THE UNIVERSITY OF QUEENSLAND

School of Information Technology and Electrical Engineering

Digital Slot Car System

by

Scott Bremner

The School of Information Technology and

Electrical Engineering

The University of Queensland

Submitted for the degree of Bachelor of Engineering in the division of Computer Systems

29 October 2003

Scott Bremner
4/24 Brisbane Street
St. Lucia, Brisbane
QLD 4067 Australia

29 October 2003

Head of School,
School of Information Technology and Electrical Engineering
The University of Queensland
St. Lucia QLD 4072

Dear Professor Simon Kaplan,

In accordance with the requirements of the Degree of Bachelor of Engineering in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

"Digital Slot Car System".

This thesis was preformed under the supervision of Richard Cocks. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Scott Bremner

Table Of Contents

1.0 INTRODUCTION

As we get caught up in our busy lives, it's good sometimes to put our work aside and let our selves have a bit of fun. There are many exciting activities that we can choose that will entertain us, such as sports, games, media (i.e. movies, theatre, etc) and hobbies. We often choose several activities that we can spend our time on, some being social activities and some done in solitude.

This project takes a look at one of these activities that has been very popular in the past. This activity is *'Slot car racing'*. Slot car racing is an activity that involves car racing in a smaller scale around a track. Slot cars and tracks usually resemble real sports/race cars and race tracks. The typical definition is a small 'car' running a DC motor is propelled around a circuit, using a guide pin, controlled by a resistive element controller which varies the voltage on the track, which controls the speed of the car. The trick to this is keeping the car on the track while trying to go as fast as possible. When a car reaches a bend in the track we have to slow the car down or the car will flip off the track.

In the late 1960's, slot car racing had a substantial following with lots of commercial support. There were large commercial raceways with factory backed racing teams and sponsorship with commercial raceways in the US earning up to $500 million a year. After several years of great success the slot car industry became to big and expensive to support it's self. A professional slot car would cost up to $500US and still had to be fine tuned to be competitive. This lost the support of the public and in the US, where there went from being around 3000 commercial raceways to about 50. Since then slot car slowly regained some support with commercial raceways being available in most cities throughout western society. Although the major use of slot car sets today is in the toy market.

This project looks at the basic toy slot car set and aims to make it more entertaining. There is lots of competition on the toy market place with the likes of board games, computer/video games and many other toys all looking to take a chunk of peoples time. Computer/video games seem to be a major threat to slot car sets today with very appealing car racing games. These games offer highly realistic graphics with a majority of features of real racing included, such as overtaking, car setup, a high level of concentration and negotiating other cars. This leaves slot car sets looking a little boring. However, by adding a few features to our slot car sets, we hope to regain some of the lost interest in slot cars and make them more exciting.

The project description as given on the Thesis website is as follows:

"Design a digital control system to allow multiple slot cars to race on the same lane of a slot car track. Provision for overtaking, and change of direction will be required. The final solution should allow a dual lane racing track to support four of more individual drivers / cars."

By adding these extra features, we will be adding more user control, a higher level of concentration and options to slot racing. This projects was aimed at making slot car racing more entertaining by adding features like passing, multiple cars per lane and changes in direction. You only have to follow a form of racing such as formula 1 to realise how passing can effect the level of entertainment in racing. In 1998, the FIA (governing body of formula 1 and international racing) changed the rules of the sport to reduce passing. Although this was in the interest of safety, it had many followers of the

sport complaining about the level of 'entertainment'. So, if there are multiple cars per lane on a slot car set there must also be prevision for passing.

The current solution we have arrived at addresses the main problem involved with achieving these features. The problem with current slot car sets is that cars cannot be controlled individually in a single lane. Our solution explores various concepts in digital control and applies them to the slot car set. Now we have a means of digital control we can later look into completing our digital communication to the cars and building the infrastructure, which will allow overtaking and changes of direction.

This document looks at solutions we have achieved and the solutions that are yet to be implemented in the design of the ‘Digital Slot Car System’. We will explore some history of slot cars and work that has been done in this area to date. We will also look into analyse what has been done and the work that should follow.

