



# AUTOMATED SLOT CARS

---



A Design Project

in

## MECHATRONICS

ME6405 Fall 2002

TEAM 4

Ryan Krauss / Lisa Ellis / Joe Frankel



## Table of Contents

1. Introduction	3
2. Concept	4
3. Timeline	5
4. Mechanical Design	6
5. Electrical Design	8
6. Software Design	16
7. Features	22
8. Team Members	23
9. Sponsors	24
10. Appendix: C code	25

## 1. Introduction

### *Can you beat the Motorola MC68HC11?*

*This project involved building an automated control system for a toy slot car racetrack.*

*The result is a system that not only utilizes many key features of the Motorola MC68HC11-E9 microprocessor, but is also great fun to play with. Who wouldn't want to try to beat a computer controlled race car?*

### **Traditional Slot Car Operation**

Traditional slot cars run on a track with two lanes, and users compete with each other by racing around the track with small electric cars. The players control the speed of their cars by squeezing a trigger on a handheld gun. The trigger operates a potentiometer which varies the voltage to two rails along the length of each lane of the track. Sliding pickups on the bottom of the car close the circuit through a dc motor, which drives the car forward.

In this situation, the user provides feedback to the potentiometer by sensing the car's speed visually and adjusting the position of the potentiometer to try to make his car go faster than his opponent without spinning out or flying off the track.

### **Automated Slot Car Operation**

In this project, one of the two human users was replaced with a Motorola MC68HC11-E9 microprocessor unit (MCU), which controls the car using pulse width modulation (PWM) of the voltage from a constant dc power source.

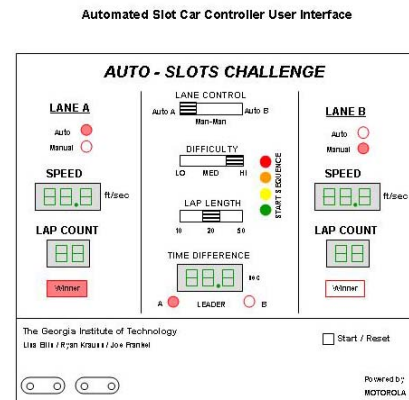
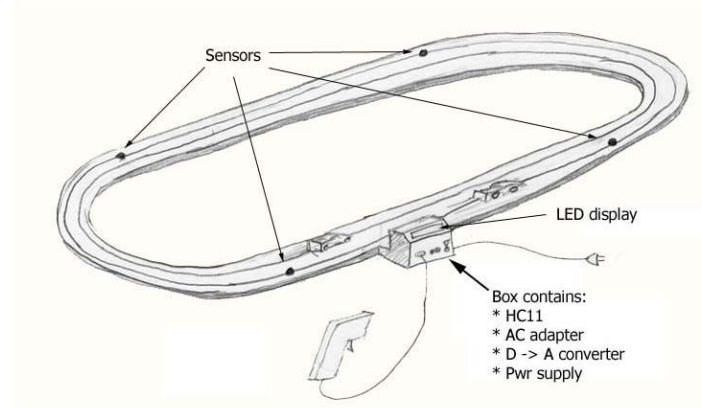
Closed-loop feedback was achieved with infrared photodetectors mounted in the track, which send pulses to the processor at various points along its length as the car passes them. Based on the timing of these pulses, the car slows down in the corners and speeds up in the straight-aways.

A liquid crystal display (LCD) provides information to the user including real-time speed and lap times, as well as startup instructions and winner/loser results. Relative speed is also displayed with arrays of light emitting diodes {LEDs} for each car.

Difficulty level may be selected from low, medium, and high, which allows users to practice and increase the difficulty as they improve their skills. As the difficulty increases, the duty cycle is increased and the computer controlled car goes faster, making it harder to beat.

## 2. Concept

### Conceptual Design



The pictures above illustrate the initial concept of the Automated Slot Car system.

### Design Goals

#### *The microprocessor will be used to:*

- Determine whether or not the car is in a corner and adjust the car speed accordingly
- Signal the user that race is going to start
- Provide feedback of lap time differences between user and computer controlled cars
- Allow both cars to be manually controlled

#### *Proposed microprocessor subsystems to be utilized:*

- Analog to Digital Converter
- Main Timer and/or Pulse Accumulator
- Parallel Input/Output
- Maskable/Nonmaskable Interrupts

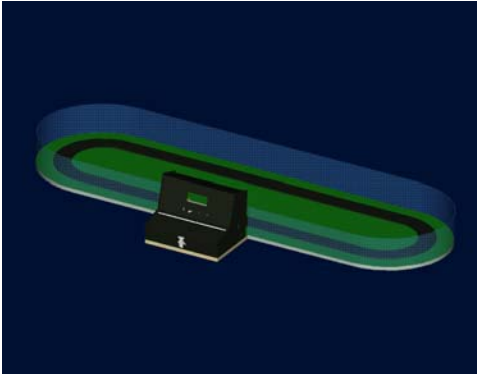
### 3. Timeline

<b>Week #</b>	<b>Goal</b>
1	Begin initial research: D/A, corner sensors, LED displays, I/O methods and pins available. Purchase track and cars.
2	Construct initial voltage measuring circuitry and signal conditioning.
3	Use the HC11 to output a constant voltage to the car and measure lap times. Determine speed versus voltage profile.
4	Be able to measure the voltage being given to the car and sense when the car is entering or leaving a corner.
5	Begin system integration.
6	Calibrate system, tweak control gains, and refine control algorithm.
7	Finalize physical design; Work on web page, report, and presentation.
8	Present

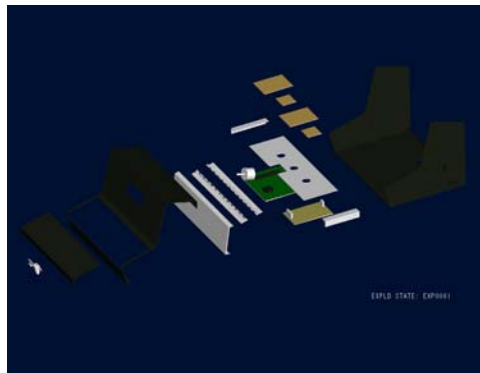
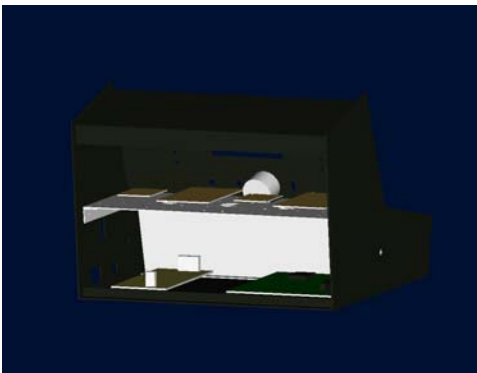
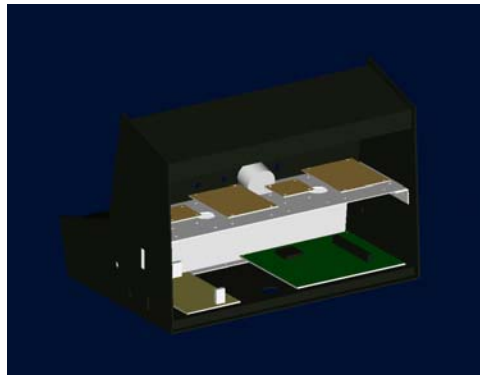
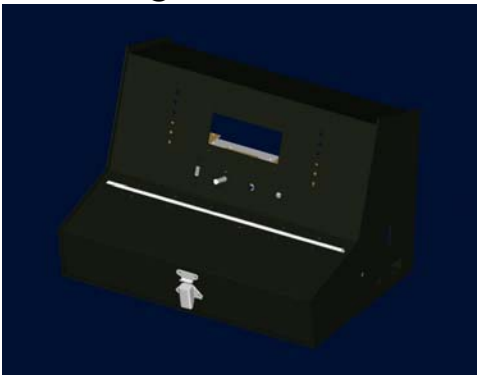
## 4. Mechanical Design

All the fabrication work was done in-house. The aluminum box containing all the electronics was milled and the hinged cover was fabricated in the machine shop. The table and lexan shield were formed in the fabrication shop. Assembly took place in the mechatronics lab.

