3RXD;

  //P6_2
  P6SEL0 &= ~J1_5 ;
  P6SEL1 &= ~J1_5;
```

```
  P6OUT &= ~J1_5;
  P6DIR |= J1_5;

  //P6_3
  P6SEL0 &= ~MAG_INT;
  P6SEL1 &= ~MAG_INT;
  P6OUT &= ~MAG_INT;
  P6DIR |= MAG_INT;

  //P6_4
  P6SEL0 &= ~P6_4;
  P6SEL1 &= ~P6_4;
  P6OUT &= ~P6_4;
  P6DIR |= P6_4;

  //P6_5
  P6SEL0 &= ~P6_5;
  P6SEL1 &= ~P6_5;
  P6OUT &= ~P6_5;
  P6DIR |= P6_5;

  //P6_6
  P6SEL0 &= ~P6_6;
  P6SEL1 &= ~P6_6;
  P6OUT &= ~P6_6;
  P6DIR |= P6_6;

  //P6_7
  P6SEL0 &= ~P6_7;
  P6SEL1 &= ~P6_7;
  P6OUT &= ~P6_7;
  P6DIR |= P6_7;

}
```

```
void Init_Ports_7(void)
{
  P7OUT = CLEAR; // P7 set low
  P7DIR = SET_ALL; // set P7 direction to output

  //P7_0
  P7SEL0 &= ~I2CSDA;
  P7SEL1 &= ~I2CSDA;
  P7OUT &= ~I2CSDA;
  P7DIR |= I2CSDA;

  //P7_1
  P7SEL0 &= ~I2CSCL;
  P7SEL1 &= ~I2CSCL;
  P7OUT &= ~I2CSCL;
  P7DIR |= I2CSCL;

  //P7_2
  P7SEL0 &= ~SD_DETECT ;
  P7SEL1 &= ~SD_DETECT;
  P7OUT &= ~SD_DETECT;
  P7DIR |= SD_DETECT;

  //P7_3
  P7SEL0 &= ~J4_36;
  P7SEL1 &= ~J4_36;
  P7OUT &= ~J4_36;
  P7DIR |= J4_36;

  //P7_4
  P7SEL0 &= ~P7_4;
  P7SEL1 &= ~P7_4;
  P7OUT &= ~P7_4;
  P7DIR |= P7_4;

  //P7_5
  P7SEL0 &= ~P7_5;
```

```
  P7SEL1 &= ~P7_5;
  P7OUT  &= ~P7_5;
  P7DIR  |= P7_5;

  //P7_6
  P7SEL0 &= ~P7_6;
  P7SEL1 &= ~P7_6;
  P7OUT  &= ~P7_6;
  P7DIR  |= P7_6;

  //P7_7
  P7SEL0 &= ~P7_7;
  P7SEL1 &= ~P7_7;
  P7OUT  &= ~P7_7;
  P7DIR  |= P7_7;

}
void  Init_Ports_8(void)
{
  P8OUT  = CLEAR; // P8 set low
  P8DIR  = SET_ALL; // set P8 direction to output

  //P8_0
  P8SEL0 &= ~IR_LED;
  P8SEL1 &= ~IR_LED;
  P8OUT  &= ~IR_LED;
  P8DIR  |= IR_LED;

  //P8_1
  P8SEL0 &= ~OPT_INT;
  P8SEL1 &= ~OPT_INT;
  P8OUT  &= ~OPT_INT;
  P8DIR  |= OPT_INT;

  //P8_2
  P8SEL0 &= ~TMP_INT  ;
  P8SEL1 &= ~TMP_INT;
  P8OUT  &= ~TMP_INT;
```

```
  P8DIR |= TMP_INT;

  //P8_3
  P8SEL0 &= ~INT2;
  P8SEL1 &= ~INT2;
  P8OUT  &= ~INT2;
  P8DIR |= INT2;

}
void Init_Ports_J(void)
{
  PJOUT  = CLEAR; // PJ set low
  PJDIR  = SET_ALL; // set PJ direction to output

  //LFXIN
  PJSEL0 |= LFXIN;
  PJSEL1 &= ~LFXIN;

  //LFXOUT
  PJSEL0 |= LFXOUT;
  PJSEL1 &= ~LFXOUT;

  //HFXIN
  PJSEL0 |= HFXIN;
  PJSEL1 &= ~HFXIN;

  //HFXOUT
  PJSEL0 |= HFXOUT;
  PJSEL1 &= ~HFXOUT;

}
```

9.5. 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;
}
```