2.0 REVIEW OF PREVIOUS WORK

2.1 History of the Slot Car

Slot car sets were derived from model trains, with the first appearing in 1908. This slot car set consisted of a windup car on a small circular track. Later on a German model train manufacturer made a set that was essentially a car bodied model train set. The track was more intricate winding between tin buildings. In 1912, Lionel made a track that ran the two cars side by side on a single rail, powered by two variable transformers. In 1935, Charles Woodland found a method that allowed a car to drift or spin across the track. By using mainly toy train parts, he built a race car set that used a slot instead of a rail. It took Charles until 1949 to develop the idea that spur others to follow on from his concepts.

Mathieu Canellas designed a system in 1940, which used a brass-pivoting guide that ran in a slot that was attached to a positive terminal on a 12 volt power supply. The negative terminal connected to a rail that ran along the surface of the track, which a copper pad contact would slide along. Multilane crossover was achieved by a small tab on the rear of the cars operating cables over the top of the track.

In 1948, B F Francis began producing clockwork race cars under the brand name SCALEX, and four years later had a patent for keyless clockwork. When sales began to drop off for his clockwork cars in 1957, he started to develop an electric slot car system that ran on rubber tracks and renamed them SCALEXTRIC.

In the 60’s a change was made toward plastic bodied cars rather then tin. This also led to the use of plastic tracks that had better conductivity. Scalextric have also added features to their cars such as rubber tires, braking systems and Magnatraction cars with lights.

Between the mid 60's and mid 80's a company named FALLER A.M.S. developed HO slot car sets with many interesting features. His sets fell into two categories, racing sets and traffic sets. The racing sets had features such as high gearing in the cars for extra speed, crossover tracks, loop tracks and banked tracks. The traffic sets were developed partly as accessories for model train sets but had many features. These include most of the things you might find on while driving, such as traffic lights, intersections petrol stations, operating railway crossings and car washes with automatic doors. They also had parts specifically for model train sets with ramps to drive a car onto the back of a rail wagon and operating container terminals that could move containers between slot 'trucks' and rail wagons.

In the late 60's slot car racing made a big boom in popularity with commercial raceways in the US earning up to $500 million a year, with big competitions and manufacture sponsored teams. However, after a few years the hobby became too expensive with professional cars costing up to $500 US, even then they had to be tuned and the motors hand wound. This drove off youngsters and in the US there went from being around 3000 commercial raceways to about 50.

Slot car picked up again in the late 70's with companies manufacturing good performing cars at a reasonable price. The market has now stabilised with many people having own or played on toy slot car sets.

2.2 Davic System

The Davic system is a design that that was developed several years ago. It uses a lap counter computer program to keep track of a car around a track. There is some kind of microchip – possibly a sensor arrangement – that is attached to each car. The lap counter has sensors in the track, which reports a cars position back to the software. There is also a button attached to each persons hand controller, which controls switches in the track to direct the car onto a different lane.

This system has been set up in a raceway in France, and allows up to eight cars on one track at a time. The system designer is currently working on increasing the number of cars per track to 12. One problem with the system is that when switching lanes, the user must be careful not to switch when there car is too close to another car as the track will sometimes switch other close cars at the same time.

2.3 Digital Train Sets

Marklin have produced a digital train system where a multiple trains run of the one track using a single supply. This is achieved by each train carrying a receiver and having a unique address. A Central control unit continually sends commands along the rails of the track to be picked up by the receiver on the trains. Each train compares the address in the command with its own address, to determine whether the command belongs to it, if it does the train runs the command. This system has very few of the problem that a slot car system would have to face. A train on a track should have a continuous connection with the track and doesn't run at the speeds that a slot car does. This would require a slot car system to run much faster communication rate then a train system.

2.4 Cars

Slot cars come in several different sizes from HO scale cars that are around 50mm to 1/32 and 1/24 scale sizes (compared to real cars). The AFX track used in the project is a HO scale. The cars vary in categories from vintage, which are essentially cars about 30 years old, through to professional cars that don't use a body but have large wings for aerodynamics. Initially car bodies were made of tin, but around the mid 60's, they started making plastic bodies. The main purpose of the body is for decoration, so they resemble various real cars. There are F1 style bodies as well as vintage, road car and even truck and motorbike bodies. The second function of the body is to protect the electronics inside from damage when the cars come off the track and hit walls etc.

