Car Simulation

By: Parth Patel

ECE 471 Project Report

School of Electrical and Computer Engineering

April 2021

Table of Contents

List of Figures

Introduction

Proposal

The purpose of this project is to use the PIC18F4331 microcontroller to create a car simulation. An LCD will be used to display the speed of the car. A potentiometer will be used to control the speed of the car. Push buttons will be used to act as an ignition button for the car and a reset for the simulation. There will be 5 total leds used as a stoplight and police lights. These lights will be controlled with a timer. The simulation will end if the vehicle is moving during a red light or the vehicle moves over the speed limit for an extended period of time. Depending on how much over the speed limit the vehicle is going the simulation will end faster.

Peripherals Used

- LCD screen
    - LCD displays speed in real time from 0-5
- Timer0 (satisfies interrupt requirement)
    - Controls both the stoplight and police lights. The colors of the stoplight change at intervals so the light is red the longest, green the second longest, and yellow for the shortest amount of time similar to a real stoplight. If the vehicle speeds or runs a red light the simulation ends with the police sirens.
- ADC (Potentiometer)

    - A variable 10k Ohm potentiometer is used to control the speed from 0-5 speed units. For this simulation 0 is not moving and 3 is the speed limit for the simulation.

Project Requirements

The general requirements for this project include:

- Each student works on his/her own project
- The project must use the PIC18F4331
- The hardware must be built using the PIC microcontroller chip. Premade PIC circuit boards will not be accepted.
- In addition to general purpose IO, each student must use the interrupt and three peripherals components such as A/D,PWM, timer, synchronous or asynchronous communication. (Note, interrupt requirement is satisfied if it is used as part of peripheral components.)

Design Details

Hardware Design

- LCD
    - The LCD uses 16 total wires. 8 of which are data wires connected to B0-B7. RD2-RD4 are used for E, RW, and RS. The other wires are used for positive voltage, ground, and contrast which was also grounded for max contrast.
- Potentiometer
    - The potentiometer is connected to AN4 and configured for ADC.
- Stoplight LEDs

    - RA0 has the red led, RA1 has the yellow led, and RA2 has the green led.
- Start Button
    - RC3 has the push button used as the ignition button for the simulation.
- Police Siren LEDs
    - RD0 has the red police light and RD1 has the blue police light.
- Reset Button
    - RC4 has the push button used for resetting the simulation.

Software Design

LCD functions

LCD_clock

Params in: void

Params out: void

Description: sets enable high then low to create a falling edge trigger.

LCD_command

Params in: void

Params out: void

Description: sets RS and RW pins to 0 which does not update the LCD screen

LCD_command_write

Params in: void

Params out: void

Description: updates the LCD and prints since last update

LCD_init_8bits

Params in: void

Params out: void

Description: Initializes LCD by sending the startup sequence: 0x38 (2 line mode), 0x0C hides the cursor, 0x06 sets increment to right, 0x01 set the cursor to first position

LCD_set_pos

Params in: 8 bit int, 8 bit int

Params out: void

Description: takes in an row and column number and set cursor to that position. If the first row is used then the column is or’s with 80, and if the second row is selected then 40 is added to the column.

LCD_print

Params in: 8 bit int

Params out: void

Description: RS is set to 1 then the char is used to set that output pin to that char

LCD_print_string

Params in: constant char *string

Params out: void

Description: sends a string character by character to the LCD_print function

reset_sim

Params in: void

Params out: void

Description: checks reset flag and if flag is set to 1 then police lights flash and an infinite loop is set that can only be broken by pressing the reset button. Once the button is pressed the variables are reset and there is a delay before the program resumes.

stoplight

Params in: void

Params out: void

Description: checks to see if flag ==1 (set by timer every second) so every second the light count is decremented and depending on the value the light changes colors. This is called from the timer. After light value is checked the speed value is checked and if the user ran a red then reset is called. If they are speeding the speed count goes up by 1 if the speed is 4 and up by 2 if speed is 5. Speed count can be reset by going the speed limit. If the speed count reaches the max value then the reset function is called.

Timer0

Params in: void

Params out: void

Description: Every second flag is set 1 and light count is decremented before calling stoplight.