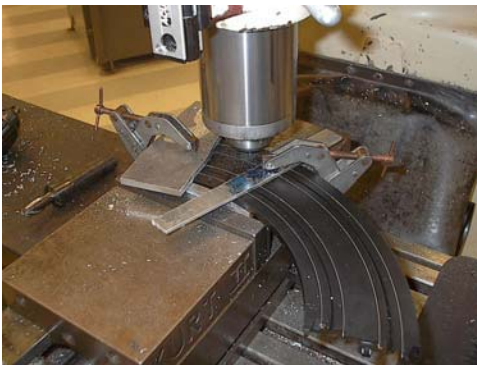
### Table Design



### Box Design



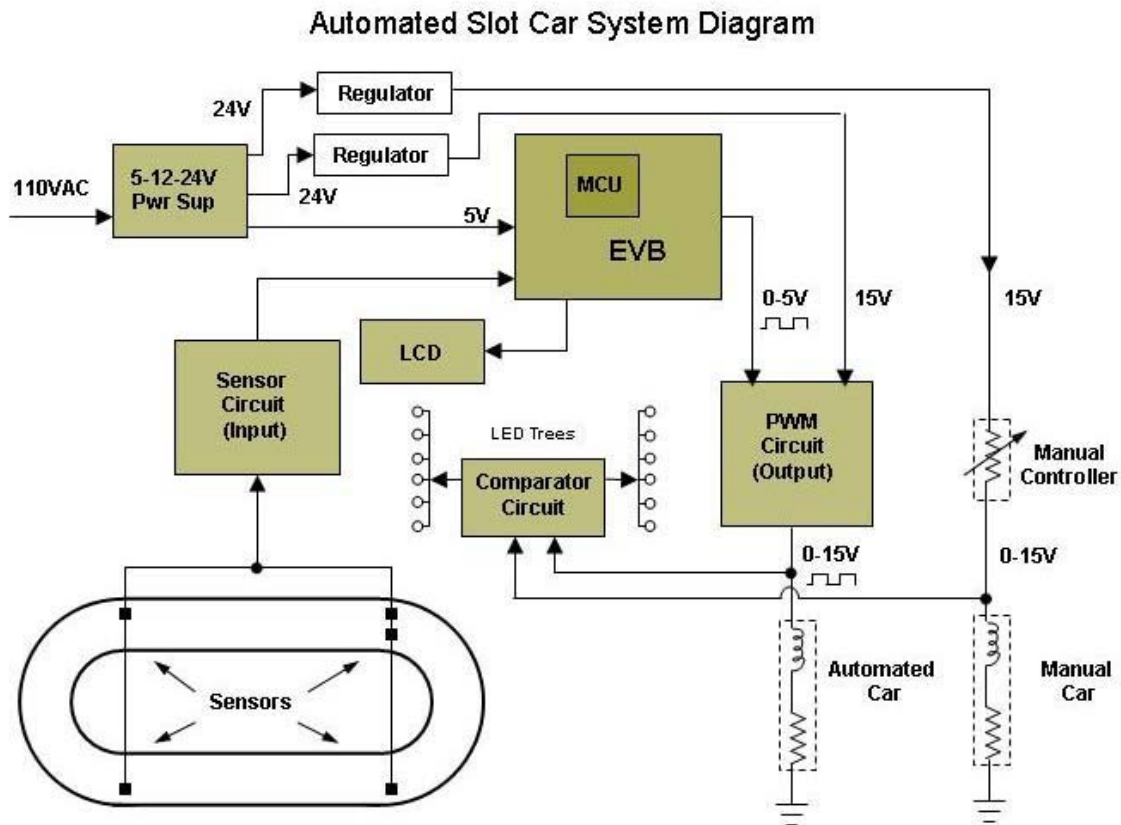
### Construction Photos



## 5. Electrical Design

### Overall System

A great deal of effort went into the design of the electrical circuitry required to run the automated slot car controller. The diagram below illustrates the overall system.

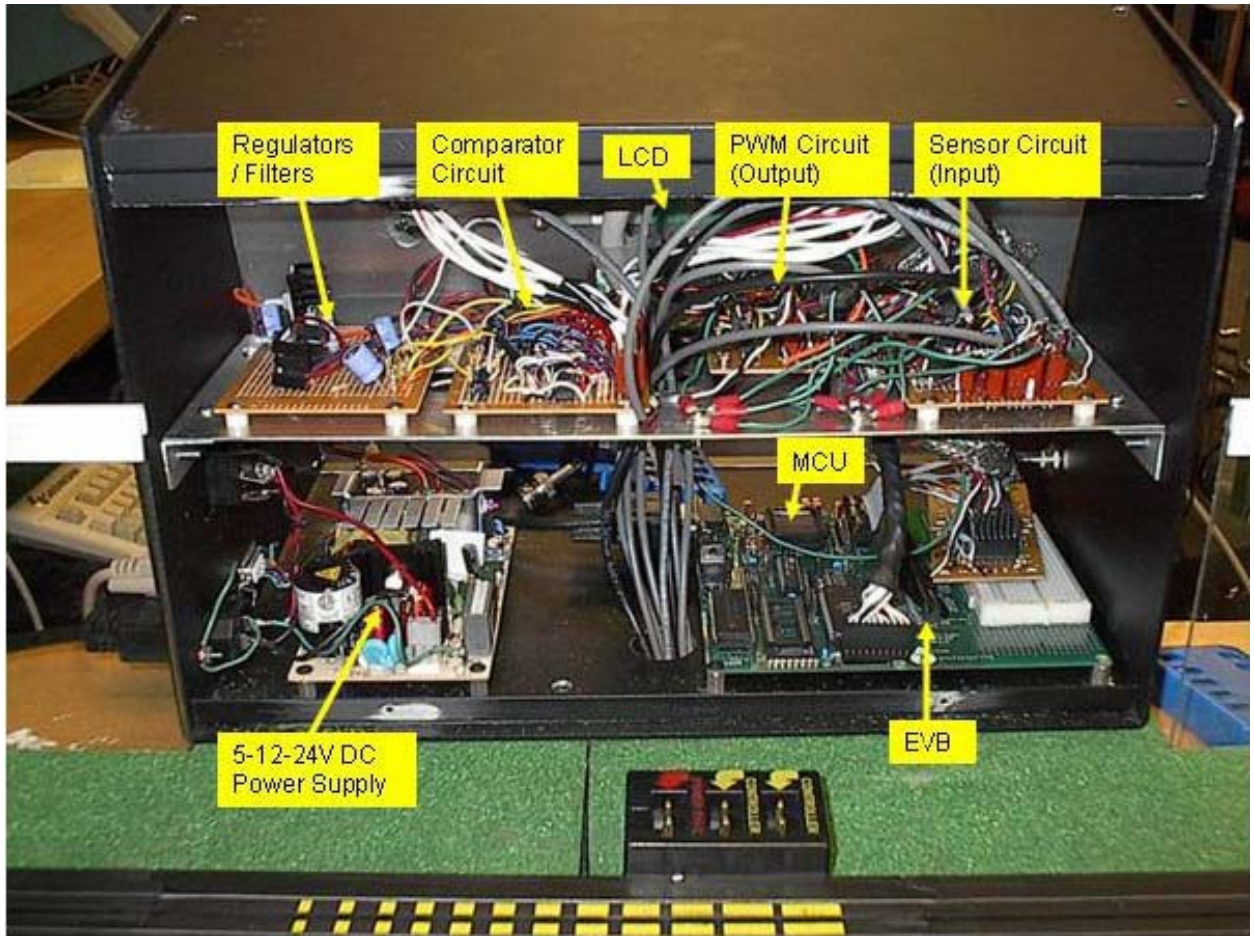


### Components

Motorola MC68HC11-E9 Microprocessor Unit (MCU)  
Axiom Evaluation Board (EVB)  
5-12-24V DC Power Supply  
LCD Display

Sensor Circuit (Input)  
PWM Circuit (Output)  
Comparator Circuit

The four components listed on the left were purchased off the shelf and integrated accordingly. The three on the right were designed and hand built in-house. For a complete description of the user interface that operates the system, see the Features section.



Motorola MC68HC11-E9 Microprocessor Unit (MCU)

The Motorola MC68HC11 microprocessor serves as the brain of the system. The E9 variant is effectively a 2MHz processor with 52 pins for I/O. It includes an internal clock, timer, A-D converter, 512b of internal RAM, 512b of internal EEPROM, and 5 I/O ports. See illustration at right for a block diagram of the MCU components.

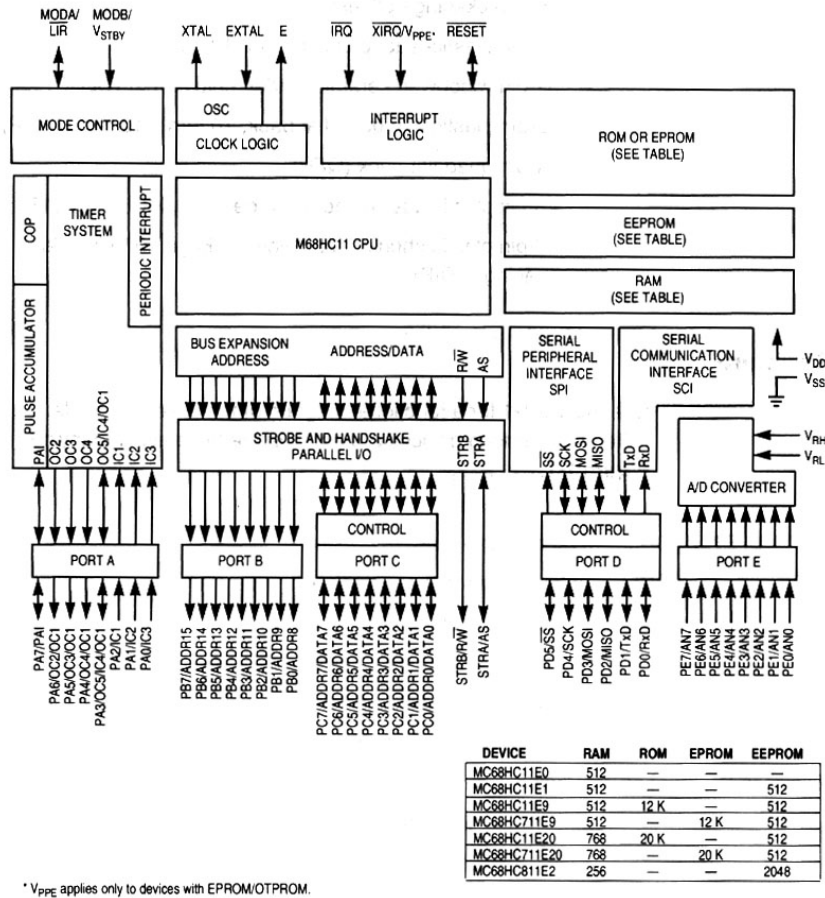
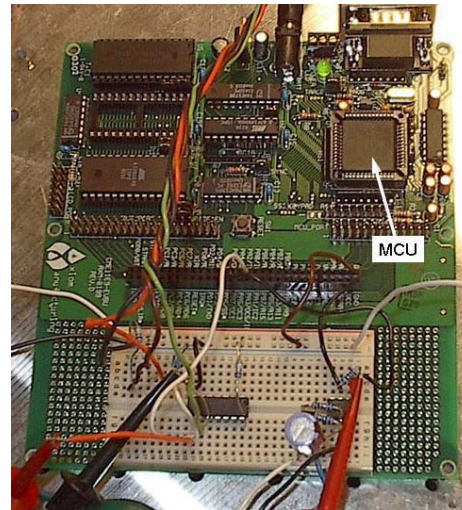
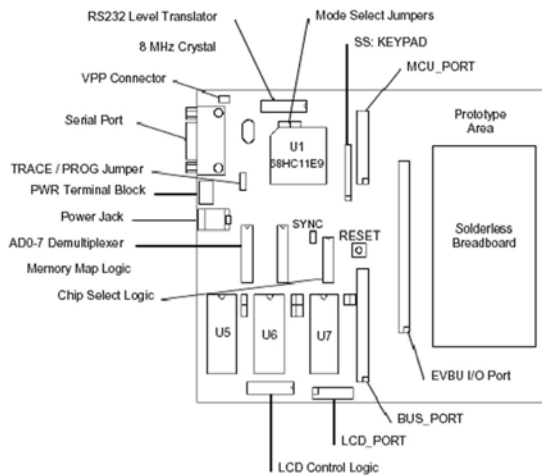