Most of the chassis under the bodies are to hold the axles, body and motor. For most sets, these are made of plastic, but once you get into the professional hobby racing end of the market you find metal chassis that allow tuning for weight distribution and handling. The tires are usually made from rubber.

2.5 Motors

Motors are one of the most important parts of the slot car. There are a large range of different motors used in slot cars from mass produced types to hand wound motors. Scratchbuilt break motors down into three classes, pre-can motors, can motors and strap motors. Pre-can motors are the old type of motors they used in developing the slot cars. They were mainly used in model train sets. Can motors are the common type of motor used today. They are mass-produced and have the armature coming out the can side. The Strap motor is basically a can motor that has been cut down to the minimum essential

parts. They are usually hand wound and have expensive high powered magnets. There are also other motors used such as the bell type, which has the armature coming out the bell side, and the dual shaft type that has the armature coming out both sides.

Before electric motors were used in slot cars, slot cars were propelled by clockwork motors. These allowed very little interaction, as the only user input was to wind the motor.

The motors used in our slot cars propel the cars fast enough to be fun but not competitive in club racing. As we have built this more for the toy market, it is not a high priority to have a super fast motor.

2.6 Tracks

There are quite a few different track types as most slot car manufactures made there own tracks. Even among the same size of cars there can be different sized tracks. Most tracks are made from hard plastic, but some are made from rubber or soft flexible plastic like the Scalextric sports track. Some of the commercial raceways even make there own tracks from wood. There are many different track pieces too, such as loops, banked corners, crossover sections and even jumps. Many track sets come in oval, figure 8 or a replica of a real race track. Tracks need to be highly insulated so that power rails can be run along them without too much loss in power. Professional tracks have high friction surfaces to help give the car greater cornering speeds.

2.7 Controllers

There are two types of controllers used today, the resistive type and the semi-conductor type. The resistive type is essentially the same as what has been used for over 40 years. Resistive controllers are a very simple design, essentially consisting of a variable resister. The problem with this design is that different motor classes require different resistor values, which means if you change the motor you may also have to change the resistivity of the controller. Semi-conductor type controllers on the other hand work independently to the load on the circuit, with the output voltage being the product of the number of semi-conductors used in the circuit and 0.6 volts that goes through each semi-conductor. Semi-conductor type controllers are also polar, so they block current flow in the negative direction.

2.8 Timers / Lap Counters

Most lap counters are designed to count one car in each lane. Therefore, to use a lap counter or timer for this project a new program would have to be written or the interface to one of the programs changed so that the data is collected for each car not each lane.

There are many lap counters and timers currently available, from basic mechanical counter then work on a trigger in each lane, to software counters/timers that include sector timings, data logging and statistical analysis.

3.0 DERIVE THE SPECIFICATIONS

As this project is about entertainment, we need to look at every design decision and ask some questions. Will this increase the 'fun factor'? Does a feature add to the users experience? I will start with the design specifications of the original product and then detail the changes in specification using a top down approach and explain how they achieve our goal.

We start with an AFX brand HO scale slot car set that includes enough track pieces to make a figure eight track, two hand controllers, two cars and the power supply.

The main feature we had aimed to implement was the ability to run multiple cars on the one track. This cannot be done on the original slot set as the energy on the track varies in relation to the input from the hand controller. Hence putting two cars on the one track would cause both cars to receive the same amount of energy and hence travel at the same speed. For multiple cars to run on one track we must specify that the car take from the track the amount of energy that they require. This requires us to sample the energy from the hand controllers package the data and send it to the cars. The cars then determine from this data, how much energy they will feed to the motors from the rails on the track. This means we must have a 'control unit' that can sample the energy from the hand controller and send the data to the car via the track. The energy from the hand controller is a varying voltage as the hand controller contains a variable resistor element. An 8-bit micro with on chip ADC can sample this data very quickly and package it up in a data packet to send to the track.