Main

Params in: void

Params out: void

Description: waits for the start button to be pressed. Once it is pressed then the potentiometer is read and converted to a value from 0-5. This value is stored in speed value and depending on the value a different line is printed to the LCD.

Implementation

List of Parts

- PIC18F4331
- 1602A 16x2 LCD
- 5 LEDs (2 Red, 1 Yellow, 1 Green, 1 Blue)
- 10k Potentiometer
- 2 Push Buttons
- External power supply

Layout Diagram

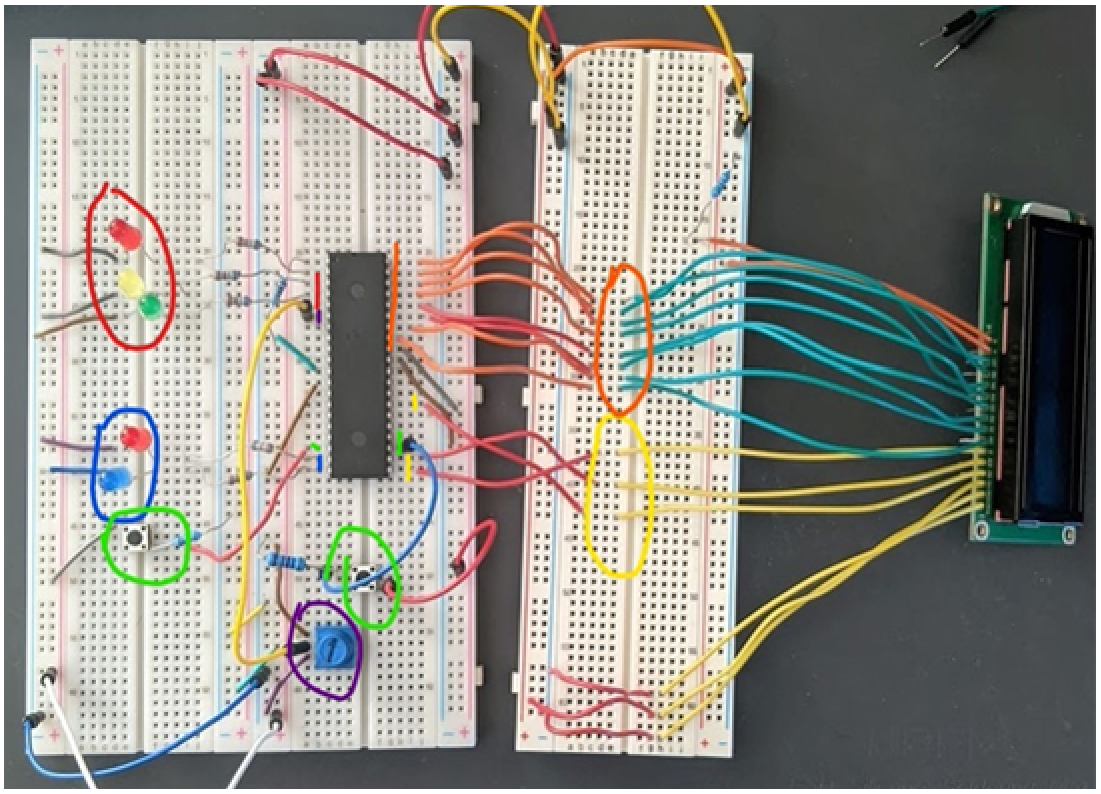

Figure 1 Circuit Diagram

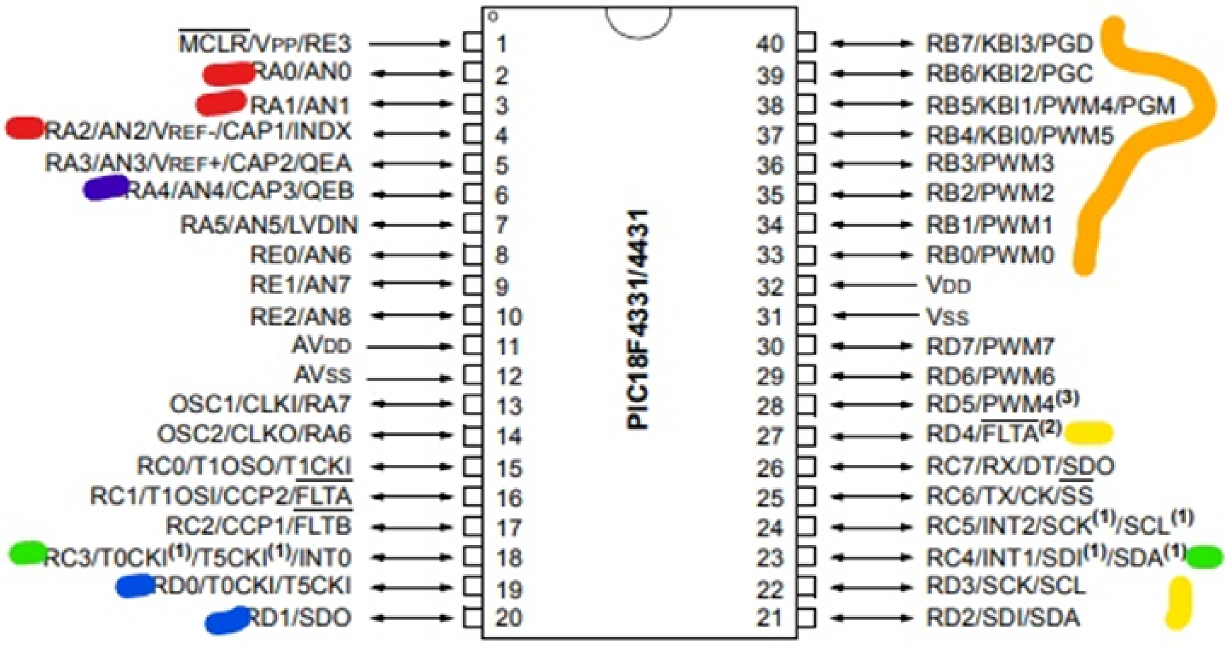

Figure 2 Pin layout from datasheet with same labeling as Figure 1

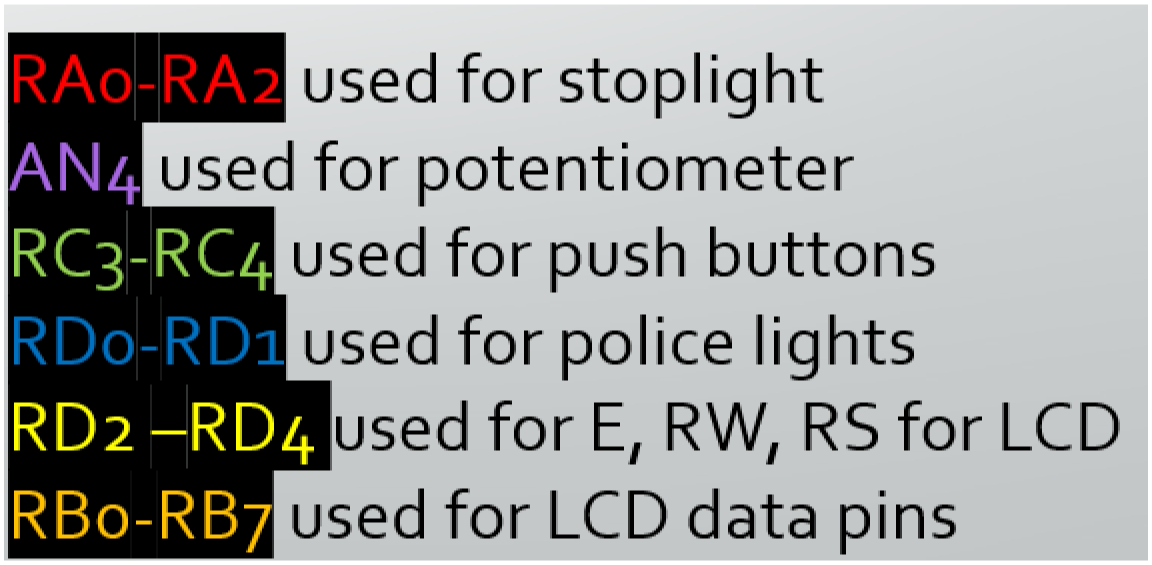
RA0-RA2 used for stoplight
AN4 used for potentiometer
RC3-RC4 used for push buttons
RD0-RD1 used for police lights
RD2 –RD4 used for E, RW, RS for LCD
RB0-RB7 used for LCD data pins

Figure 3 Legend for Figure 1 and Figure 2

Software Tools

- MPLABX IDE v5.45
    - IDE provided by Microchip for PIC microcontrollers.
- XC8 Compiler
    - This is the compiler provided by Microchip for 8-bit PIC microcontrollers.