Figure 1-1. M68HC11 E-Series Block Diagram

This project utilized the following capabilities of the HC11:

- **Input Capture:** sensor pulses from cars
- **Main Timer:** overflow count and output compare for duty cycle control
- **Parallel I/O:** sensor position pulses, driving voltage measurements, toggle/dial/momentary switches (input), PWM control, LCD text display (output)
- **Analog-to-Digital Converter:** driving voltage and difficulty level
- **Maskable Interrupts:** timer overflows, reset race button

## Axiom Evaluation Board (EVB)



The Axiom EVB contains the Motorola MC68HC11-E9 64-pin microprocessor, and connects it with several peripheral devices for expanding memory and interface capabilities. In the Automated Slot Car system, pin assignments were as follows:

**Inputs:**

PE0: Auto Car Voltage  
 PE1: User Car Voltage  
 PE2: Difficulty Level Switch  
 PD4: Computer Control  
 PD5: Start Button  
 IRQ: Reset button  
 PA0: Starting Line sensor for auto car  
 PA1: Both into corner sensor for auto car  
 PA2: Out of corner sensor for auto car  
 PA3: Starting line sensor for user car

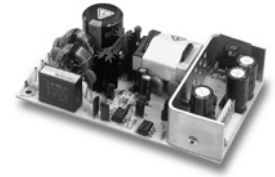
**Outputs:**

PA6: PWM to Auto Car  
 LCD Port

For more information on the EVB, see the Buffalo Manual.

## 5-12-24V DC Power Supply

A power supply was used that converted line power to 5, 12, and 24 VDC. The 24V output was regulated to 15V, which powered the motors in the cars. 15V was found to be the maximum that the cars could use before flying off the track.



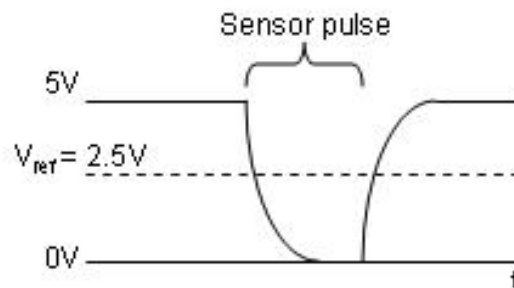
## LCD Display

A liquid crystal display was used to display text during the startup sequence, speed and lap time during the race, and winner / loser results at the end of the race.



## Sensor Circuit (Input)

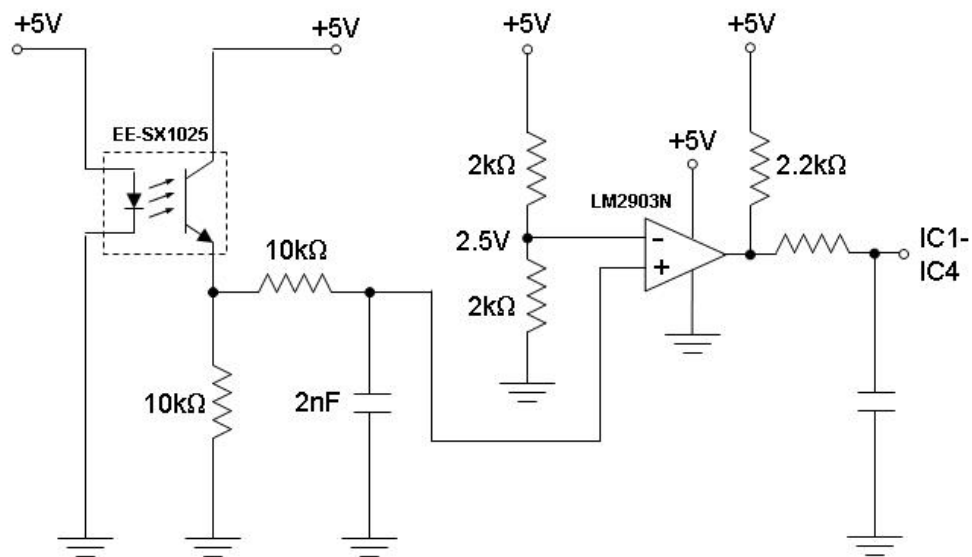
The slot cars have a small pin underneath which rides in the slot on the track and keeps the car in its lane. Slot-type photomicrosensors were mounted in the track, which produce a continuous 5V at the output until the infrared beam that spans the slot is broken. The sensors were mounted so that when the car passed a sensor, the pin would pass through the slot in the sensor and break the beam. During the period that the beam is blocked, the sensor output drops to zero.



The sensor circuit was designed to capture these zero pulses and send a signal to the MCU that it could detect (falling edge). These edges were used to trigger interrupt routines in the software that increased or decreased the duty cycle--and therefore speed of the car--accordingly.

The circuit operates such that sensor pulses pass through an RC filter and on to a comparator before closing the loop to the input capture (IC) pins on the MCU.

Photomicrosensor Input Capture Circuit



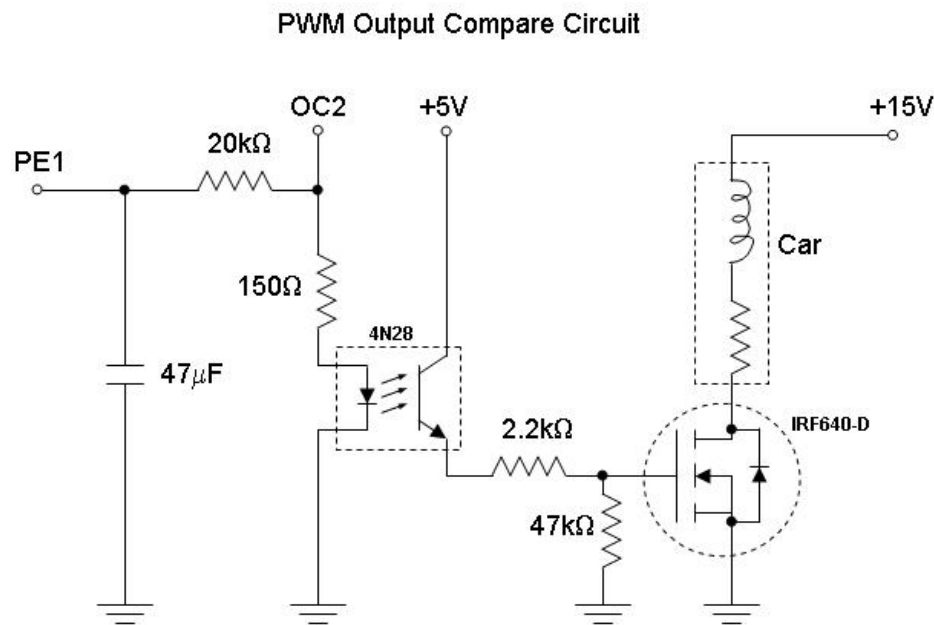
Perhaps the greatest challenge in this project came in protecting the signal from noise on its path between the sensors and MCU.

At racing speed, the cars draw around 300mA at 15V, through sliding pickups on the rails and clicking across joints. This generates noise spikes very close to the sensors, which can cause false triggering to the MCU and subsequent loss of control.

To remedy this problem, shielded sensor wires were used, low-pass RC filters were built into the circuit, and comparators were used to try to clean up the signal. Several iterations of changing RC values in the filters were necessary before the best cutoff frequency was found, which balanced between noise reduction and sufficient notch depth to trigger an interrupt in the MCU.

## PWM Circuit (Output)

The power delivered to the automated car is controlled via a two-stage PWM transistor circuit. The variable duty cycle square wave from the MCU is sent to the first of the circuit's two stages, which consist of an Optocoupler and a MOSFET.



Voltage pulses across the LED in the Optocoupler cause current to flow through it, which in turn causes current pulses to flow through the transistor side from the power supply, multiplied but yet decoupled from the MCU by means of an infrared beam.