To transfer data between the control unit and the slot car we must a means of means of transferring the data. The rails on the slot car track are used for sending the energy to the cars. As we do not want to add more wires to the track (this would stop us from using standard tracks pieces for this project), we had two options for sending data – wireless communication (i.e. radio, infra-red) or encoding the data onto the power signal on the rails. We have used the second method and have designed a circuit that combines the power and data signal and sends them to the track. The standard power supply that comes with the track is barely able to run the standard circuit. We had to use a larger power supply as we have added more to our circuit, which will use more power, and have specified that we should be able to run up to four cars in the circuit.

The car now has to receive a signal from the rails and decode it to determine how much energy to allow the motor to use. At this point, we need to define the type of communication used. We have chosen to use a digital signal as it is less influenced by noise. There are many ways to send data but the method we have chosen uses a two voltage levels and switches between the higher and the lower level to signify 1's and 0's respectively. We also have to look at the speed of communication. We would like our slot car to have no noticeable reduction in response over the original design. If the response of the car is much less then the human response time, we shouldn't notice a lack of response in the system. Through testing, we found our fastest response to be over 100ms from zero power to full power to zero again. We then halved this result and used this for our minimum time to transfer the at least two data packets. We transfer two in case one packet is missed the cars will still have the desired response as the cars often loose contact with the track for brief amounts of time. We define our data packet to be a sync byte, four data bytes (one for each car in order of car number), and two error checking bytes.

To read the signal the car first has to filter out the data signal from the power signal. We use a simple clamping circuit and then send the data to an 8-bit micro. We then use pulse width modulation (PWM) to send the data to the motor driver circuit.

We also connect the rails together on the track so that each lane of the track has the same signal. This allows us to run a car on either track without changing the hand controller, as a hand controller input relates to a car not a lane on the track. This enables us to use lane changing once the lane changing system is developed.

4.0 HARDWARE IMPLEMENTATION

The hardware for this project was the responsibility of Nick Cootes. There are two main parts of the hardware, the first being the control unit and the second being the car circuit. The car circuit contains a MOSFET driver to drive the PWM signal to the motor. There is also a Voltage regulator and a bridge rectifier to allow the high voltage level to be seen by the motor and a 5V power supply for the micro. The voltage seen by the motor is also smoothed by a large capacitor to try to remove some of the effects of having the transmission signal encoded onto the power signal. The signal that is encoded onto the power signal on the track is also separated from the power circuit via a basic clamping circuit. This signal is then sent to the input of the micro for the signal regeneration.

The control unit circuit contains a resistor arrangement that ensures the input to the ADC from the hand controller is in the range of 0 – 5V. There is a voltage regulator circuit to power the micro. The is also a power/transmission circuit that combines the power for the slot car and the transmission signal to transmit onto the rails of the track.

5.0 SOFTWARE IMPLEMENTATION

5.1 Control Unit implementation

The most important part of the software design was the control unit. This is responsible for reading a data value from the hand controllers and sending it to the power circuit. As stated previously, the hand controller contains a variable resistor. Therefore, by moving the trigger on the controller we vary it's output voltage. We sample this voltage using an ADC and this is the data that we send to the car. We decided to go with an 8-bit micro that had at least four on chip ADC's. We also decided that we wanted in-circuit programming and UART to support extra features later on (i.e. PC control of vehicles, lap-counter/timer software). While there were many micro's available we chose the Atmel AVR AT90S8535. The main reason for this choice was that there are development boards available through the Uni and there is good support and resources for them.