Experiment Results

Results

The final results from this project is a working car simulation that will end when the car crashes or is pulled over. The simulation is started with the start button and the potentiometer is used to control the speed. This speed is sent to the LCD in real time. The stoplight changes between colors on a timer. If the car goes above the speed limit of 3 then they get pulled over after an amount of time depending on how fast they were going. Going at speed 5 gets the driver pulled over faster than a speed of 4. The LCD also displays end messages depending on if the simulation ended with speeding or crashing due to running a red light.

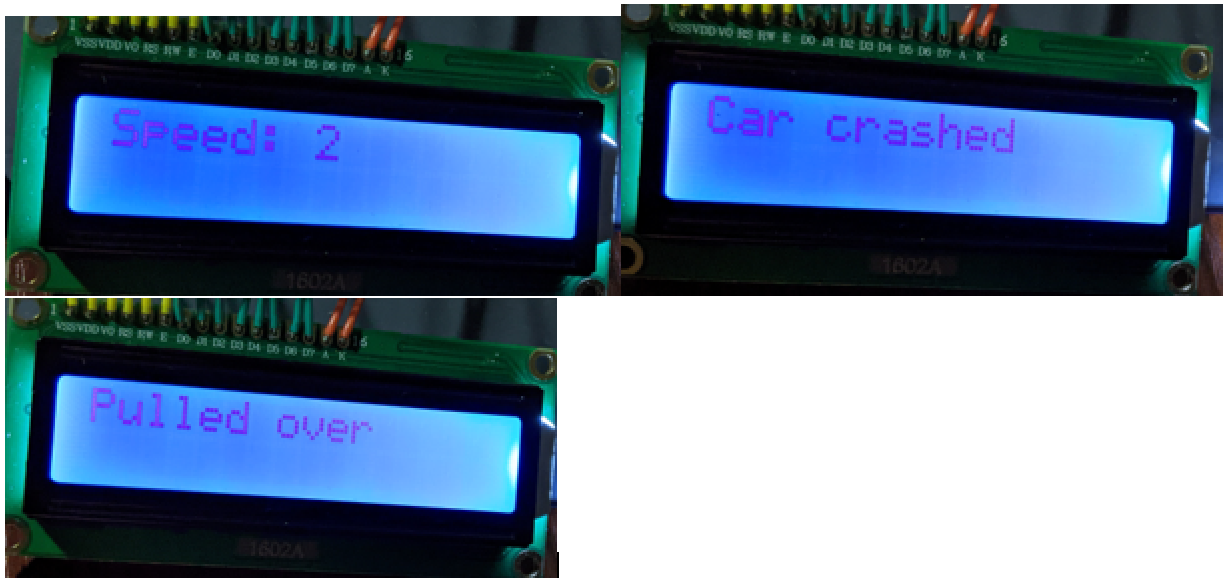

Figure 4 LCD Printouts

Requirements Satisfied and Not Satisfied

All the requirements were met, however the reset button acts as a pseudo reset button instead of a full reset. In its current state the reset button resets all the values and restarts the reading instead of stopping and restarting the simulation. The start button is no longer needed after the reset button. The three required peripherals all work as intended: the potentiometer, timer0, and the LCD.

Discussion

Software

For the software each peripheral was written and tested separately before being combined together into the final version. Each peripheral is split into functions with some peripherals using the same functions. Outside of the LCD, the other peripherals were done similarly to the previous work in the class with modifications made for this project. The LCD required more research to find a compatible LCD screen since the PIC18F4331 does not support I2C. After finding a compatible LCD the datasheet for it was used to configure it to work with the rest of the project.