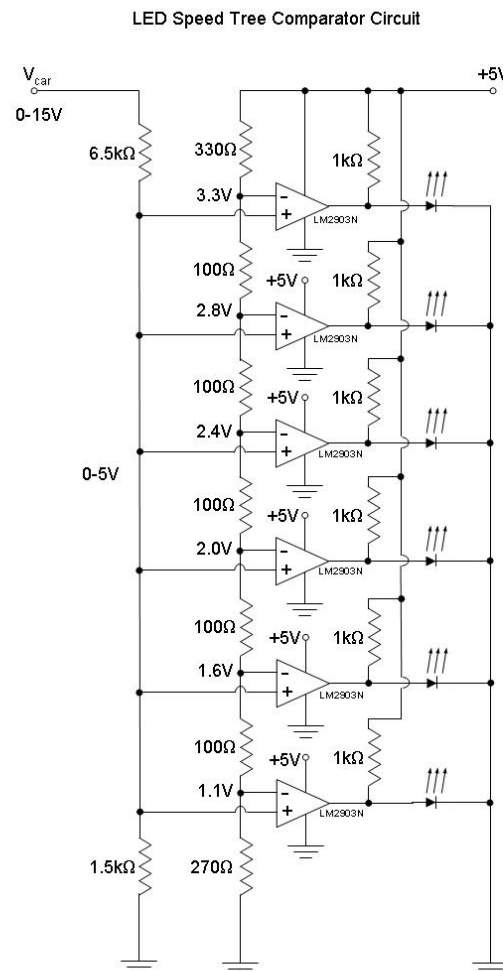
The emitter of the Optocoupler in turn feeds current through a voltage divider at the gate of a MOSFET transistor. The voltage at the gate controls the source/drain voltage at the output, which controls the voltage seen at the rails of the car.

This circuit provides complete isolation and protection between the MCU and the car, while allowing sufficient power to flow from the power supply to the car motor. The speed of the car is adjusted by adjusting the duty cycle of the square wave coming from the MCU.

Except for a small time delay due to the dynamics of the car, which is accounted for in the software, the MCU can control the car with such precision that is very difficult for a human opponent to defeat.

## Comparator Circuit

In addition to the LCD screen which displays real-time velocity measurements of both cars to the user, two trees of LED's on the front panel also provide a qualitative measurement of the speed of each car. These trees, one for each car, are driven by an array of comparators arranged in a ladder-type configuration.



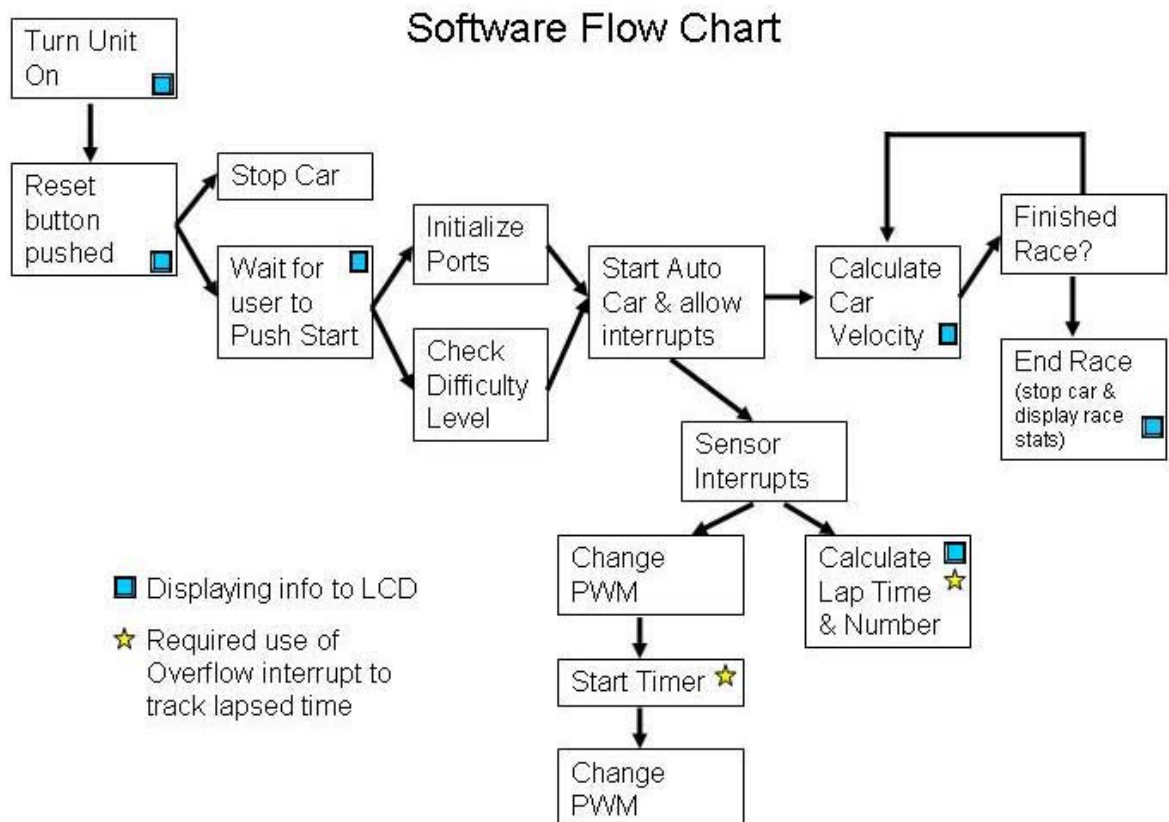
A constant reference signal is applied to the inverting input of each of six comparators, increasing between each level of the voltage divider ladder. The car's voltage is divided appropriately and sent to the non-inverting input of each comparator. When the voltage difference at the inputs to each comparator is positive, the output to that comparator rails to its supply value and excites its LED on the tree.

The result is a string of LEDs that illuminate sequentially with the speed of each car, giving the user a sense of how fast each car is moving.

## 6. Software Design

### Software Flow Chart

Below is a diagram illustrating the logical flow of the software written to operate the automated slot car system.



At startup, a string of characters are sent to the LCD to welcome the user to the race, and then the processor goes into an infinite loop while waiting for the user to press the start button. When the start button is pressed, PD5 goes high and a countdown begins while the LCD is updated. If the user crosses the sensor at the starting line during this period, a false start will be detected and the race will stop.

When the countdown reaches zero, the automated car takes off at 100% duty cycle (15V) and the race begins. During the race, the duty cycle is updated each time the auto car crosses a sensor going into and out of a turn. See the Pulse Width Modulation Strategy section below. In addition, when each of the cars cross the start/finish line sensors, lap counters are incremented, lap times calculated, and the results are sent to the LCD display.

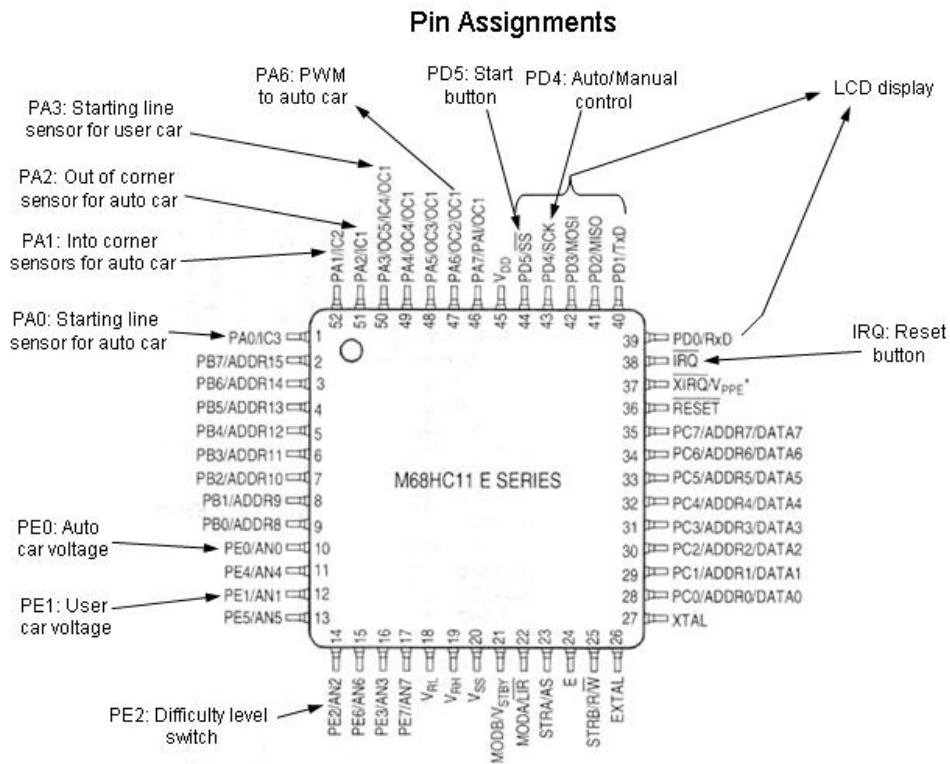
When a preset number of laps are met by either car, the race is finished, the auto car stops, and the winner is determined by which car's lap counter reached the lap count first. A message is sent to the user on the LCD display declaring who won the race.

The user can press the reset button at this point if he or she wants to start another race.

The C code that operates the Automated Slot Car system is listed in the Appendix.

### Pin Assignments

The following pin assignments determine how the MCU is connected to the rest of the system.



Note that the pin connections to the chip are routed through the MCU and LCD Ports on the EVB.