For the timer, initially 2 timers were going to be used but after receiving feedback on the proposal the functionality was combined into Timer0. Timer0 was used since it was used in previous projects and it has the most functionality since it supports 16 bit operations. This timer would increment values for the speed counter and stoplight counter every second. The speed counter would only be incremented if the car was speeding and was reset if the speed was acceptable. The light counter always incremented and at certain times the color would change before resetting. If at any point the speed counter reached a max value the reset function would be called.

For analog to digital conversion, a 10k ohm potentiometer was used. This potentiometer would read in a value then convert into a value from 0-5 so it can be used by the lcd and speed variables for various functions.

Hardware

Previous projects made working with hardware much easier since the only new hardware was the LCD. Also every peripheral was tested separately so that made it easier to diagnose hardware problems since only parts of the project were functional during testing. The most difficult part of this project was learning to use the LCD. This required finding a non I2C LCD that had a detailed datasheet that would work with this microcontroller. The LCD also requires 16 pins so that meant other peripherals needed to be compatible with the other pins left over. Most of the debugging was done for the LCD since the wiring was easy to mix up and the code in the datasheet needed to be modified before it would work with the PIC18F4331.

Conclusion

This project provided experience in programming and using the PIC microcontroller and the peripherals described above. This project was able to correctly use the PIC's timer, ADC, and an LCD to create a working car simulation. Creating this car simulation gave me experience using the concepts learned about embedded systems and I am confident that I could do something more complex in the future.

References

- PIC18F4331 Datasheet
    - http://ww1.microchip.com/downloads/en/devicedoc/39616b.pdf
- 1602A LCD Datasheet
    - https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

Appendices

Complete Code

In order to make code more readable all files were combined into one and functions were kept separate and defined before main. Configuration bits were also included.

```
#ifndef CONFIG_H
#define CONFIG_H
#include <xc.h>

#define _XTAL_FREQ 8000000

// PIC18F4331 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1H
#pragma config OSC = IRCIO      // Oscillator Selection bits (Internal oscillator block, port function on
RA6 and port function on RA7)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor
enabled)
#pragma config IESO = ON        // Internal External Oscillator Switchover bit (Internal External
Switchover mode enabled)

// CONFIG2L
#pragma config PWRTEN = OFF     // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = OFF      // Brown-out Reset Enable bits (Brown-out Reset disabled)
// BORV = No Setting

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bit (WDT disabled (control is placed on
the SWDTEN bit))
#pragma config WDPS = 32768     // Watchdog Timer Postscale Select bits (1:32768)
#pragma config WINEN = OFF      // Watchdog Timer Window Enable bit (WDT window disabled)

// CONFIG3L
#pragma config PWMPIN = OFF     // PWM output pins Reset state control (PWM outputs disabled upon
Reset (default))
#pragma config LPOL = HIGH      // Low-Side Transistors Polarity (PWM0, 2, 4 and 6 are active-high)
```

```
#pragma config HPOL = HIGH      // High-Side Transistors Polarity (PWM1, 3, 5 and 7 are active-high)
#pragma config T1OSCMX = ON     // Timer1 Oscillator MUX (Low-power Timer1 operation when
microcontroller is in Sleep mode)

// CONFIG3H
#pragma config FLTAMX = RC1     // FLTA MUX bit (FLTA input is multiplexed with RC1)
#pragma config SSPMX = RC7      // SSP I/O MUX bit (SCK/SCL clocks and SDA/SDI data are
multiplexed with RC5 and RC4, respectively. SDO output is multiplexed with RC7.)
#pragma config PWM4MX = RB5     // PWM4 MUX bit (PWM4 output is multiplexed with RB5)
#pragma config EXCLKMX = RC3    // TMR0/T5CKI External clock MUX bit (TMR0/T5CKI external
clock input is multiplexed with RC3)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (Enabled)

// CONFIG4L
#pragma config STVREN = OFF     // Stack Full/Underflow Reset Enable bit (Stack full/underflow will
not cause Reset)
#pragma config LVP = OFF        // Low-Voltage ICSP Enable bit (Low-voltage ICSP disabled)

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection bit (Block 0 (000200-000FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection bit (Block 1 (001000-001FFF) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot Block (000000-0001FFh) not
code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not
code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection bit (Block 0 (000200-000FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection bit (Block 1 (001000-001FFF) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers
(300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0001FFh) not
write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not
write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection bit (Block 0 (000200-000FFFh) not protected
from table reads executed in other blocks)
```

```
#pragma config EBTR1 = OFF      // Table Read Protection bit (Block 1 (001000-001FFF) not protected
from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block
(000000-0001FFh) not protected from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.


#endif   / XC_HEADER_TEMPLATE_H /

#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LCD_DATA    LATB
#define LCD_READ    PORTB
#define LCD_IO_BF   TRISB
#define LCD_BF      PORTDbits.RD4
#define LCD_RS      LATDbits.LD4
#define LCD_RW      LATDbits.LD3
#define LCD_EN      LATDbits.LD2

uint8_t print_buffer[33] = {0}; // buffer to print stuff to serial

volatile uint8_t uart_char = 0;
volatile bool uart_rcv_data = false;

uint16_t count = 1000; //counter for timer
uint16_t light_count= 100;
uint16_t flag = 0; //flag that tells the program timer is done
uint16_t pot_value; //will hold the original value
uint8_t voltage = 0; //converted value ranging from 0-5
uint8_t reset_flag = 0; //reset flag
uint8_t speed_value = 0; //speed value