#### PE0: Auto Car Voltage

0-15V analog input, used for calculating current speed

PE1: User Car Voltage

0-15V analog input, used for calculating current speed

**PE2: Difficulty Level Switch**

0V = Lo difficulty

2.5V = Med difficulty

5V = Hi difficulty

**PD4: Auto/Manual Control**

Hi = Auto, Lo = manual control to outside lane

**PD5: Start Button**

Breaks loop and continues with start sequence

**IRQ: Reset button**

Triggers interrupt routine to cut voltage to cars and return to waiting for start button to be pressed

**PA0: Starting Line sensor for auto car**

Triggers service routine to count laps, and to increase duty cycle when falling edge is detected

**PA1: Both into corner sensors for auto car**

Two sensors tied to this pin through an AND gate which checks a toggle to determine which sensor

**PA2: Out of corner sensor for auto car**

Triggers service routine to increase duty cycle when falling edge is detected

**PA3: Starting line sensor for user car**

Triggers service routine to count laps, update user's lap time, and to detect a false start when a falling edge is detected

**PA6: PWM to Auto Car**

Variable duty cycle square wave output, sent to PWM circuit for amplification and auto car speed control

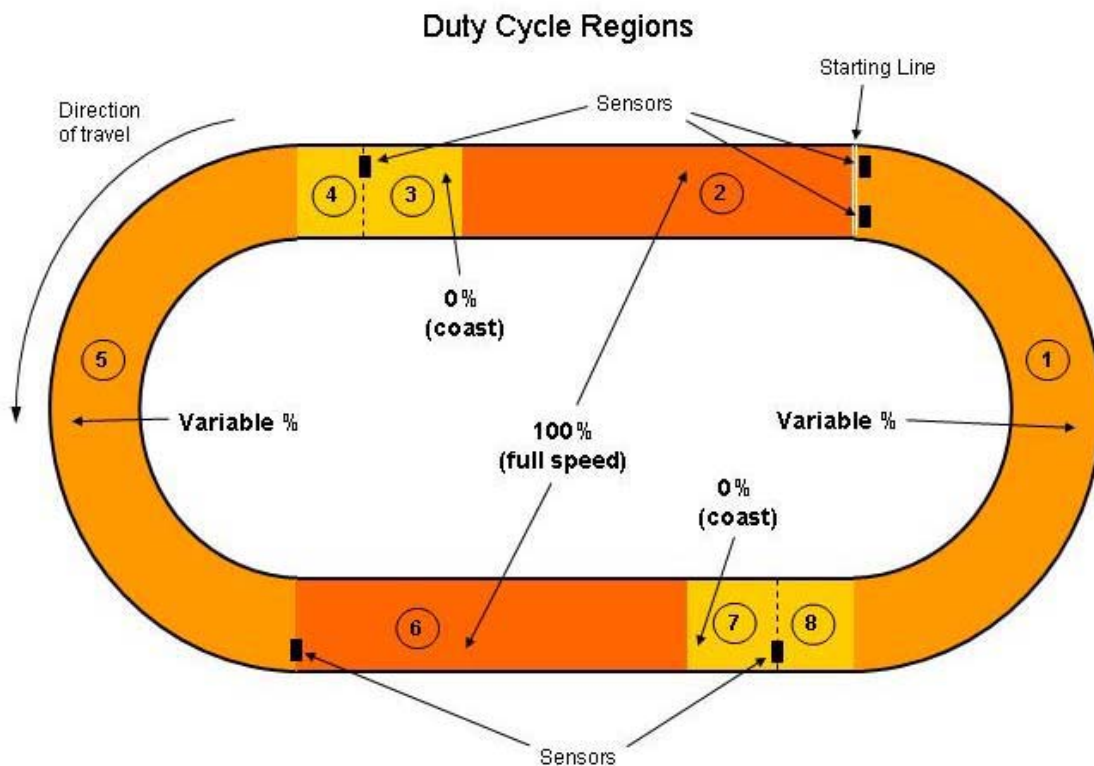
**LCD Port: LCD display**

Port D pins are temporarily used for sending print commands to LCD display during welcome sequence, start sequence, while racing, and winner/loser results.

## Pulse Width Modulation Strategy

A total of five photomicrosensors were mounted in the track, four in the automated car's (outside) lane, and one in the manual car's (inside) lane. Two of the four sensors on the outside lane were placed approximately six inches before each turn and the other two at the exit of each turn, one at the starting line. The sensor in the manual lane was placed at the start line also. Sensors pulses were used for adjusting duty cycle, counting laps, computing speed lap times, and determining the winner of the race.

The outside lane was then divided into eight regions where the duty cycle was held constant over each region, as illustrated below.



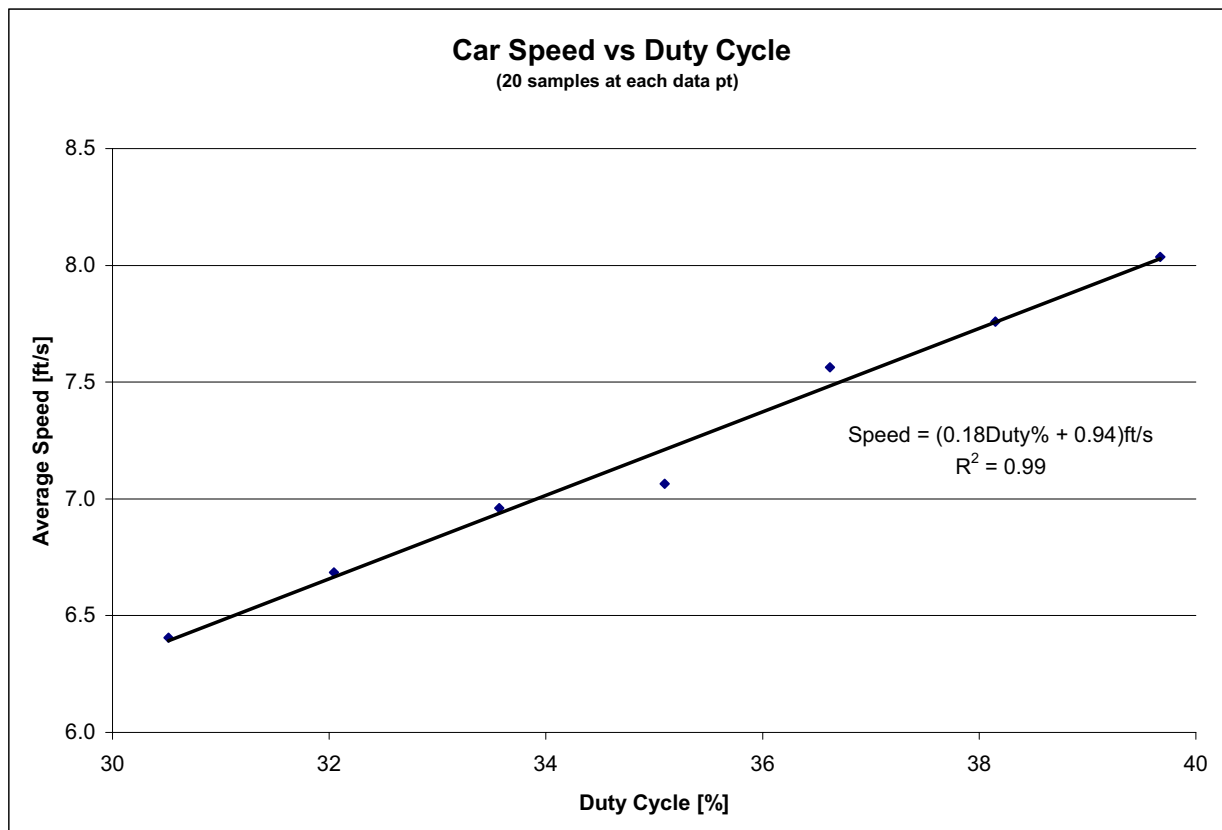
A counter is defined in the software that counts timer overflows. Each time the automated car passes through a sensor, this counter is set to zero and the current timer value is stored. Since the timer resets every \$FFFF (65,535) clock cycles and the clock speed is 2MHz, a timer overflow and subsequent increment of the counter occurs approximately every 30ms.

In the beginnings of the straight-aways, (regions 2 & 6 in the diagram above), the automated car is driven at 100% duty cycle, the full 15V available. When 210ms have elapsed (about 7 overflows), the car is going full speed, nearing the turn, and the duty cycle is reduced to zero (regions 3 & 7). The car then coasts through the sensor going into the turn, during which time an input capture is detected from the sensors. At a predefined time after the sensor pulse going into the turn has been detected, the duty cycle is then changed to a variable percentage that depends on calibration as the car goes around the turn (regions 1 & 5). An interrupt service routine is executed at each sensor pulse: input capture for the start of regions 2, 4, 6, and 8, and output compare at the start of 1, 3, 5, and 7.

The outside lane was chosen to be the automated car's lane in an effort to ensure that the human challenger would have the inside lane advantage. Testing has shown that the computer controlled car overcomes this disadvantage and is nearly impossible to beat when set to the Hi difficulty level.

### Speed vs. Duty Cycle

Data was collected to correlate the steady-state speed of the car with duty cycle. Using this data, the car could be driven at any desired speed between 6 and 9 ft/s by changing the duty cycle accordingly. The data is summarized in the chart below. As shown, duty cycle ranged between 30 and 40%.