uint8_t speed_count = 0; //speed count for speeding
uint8_t light_value = 0; //int that keeps track of light color
```

```
void LCD_clock(void){ //LCD commands were made using data sheet with very some modification for
this pic
   LCD_EN = 1;
   __delay_ms(1);

   LCD_EN = 0;
   __delay_ms(1);
}

void LCD_command(uint8_t command){
   LCD_RS = 0;
   LCD_RW = 0;

   LCD_DATA = command;
   LCD_clock();
   __delay_ms(10);
}

void LCD_command_write(char data)  //Send 8-bits through 4-bit mode
{
   LCD_DATA = data;
   LCD_RW = 0;
   LCD_RS = 0;
   LCD_RS = 1;             // => RS = 1
   __delay_ms(50);
   LCD_EN = 0;
   LCD_EN = 1;
   for(int i=2130483; i<=0; i--)  NOP();
   LCD_EN = 0;
}

void LCD_init_8bits(void){
   LCD_RS = 0;
   LCD_RW = 0;

   __delay_ms(50);
   LCD_command(0x38);
   LCD_command(0x38);
   LCD_command(0x38);

   LCD_command(0x0C); //turn on cursor blink on
   LCD_command(0x06); //set auto increment right mode no shifting
   LCD_command(0x01); // go to first position
}
```

```
uint8_t LCD_set_pos(uint8_t x,uint8_t y){
   if (y == 1){
      x+=0x40;
   }
   __delay_ms(100);
   LCD_command(0x80 | x);

   return x;
}

static uint8_t pos_track = 0;
static uint8_t line_track = 0;

void LCD_print(uint8_t x){
   LCD_RS = 0;
   LCD_RS = 1;           // => RS = 1
   __delay_ms(50);
   LCD_DATA = x;
   LCD_EN = 0;
   LCD_EN = 1;
   for(int i=2130483; i<=0; i--)  NOP();
   LCD_EN = 0;
   pos_track++;
}


void LCD_print_string(const char *str){
   int i = 0;
   do{
      LCD_print(str[i]);
      i++;
   }while(str[i] != '_');
}

void reset_sim(void){ //infitinte loop when simulation ends
    LATDbits.LD0 =0; //clear police lights if reset flag is off
    LATDbits.LD1=0;
   if(reset_flag==1){ //flash lights if reset is on
      while(1){
    __delay_ms(85);
    LATDbits.LD0 =~LATDbits.LD0;
    __delay_ms(85);
    LATDbits.LD1=~LATDbits.LD1;
```

```
     __delay_ms(100);

     while(PORTCbits.RC4){ //reset the simulation once button is pressed
        light_value=0;
        speed_value=0;
        LATA=0x00;
        LATDbits.LD0 =0;
        LATDbits.LD1=0;
        LCD_command(0x01); //clear lcd on reset
        reset_flag=0;
        speed_count=0;
        count=1000;
        light_count=125;
        return;}
     }
        }

    else{
     LATDbits.LD0 =0;
     LATDbits.LD1=0;
     return;
    }

}


void stoplight(void){ //in charge of stoplight and speed limit checking
   if(flag ==1){
      if(light_count == 100){
         //led changes to green
        light_value=3;
        LATA=0x04;

      }
       else if(light_count == 60){ //if counting
        //led changes to yellow
         light_value=2;
         LATA=0x02;

      }
      else if(light_count== 40){
         //led changes to red
         light_value=1;
         LATA=0x01;
```

```
 }
 else if(light_count ==0){
    //led changes to green
    light_value =3;
    LATA=0x04;
    light_count =100;

 }
 else{ //when done set flag and reset timer flag
 count--;
 //check speed
 }

if(light_value==1){
 if(speed_value==0){//red car crashes when speed >0
    speed_count=0;
    reset_flag=0;
    reset_sim();

 }
 else if (speed_value==1){ //crash senerio
     LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Car crashed_");
    reset_flag=1;
    reset_sim();
 }
 else if (speed_value==2){
     LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Car crashed_");
    reset_flag=1;
    reset_sim();
 }
 else if (speed_value==3){
     LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Car crashed_");
    reset_flag=1;
    reset_sim();
 }
 else if (speed_value==4){
     LCD_set_pos(0,0);
```

```
        __delay_ms(200);
        LCD_print_string("Car crashed_");
        reset_flag=1;
        reset_sim();
    }
    else if (speed_value==5){
         LCD_set_pos(0,0);
        __delay_ms(200);
        LCD_print_string("Car crashed_");
        reset_flag=1;
        reset_sim();
    }
}
   else if(light_value==2){ //yellow pulled over >3
    if(speed_value==0){
        speed_count=0;
        reset_flag=0;
        reset_sim();
    }
    else if (speed_value==1){
        speed_count=0;
        reset_flag=0;
        reset_sim();
    }
    else if (speed_value==2){
        speed_count=0;
        reset_flag=0;
        reset_sim();
    }
    else if (speed_value==3){
        speed_count=0;
        reset_flag=0;
        reset_sim();
    }
    else if (speed_value==4){ //speed count up 1
       speed_count++;

    }
    else if (speed_value==5){ //speed count up 2 for faster speed
       speed_count=speed_count +2;

    }
}
   else if(light_value==3){ //green pulled over >3
```

```
        if(speed_value==0){
            speed_count=0;
            reset_flag=0;
            reset_sim();
        }
        else if (speed_value==1){
            speed_count=0;
            reset_flag=0;
            reset_sim();
        }
        else if (speed_value==2){
            speed_count=0;
            reset_flag=0;
            reset_sim();
        }
        else if (speed_value==3){
            speed_count=0;
            reset_flag=0;
            reset_sim();
        }
        else if (speed_value==4){
            speed_count++;

        }
        else if (speed_value==5){
            speed_count= speed_count +2;

        }
    }
        if(speed_count==40){ //if caught speeding when max speed_count is reached
           LCD_set_pos(0,0);
            __delay_ms(200);
            LCD_print_string("Pulled over_");
            reset_flag=1;
            reset_sim();
        }

        count =1000; //resets count value that controls timing
        flag=0; //resets flag
        }
}


void main(void){
```

```
OSCCON=0x73; //osciallator setup to 4MHz
TRISD = 0x00; //set D as output
TRISB = 0x00;
TRISCbits.RC4=1;        //Set RC4 as input
TRISCbits.RC3=1;        //RC3 as input
TRISAbits.RA0=0;        //Set RA0 as output
TRISAbits.RA1=0;        //Set RA0 as output
TRISAbits.RA2=0;        //Set RA0 as output
TRISDbits.RD4 = 0;      //RD4 as output
TRISDbits.RD3 = 0;      //RD3 as output
TRISDbits.RD2= 0;       //RD2 as output
LATA = 0x00;       //all leds start off
LCD_RS = 0;
LCD_RW = 0;
LCD_EN = 0;
LATD = 0x00;
RCONbits.IPEN = 1;
INTCONbits.GIEH = 1;
INTCONbits.GIEL = 1;// base interrupt setup
LCD_init_8bits();
TRISAbits.RA4 = 1; //AN4 as input
TMR0=0xBFA; //clear timer0
T0CON=0x82; //sets bits for T0CON register 10000010
ADCON0 = 0x01; //intialize ADC
ADCON1 = 0x10;
ADCON2 = 0x00;
ADCON3 = 0xA0;
ADCHS = 0x01;
ANSEL0 = 0x00;
ADRESH=0;          //reset values
ADRESL=0;
RCONbits.IPEN = 1;
INTCONbits.GIEH = 1;
INTCONbits.GIEL = 1;
INTCONbits.TMR0IE = 1; //enable timer0
INTCON2bits.TMR0IP = 1;
//LCD_set_pos(0,0);
  //      __delay_ms(200);
    //    LCD_print_string("Push to start_"); was printing infinitly
while(PORTCbits.RC3){ //wait for start button
while(1){
ADCON0bits.ACMOD0 = 0;
ADCON0bits.ACMOD1 = 0;
```

```
ADCON0bits.GO = 1;
while(ADCON0bits.GO == 1); //gets pot value and converts
pot_value = 0x00; //reset
pot_value = ADRESH; //gets pot value
voltage = pot_value / 51;  //255 max pot value / 5 potential options
  if (voltage < 1){ //if less then 1 volt
    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 0_");
    speed_value=0;
  }
  else if (voltage < 2){ //if less then 2 volts but more then 1

    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 1_");
    speed_value=1;
  }
  else if (voltage < 3){ //between 2 and 3

    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 2_");
    speed_value=2;
  }
  else if (voltage < 4){ //between 3 and 4

    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 3_");
    speed_value=3;
  }
  else if (voltage == 4){ //exactly 4

    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 4_");
    speed_value=4;
  }
  else { // max value
    LCD_set_pos(0,0);
    __delay_ms(200);
    LCD_print_string("Speed: 5_");
    speed_value=5;
```

```
        }
    }
}
}


void __interrupt() high_isr(void){
    if(TMR0IF){
        if(count > 0){ //if counting
            count--;
        }
        else{ //when done set flag and reset timer flag
        flag=1;    //this should happen every second to verify timing
        stoplight(); //calls stoplight and speed checker
        light_count--;
        TMR0=0x0BFA;
        TMR0IF=0;
        }
    }
}

void __interrupt(low_priority) low_isr(void){
    INTCONbits.GIEH = 0;

    if(0){

    }

    INTCONbits.GIEH = 1;
}
```