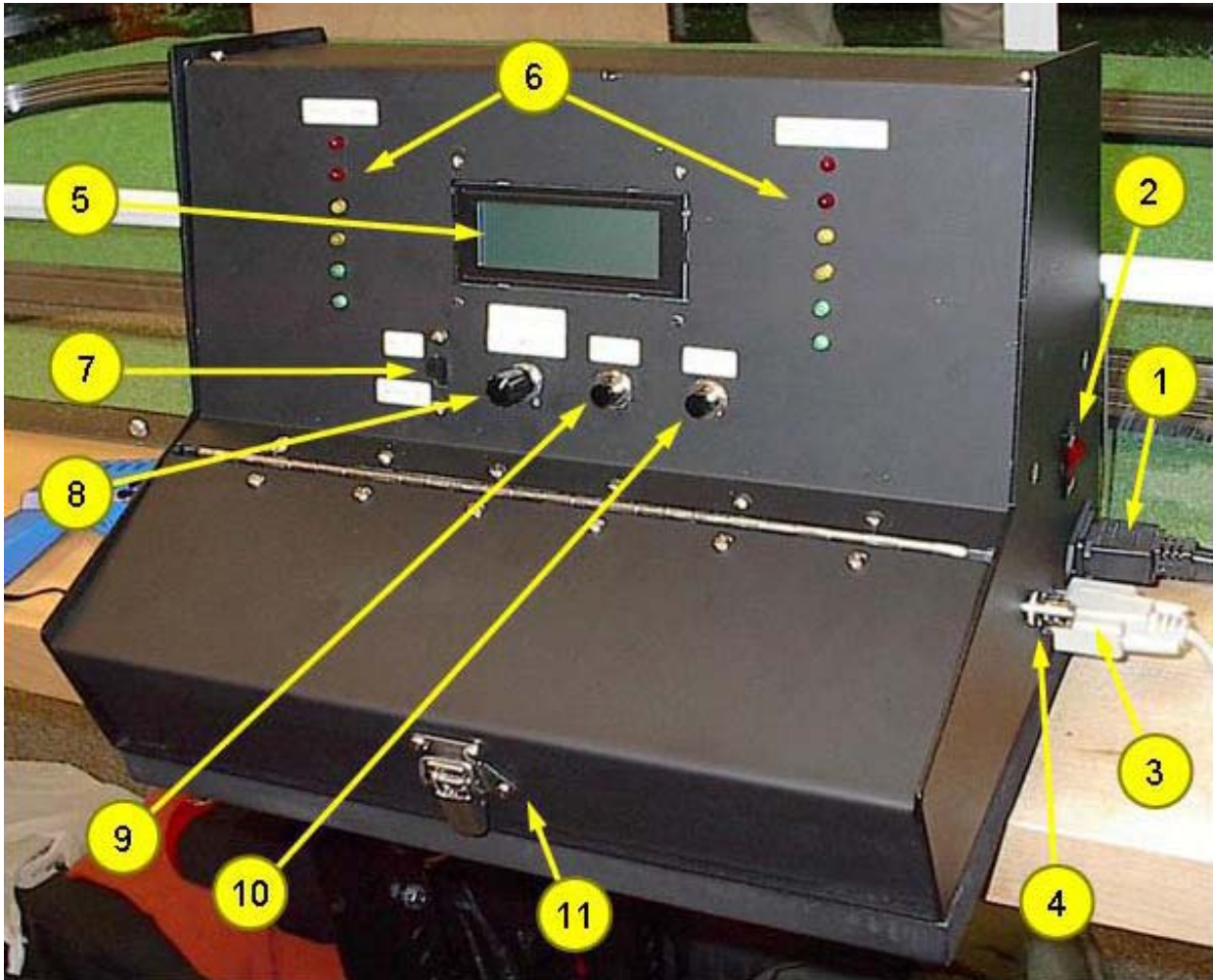
**Experimental Data of slot Car travel time for given duty cycle**

On Time (Clock Cycles)	20000	21000	22000	23000	24000	25000	26000
PWM Duty Cycle = $100 \cdot \frac{\text{On time}}{65535}$	30.5	32.0	33.6	35.1	36.6	38.1	39.7
<b>Average Time [ms]</b> Time it takes for car to traverse straightaway between on track (54in) at given duty cycle	665	628	601	594	531	550	528
	672	621	622	617	549	588	534
	657	650	620	594	521	558	538
	722	668	630	602	593	544	536
	696	651	635	603	588	585	583
	694	650	640	625	578	583	530
	697	684	661	611	579	586	579
	740	665	678	617	582	582	575
	737	643	646	627	581	589	556
	715	664	646	651	589	576	560
	729	654	640	657	582	583	578
	691	673	664	670	593	579	562
	721	727	652	665	593	596	570
	682	688	662	659	595	615	575
	667	724	659	661	591	586	577
	718	695	689	667	584	583	576
	716	709	648	674	597	560	588
725	721	644	670	614	580	537	
Average Time [ms] =	702	673	647	637	580	579	560

**Summary of Experimental data of slot car time per duty cycle**

On Time (clock cycles)	PWM Duty Cycle = $\frac{\text{On Time}}{65535}$	Average time [ms]	Average speed [ft/s]
20000	31	702	6.41
21000	32	673	6.69
22000	34	647	6.96
23000	35	637	7.06
24000	37	595	7.56
25000	38	580	7.76
26000	40	560	8.04

## 7. Features



1. Main power connection
2. Main power switch
3. Serial port connection
4. Manual controller connection (2)
5. LCD display
6. LED speed trees

7. Auto/Manual select switch
8. Difficulty select switch
9. Reset button
10. Start button
11. Storage bin

## 8. Team Members



**Ryan Krauss**  
Electrical Design  
Circuit Construction  
Testing



**Joe Frankel**  
Mechanical Design  
Machining & Fabrication  
Comparator Circuit  
Webpage



**Lisa Ellis**  
Logic & Software Design  
Programming  
Testing



## 9. Sponsors

### Donating Sponsors

#### *Johnson Controls*

Special thanks is due to our corporate sponsor, Johnson Controls, without whose support the project would have certainly been less successful. We thank them for purchasing the box, LCD display, power supply, and other misc. parts.



#### *Omron*

Thanks also go out to Omron for donating the photosensors.



### Consultants

Bao Mi, Mechatronics T.A.  
Akio Kita, Mechatronics T.A.  
John Graham, M.E. Machine Shop  
Butch Cabe, M.E. Fabrication Shop  
Sterling Skinner, Instructional Laboratory Director

### Suppliers

Johnson Controls  
Newark Supply Co  
DigiKey  
Omron  
McMaster-Carr  
Radio Shack

Home Depot  
Laird Plastics  
Hobbytown USA  
Ack Radio  
Motorola  
Lambda Electronics

---

## 10. Appendix: C code

```

#include <stdio.h>
#include <math.h>
#include <hcl1e9.h>

#define LINE_1      0x80          // beginning position of LCD line 1
#define LINE_2      0xC0          // beginning position of LCD line 2
#define LINE_3      0x94          // beginning position of LCD line 3
#define LINE_4      0xD4          // beginning position of LCD line 4
#define DELAY1MS    73           // number of loops = 1ms

// I/O Port Addresses
#define LCD_CMD     *(unsigned char *) (0xB5F0)
#define LCD_DAT     *(unsigned char *) (0xB5F1)

// Function Prototypes
void LCD_Command(unsigned char cval);
void LCD_busy(void);
void cprint(char dval);
void LCDprint(char *sptr);
void delay100ms(unsigned int secs);
void delay1ms(unsigned int msec);
float ReadVoltage(int BitChoice);
void CheckDiffclty(void);
void Initialize(void);
void Start(void);
void AvgVoltageFromCars(void);
void EndOfRace(void);
void InitDelayforOC3(int NumOF, int TimeOF, int PWM);
void CalcNumOverflows(void);
char ArrayLine1[20];
char ArrayLine2[20];
char ArrayLine3[20];
char ArrayLine4[20];

/*=====*/
/* Define Global Variables */

int On = 1;
int RaceStart = 1;
int ReadyToRace;
int Count;
int Corner = 1;
int Lap1;
int ULapNum, UOverflowIndex, UEdgeIndex;
int CLapNum, COverflowIndex, CEdgeIndex;
int PWMStraight, PWMCorner, PWMStraightSlow, PWMOff = 1, PWMFull = 65500;
int ADVRH = 5;
int NumOverflow, OverflowCounter, StartOC3Counter = 0;
int NumStartOverflow, NumStrtOverflow, NumCornerOverflow;
int StartOverflowTime, StrtOverflowTime, CornerOverflowTime;
int LapNum1, LapNum2;
float InputStartOverflowTime = 225;
float InputStrtOverflowTime = 225; /*Enter time in milli seconds*/
float InputCornerOverflowTime = 100; /*Enter time in milli seconds*/

```

---

```

int OutOfCorner = 0, IntoCorner = 0;
float UEdge1, UEdge2, ULapTime, UTotalLapTime;
float CEdge1, CEdge2, CLapTime, CTotalLapTime;
float UCarVoltage, UCarVelocity;
float CCarVoltage, CCarVelocity;
float PE2Level;
float StraightTime, CornerEdge;
float BestCLapTime, BestULapTime;
int JumpedGun;
int Time;
int PWMSelection;
int CheckCarLocn;

/*=====*/

void main(void)
{
    asm("\tCLI");

    // Initialize the LCD
    LCD_Command(0x3C);           // initialize command
    LCD_Command(0x0C);           // display on, cursor off
    LCD_Command(0x06);
    LCD_Command(0x01);

    while(1)
    {
        if(On == 1)
        {
            LCD_Command(0x01);
            LCD_Command(LINE_1);
            LCDprint("  Welcome to the");
            LCD_Command(LINE_2);
            LCDprint("    Auto Slot 20");
            delay100ms(10);
        }

        /* If Compter control turned on, ie PD4 is high */
        if( (PORTD & 0x10) && RaceStart == 1)
        {
            LCD_Command(0x01);
            LCD_Command(LINE_1);
            LCDprint("Push reset to start");
            LCD_Command(LINE_2);
            LCDprint(" race");
            delay100ms(10);
        }

        /* If Computer control off */
        if( !(PORTD & 0x10) && RaceStart == 1)
        {
            LCD_Command(0x01);
            LCD_Command(LINE_1);
            LCDprint("To race automated");
        }
    }
}

```

```

        LCD_Command(LINE_2);
        LCDprint(" car, flip switch.");
        delay100ms(10);
    }

    Count = 0;
    On = 0;

    /* If buttons pushed and Computer control on or PD4 is high */
    while(ReadyToRace == 1)
    {
        if(On == 0)
        {
            LCD_Command(0x01);
            sprintf(ArrayLine1,"Your Speed: ");
            LCD_Command(LINE_1);
            LCDprint(ArrayLine1);
            sprintf(ArrayLine2,"Lap #    Time:");
            LCD_Command(LINE_2);
            LCDprint(ArrayLine2);
            sprintf(ArrayLine3,"Auto Speed: ");
            LCD_Command(LINE_3);
            LCDprint(ArrayLine3);
            sprintf(ArrayLine4,"Lap #    Time:");
            LCD_Command(LINE_4);
            LCDprint(ArrayLine4);
            On = 2;
            delay100ms(2);
        }

        if(Count == 10000)
        {
            AvgVoltageFromCars();
            Count = 0;
        }

        ++Count;

        if(ULapNum == 21 || CLapNum == 21)
        {
            EndOfRace();
        }
    }
}

/*=====*/
/*Function that calculates number of needed overflows*/

void CalcNumOverflows(void)
{
    InputStartOverflowTime = 2000*InputStartOverflowTime/65535;
    NumStartOverflow = floor(InputStartOverflowTime);
    InputStartOverflowTime = 65535*(InputStartOverflowTime -
NumStartOverflow);
    StartOverflowTime = floor(InputStartOverflowTime);
}

```

```

    InputStrtOverflowTime = 2000*InputStrtOverflowTime/65535;
    NumStrtOverflow = floor(InputStrtOverflowTime);
    InputStrtOverflowTime = 65535*(InputStrtOverflowTime -
NumStrtOverflow);
    StrtOverflowTime = floor(InputStrtOverflowTime);

    InputCornerOverflowTime = 2000*InputCornerOverflowTime/65535;
    NumCornerOverflow = floor(InputCornerOverflowTime);
    InputCornerOverflowTime = 65535*(InputCornerOverflowTime -
NumCornerOverflow);
    CornerOverflowTime = floor(InputCornerOverflowTime);
}

/*=====*/
/*Interupt functions for reset*/

__interrupt void Reset_IRQ(void)
{
    OC1D = 0x00;          /* Disable OC1 so don't output to PA6 */
    OC1M = 0x00;

    StartOC3Counter = 0;
    RaceStart = 1;
    /* If computer control on then tell user to hit start and wait until they
do */
    if(PORTD & 0x10)
    {
        RaceStart = 0;
        ReadyToRace = 0;
        LCD_Command(0x01);
        LCD_Command(LINE_1);
        LCDprint("Line up Cars.");
        LCD_Command(LINE_2);
        LCDprint("Press start when");
        LCD_Command(LINE_3);
        LCDprint("  ready to race.");

        /* Wait until user hits start button connected to PD5*/
        while(!(PORTD & 0x20)){

        Start();
        return;
    }
    else
    {
        LCD_Command(0x01);
        LCD_Command(LINE_1);
        LCDprint("To race automated");
        LCD_Command(LINE_2);
        LCDprint(" car, flip switch.");
        return;
    }
}

/*=====*/
/*Function for the Start of program*/

```

```

void Start(void)
{
    int Countdown = 5;
    JumpedGun = 0;
    RaceStart = 0;
    ReadyToRace = 1;
    On = 0;
    BestCLapTime = 0;
    BestULapTime = 0;
    LapNum1 = 0;
    LapNum2 = 0;

    CheckDiffclty();

    // Enable sensor interrupts in case user jumps start */
    asm("\tSEI");
    PACTL = 0x04;          /* Make PA3 be IC4F */
    TMSK1 = 0x0F;         /* Enable Interrupts for IC1F-IC4F */
    TCTL2 = 0xAA;         /* IC1F - IC4F will capture falling edge */
    TFLG1 = 0xCF;         /* Clear flags for IC1F-IC4F, OC2, and OC1 */
    asm("\tCLI");

    LCD_Command(0x01);
    LCD_Command(LINE_1);
    LCDprint("Are you ready to");
    LCD_Command(LINE_2);
    LCDprint("eat automated dust!");
    delay100ms(10);

    LCD_Command(0x01);

    while(Countdown > 0)
    {
        sprintf(ArrayLine2,"          %i",Countdown);
        LCD_Command(0x01);
        LCD_Command(LINE_2);
        LCDprint(ArrayLine2);
        delay100ms(5);
        --Countdown;
    }

    InitDelayforOC3(NumStartOverflow, StartOverflowTime, PWMOFF);

    Initialize();
    LCD_Command(LINE_2);
    LCDprint("          Go");

    JumpedGun = 1;

    return;
}

/*=====*/
/* Calculate how many overflows need */
void InitDelayforOC3(int NumOF, int TimeOF, int PWM)
{
    Time = TCNT;

```

---

```

    Time = Time + TimeOF;
    if(Time > 65535)
    {
        Time = Time - 65535;
        ++NumOF;
    }
    NumOverflow = NumOF;
    OverflowCounter = 0;
    StartOC3Counter = 1;
    PWMSelection = PWM;
}

/*=====*/
/*Interrupt function for the counting number of overflows*/

__interrupt void OverFlow(void)
{
    ++COverflowIndex;
    ++UOverflowIndex;

    if(StartOC3Counter == 1)
    {
        if(OverflowCounter == NumOverflow)
        {
            StartOC3Counter = 0;
            OverflowCounter = 0;

            asm("\tSEI");
            TMSK1 = 0x2F;
            TFLG1 = 0xEF;
            asm("\tCLI");

            TOC3 = Time;
            TFLG1 |= 0x20;
        }

        ++ OverflowCounter;
    }

    TFLG2 |= 0x80;      /* Clear Overflow flag */
    return;
}

/*=====*/
/* Interrupt that changes PWM after given number of overflows */

__interrupt void StopCar_OC3(void)
{
    TOC2 = PWMSelection;
    TMSK1 &= 0x0F; /* Disable OC3 interrupt */
    return;
}

/*=====*/

/* Intialize interrupt function */

```

---

```

void Initialize(void)
{
    /* Initialize variables */
    UOverflowIndex = 0;
    COverflowIndex = 0;
    UEdgeIndex = 1;
    CEdgeIndex = 1;
    ULapNum = 0;
    CLapNum = 0;
    UTotalLapTime = 0;
    CTotalLapTime = 0;
    ULapTime = 0;
    CLapTime = 0;
    Lap1 = 1;
    Corner = 1;

    /*This section initializes AD port*/
    OPTION &= 0xBF;          /* Clear CSEL bit */
    OPTION |= 0x80;          /* Enable A/D */
    TFLG1 = 0x40;           /* Clear OC2 flag */
    TOC2 = TCNT + 200;      /* Start 100 micro sec delay */
    while(!(TFLG1 & 0x40)){ /* Wait for delay */

    // Initialize the LCD
    LCD_Command(0x3C);      /* initialize command
    LCD_Command(0x0C);      /* display on, cursor off
    LCD_Command(0x06);
    LCD_Command(0x01);

    /*This section initializes ports and interrupts*/
    asm("\tSEI");           /* Set Interrupt Mask */

    DDRD = 0x00;           /* Ensure that port D low */

    PACTL = 0x04;          /* Make PA3 be IC4F */

    TMSK1 = 0x0F;          /* Enable Interrupts for IC1F-IC4F*/
    TMSK2 = 0x80;          /* Enable overflow interupt */

    TCTL1 = 0x80;          /* Turn PA6 off when OC2 reaches desired time */
    TCTL2 = 0xAA;          /* IC1F - IC4F will capture falling edge */
    TFLG1 = 0xCF;          /* Clear flags for IC1F-IC4F, and OC1 */
    TFLG2 = 0x80;          /* Clear overflow flag */

    OC1D = 0x40;           /* Make pin that OC1 controls (PA6) high w/
successful OC1 compares*/
    OC1M = 0x40;           /* OC1 will control PA6 */

    TOC1 = 0;              /* When TOC1 = 0 set output high on PA6 */
    TOC2 = PWMstraight;    /* Define time for PA6 to be high */

    return;
    asm("\tCLI");
}

/*=====*/

```

```

/* This function checks difficulty level and redefines PWM accordingly*/

void CheckDiffclty(void)
{
    PE2Level = ReadVoltage(2);

    if(PE2Level < 1)
    {
        InputStartOverflowTime = 460; /* Input time in ms */
        InputStrtOverflowTime = 460;
        InputCornerOverflowTime = 40;
        PWMStraight = 35000;
        PWMCorner = 27000;
    }
    else if(PE2Level < 4)
    {
        InputStartOverflowTime = 240; /* Input time in ms */
        InputStrtOverflowTime = 240;
        InputCornerOverflowTime = 75;
        PWMStraight = PWMFull;
        PWMCorner = 34000;
    }
    else if(PE2Level > 4)
    {
        InputStartOverflowTime = 240; /* Input time in ms */
        InputStrtOverflowTime = 240;
        InputCornerOverflowTime = 75;
        PWMStraight = PWMFull;
        PWMCorner = 34000;
    }

    /* Convert overflow time to clock cycles */
    CalcNumOverflows();

    return;
}

/*=====*/
/* Interrupt function that looks at into corner sensors for Cmpter car */

__interrupt void IntoCorner_IC2F_PA1(void)
{
    //printf("22\n");
    CornerEdge = TIC2;
    TOC2 = PWMOff;

    InitDelayforOC3(NumCornerOverflow, CornerOverflowTime, PWMCorner);

    if(Lap1 == 1)
    {
        CEdgeIndex = 1;
        UEdgeIndex = 1;
        ULapNum = 1;
        CLapNum = 1;
        Lap1 = 0;
    }
}