Manual

Start simulation

To start the simulation the start button is pressed with the potentiometer at the lowest value. Once this is done the LCD will display speed and the spotlight will cycle through colors starting with green.

Controlling speed

Once the simulation is started the potentiometer is used to control the speed with the LCD being used to view the change in speed.

Resetting the Simulation

The simulation ends in two cases. If the vehicle is moving at a red light, or if the vehicle is caught going above the speed limit (3) for an extended period of time. If the vehicle is moving at 4 then the speed count increases by one and if they go 5 speed units the speed count increments twice so the exact time is different depending on what the speed is. If the driver runs a red light the car crashes and the LCD will display a crash message and the police lights flash waiting to be reset. If the driver is caught speeding then the LCD will display a pulled over message and the police sirens flash waiting to be reset. In either of these scenarios the reset button is used to reset the simulation.When resetting all the leds will turn off and there will be a slight delay before the stoplight restarts and the poteinmeter reads speed and outputs to the LCD.

Video Demonstration

A google drive link is provided to show a full demonstration. This demo shows the state before the start button, then the start button. Then time is given to show the stoplight make a full loop. Then the LCD and potentiometer are shown. The lighting made it tough to see the screen during certain angles but Figure 4 shows examples of what is shown. Then the 2 end scenarios are shown and a reset after each.

https://drive.google.com/file/d/1wTq4ZqBLKs9Ft1yv-fy6W1kL9PKQnN1D/view?usp=sharing