```

```

    TFLG1 |= 0x02;      /* Clear flag for IC2 at PA1 */
    return;
}

/*=====*/
/* Interrupt function that looks at Out of corner sensors for cmpter car */

__interrupt void OutOfCorner_IC1F_PA2(void)
{
    //printf("333\n");
    TOC2 = PWMStraight;    /* Change PWM for straight away */

    InitDelayforOC3(NumStrtOverflow, StrtOverflowTime, PWMOff);

    TFLG1 |= 0x04;      /* Clear flag */
    return;
}

/*=====*/
/* Interrupt function that looks at Out of corner sensors at start line for
cmpter car */

__interrupt void OutOfCornerStart_IC3F_PA0(void)
{
    //printf("1\n");
    TOC2 = PWMStraight;    /* Change PWM for straight away */
    CEdge2 = TIC1;        /* Record edge */

    InitDelayforOC3(NumStrtOverflow, StrtOverflowTime, PWMOff);

    if(CEdgeIndex == 2)
    {
        CLapTime = 0.0328 * ( COverflowIndex + ((CEdge2-CEdge1)/65535));

        if(LapNum1 == 0)
        {
            BestCLapTime = CLapTime;
            LapNum1 = 1;
        }

        if(CLapTime < BestCLapTime)
        {
            BestCLapTime = CLapTime;
        }
        LCD_Command(0x01);
        LCD_Command(LINE_4);
        sprintf(ArrayLine4,"Lap %i Time: %.2f s",CLapNum, CLapTime);
        LCDprint(ArrayLine4);
        LCD_Command(LINE_2);
        LCDprint(ArrayLine2);
        LCD_Command(LINE_1);
        LCDprint(ArrayLine1);
        LCD_Command(LINE_3);
        LCDprint(ArrayLine3);

        ++CLapNum;
    }
}

```

```

    CTotallapTime = CTotallapTime + CLapTime;
}

CEdgeIndex = 2;
COverflowIndex = 0;
CEdge1 = CEdge2;

TFLG1 |= 0x01;      /* Clear flag */
return;
}

/*=====*/
/* Interrupt function that looks at Out of corner sensors for User car */

__interrupt void UOutofCorner_IC4F_PA3(void)
{
    //printf("User\n");
    if(JumpedGun == 0)
    {
        LCD_Command(LINE_2);
        LCDprint("You jumped the gun");
        LCD_Command(LINE_3);
        LCDprint("Press Reset");
        OC1D = 0x00;      /* Disable OC1 so don't output to PA6 */
        OC1M = 0x00;
        return;
    }

    UEdge2 = TIC3;      /* Record edge */

    if(UEdgeIndex == 2)
    {
        ULapTime = 0.0328 * ( UOverflowIndex + ((UEdge2-UEdge1)/65535) );
        sprintf(ArrayLine2,"Lap %i Time: %.2f s",ULapNum, ULapTime);
        LCD_Command(0x01);
        LCD_Command(LINE_2);
        LCDprint(ArrayLine2);
        LCD_Command(LINE_1);
        LCDprint(ArrayLine1);
        LCD_Command(LINE_3);
        LCDprint(ArrayLine3);
        LCD_Command(LINE_4);
        LCDprint(ArrayLine4);

        if(LapNum2 == 0)
        {
            BestULapTime = ULapTime;
            LapNum2 = 1;
        }

        if(ULapTime <= BestULapTime)
        {
            BestULapTime = ULapTime;
        }
    }
}

```

```

    ++ULapNum;
    UTotalLapTime = UTotalLapTime + ULapTime;
}

UEdgeIndex = 2;
UOverflowIndex = 0;
    /*Variables start at zero*/
UEdge1 = UEdge2;

TFLG1 |= 0x08;    /* Clear flag */
return;
}

/*=====*/
/* Function reads voltage from different Pins: PE0 = UCar, PE1 = CCar, PE2 =
Difficulty level*/

float ReadVoltage(int BitChoice)
{
    float AvgVoltage = 0, sum = 0;

    switch(BitChoice)
    {
        case 0:
            ADCTL = 0x00;    /* Read pin PE0 */
            break;
        case 1:
            ADCTL = 0x01;    /* Read pin PE1 */
            break;
        case 2:
            ADCTL = 0x02;    /* Read pin PE2 */
            break;
    }

    while(!(ADCTL & 0x80)){    /* Wait for CCF to be set */
        sum = ADR1 + ADR2 + ADR3 + ADR4;
        AvgVoltage = sum/4;
        AvgVoltage = (AvgVoltage/255)*ADVRH;
        return(AvgVoltage);
    }

    /*=====*/
    /* This function calls the ReadVoltage function multiple times to avg Voltage
reading and*/
    /* Convert to a velocity */

void AvgVoltageFromCars(void)
{
    CCarVoltage = 100*ReadVoltage(0);
    UCarVoltage = 100*ReadVoltage(1);

    sprintf(ArrayLine1,"Your speed: %.0f mph",UCarVoltage);
    sprintf(ArrayLine3,"Auto speed: %.0f mph",CCarVoltage);
    LCD_Command(0x01);
}

```

```

LCD_Command(LINE_1);
LCDprint(ArrayLine1);

LCD_Command(LINE_3);
LCDprint(ArrayLine3);

LCD_Command(LINE_2);
LCDprint(ArrayLine2);

LCD_Command(LINE_4);
LCDprint(ArrayLine4);
}

/*=====*/
/* This function converts given Voltage to Velocity */

float CvtVoltToVel(float Voltage)
{
    float velocity;

    /* Create some table that says certain voltage corresponds to certain
Velocity */
    return velocity;
}

/*=====*/
/* This function handles end of race */

void EndOfRace(void)
{
    RaceStart = 1;
    ReadyToRace = 0;

    asm("\tSEI");
    OC1D = 0x00;          /* Disable OC1 so don't output to PA6 */
    OC1M = 0x00;
    TMSK1 = 0x00;        /* Disable sensor interrupts */
    TFLG1 != 0xCF;
    TCTL2 = 0xAA;
    asm("\tCLI");

    if(ULapNum > CLapNum)
    {
        LCD_Command(0x01);
        LCD_Command(LINE_2);
        LCDprint("      You Win");
        delay100ms(20);
    }
    else if(CLapNum > ULapNum)
    {
        LCD_Command(0x01);
        LCD_Command(LINE_2);
        LCDprint("      You Lose");
        delay100ms(20);
    }
}

```

```

    if(UTotalLapTime == 0)
        UTotalLapTime = 0;
    else
        UTotalLapTime = UTotalLapTime/(ULapNum-1);
    if(CTotalLapTime == 0)
        CTotalLapTime = 0;
    else
        CTotalLapTime = CTotalLapTime/(CLapNum-1);

    LCD_Command(0x01);
    LCD_Command(LINE_1);
    LCDprint("Your Avg Lap Time:");
    LCD_Command(LINE_2);
    sprintf(ArrayLine2,"          %.2f s", UTotalLapTime);
    LCDprint(ArrayLine2);
    LCD_Command(LINE_3);
    LCDprint("Auto Avg Lap Time:");
    LCD_Command(LINE_4);
    sprintf(ArrayLine4,"          %.2f s", CTotalLapTime);
    LCDprint(ArrayLine4);
    delay100ms(10);

    LCD_Command(0x01);
    LCD_Command(LINE_1);
    LCDprint("Your Best Lap Time:");
    LCD_Command(LINE_2);
    sprintf(ArrayLine2,"          %.2f s", BestULapTime);
    LCDprint(ArrayLine2);
    LCD_Command(LINE_3);
    LCDprint("Auto Best Lap Time:");
    LCD_Command(LINE_4);
    sprintf(ArrayLine4,"          %.2f s", BestCLapTime);
    LCDprint(ArrayLine4);
    delay100ms(10);

    return;
}

/*=====*/
// Wait for the LCD busy pin to clear
void LCD_busy(void)
{
    while ((LCD_CMD & 0x80)) ;
}

/*=====*/
void LCD_Command(unsigned char cval)
{
    //unsigned char cval;
    LCD_busy();                // wait for busy to clear
    LCD_CMD = cval;           // ouput command
}

/*=====*/
/* LCD Display Character
void cprint(char dval)
{

```

---

```

    LCD_busy();                // wait for busy to clear
    LCD_DAT = dval;           // ouptut data
}

/*=====*/
// LCD Display String
void LCDprint(char *sptr)
{
    while( *sptr )
    {
        cprint(*sptr);
        ++sptr;
    }
}

/*=====*/
// Busy wait loop to generate a delay multiple of 1ms
void delay1ms(unsigned int msec)
{
    unsigned int i,j;

    if (msec > 0) {
        for (j=0; j<=msec; j++)
            for (i=0; i<DELAY1MS; i++);
    }
    return;
}

/*=====*/
// Busy wait loop to generate a delay multiple of 100ms.
void delay100ms(unsigned int msec)
{
    unsigned int i,j;

    if (msec > 0) {
        for (j=0; j<=msec; j++)
            for (i=0; i<100; i++) delay1ms(1);
    }
}

/*=====*/
/* Define interrupt vectors */
asm("\tSWI");

asm("\tORG $00D0");
asm("\tJMP Overflow");

asm("\tORG $00E2");
asm("\tJMP OutOfCornerStart_IC3F_PA0");

asm("\tORG $00E8");
asm("\tJMP OutOfCorner_IC1F_PA2");

asm("\tORG $00E5");
asm("\tJMP IntoCorner_IC2F_PA1");

asm("\tORG $00D3");

```

---

```
asm("\tJMP UOutOfCorner_IC4F_PA3");
```

```
asm("\tORG $00EE");  
asm("\tJMP Reset_IRQ");
```

```
asm("\tORG $00D9");  
asm("\tJMP StopCar_0C3");
```