Brand	Model	Freq.	Prog Mem	RAM	EEPROM	Ports	Timers	Serial I/O	ADC	Packages	Cost
Atmel	AT90S8535	8	8K	512	512	32	2x8/1x16/WD	UART/SPI	8x10-bit	PDIP40/TQFP44	50.00
Microchip	PIC16F870	20	2K	128	64	22	2x8/1x16/WD	UART/SPI/I2C	5x10-bit	DIP28/SSOP28	15.00
Microchip	PIC16F871	20	2K	128	64	33	2x8/1x16/WD	UART/SPI/I2C	8x10-bit	PLCC44/DIP40	20.00
Microchip	PIC16F872	20	2K	128	64	22	2x8/1x16/WD	UART/SPI/I2C	5x10-bit	SOIC28/DIP28	15.00
Microchip	PIC16F873	4	4K	192	128	22	2x8/1x16/WD	UART/SPI/I2C	5x10-bit	SOIC28/DIP28	24.00
Microchip	PIC16F874	4	4K	192	128	33	2x8/1x16/WD	UART/SPI/I2C	8x10-bit	DIP40/TQFP44	30.00
Microchip	PIC16F876	4	8K	368	256	22	2x8/1x16/WD	UART/SPI/I2C	5x10-bit	SOIC28/DIP28	26.00
Microchip	PIC16F877	4	8K	368	256	33	2x8/1x16/WD	UART/SPI/I2C	8x10-bit	PDIP40/TQFP44	36.00
ST	ST72314J2	16	8K	384		32	2x16/WD	SPI/UART	6x8-Bit	SDIP42/TQFP44	25.00
ST	ST72331G1	16	4K	256		22	2x16/WD	SPI/UART/I2C	6x10-Bit	SDIP32/SO28	20.00
ST	ST72272K2	16	8K	384		24	2x16/WD	SPI/UART	8x10-Bit	SDIP32/TQFP32	20.00

Table 1. Selection of microcontrollers for Control unit circuit. First entry is the selected option. Sixth entry is the preferred option.

Control unit that can sample the energy from the hand controller and send the data to the car via the track. The energy from the hand controller is a varying voltage as the hand controller contains a variable resistor element. An 8-bit micro with on chip ADC can sample this data very quickly and package it up in a data packet to send to the track. There are many different micro's available that can do this, but we chose the Atmel AVR AT90S8535. We chose this chip mainly because the Uni has development boards available and because of the support available for this chip. For a commercial product though cost is important. Taking this into consideration the better choice would have been the Microchip PIC16F870 as this chip costs around a third of the price of the Atmel. It also has all the necessary features such as in-circuit programming, UART, five 10-bit ADC channels, three timers (two 8-bit timers and one 16-bit) and a 20MHz operating frequency. Hence, it is similar in features to the Atmel but faster and cheaper but has less memory. The only problems that might arise with this choice is the program memory size of 2Kb or number of ports (three 8-bit ports). If this is the case, we can step up to the PIC16F871 for more ports (five 8-bit ports), the PIC16F873 for more memory (4Kb) or the PIC16F874 that has both. These chips cost around $5, $10 and $15 more respectively, so even with the PIC16F874 that costs around $30, we would still be ahead by around $20. Learning the assembler for either micro's will not be a problem, we have chosen to use C to program the micro as it has large support and saves having to learn a new language for each micro.

The first flow version of code written for the control unit had a very solid structure. It started with the initialisation of the hardware and then was based around two loops. The loop controlled the transmission of data to the track, while the second loop updated the data in the transmission signal while using the first loop for timing (such as when to read the ADC value etc.). This code ended up very inflexible for changes in ADC conversion rates or data transmission rates so we decided to revise it to make it flexible. We decided to use interrupt routines to do the bulk of the work and just do the initialisation of hardware in the main routine. We ended up with two ISR's. The first one triggers one ADC conversion completion to save the value to the data signal and reset the ADC again, while the second ISR is triggered by a timer overflow, which is responsible for transmitting the next bit in the data signal to the power circuit. This code ended up being a lot more efficient then the original version and surprisingly simple as well. It also allows for simple editing in case of changes such as transmission rate or even method and also the addition of other features.

5.2 Car Software Implementation

The task of the micro on the car was to read in the data that was encoded onto the track and send a PWM signal to the motor driver circuit that relates to the hand controller input. We decided once again to use interrupts is the software to read in the data from the track and send it to the PWM register. As there is very little room underneath the body of the slot cars, we needed to have a very small circuit for the car. This meant the best choice in micro would be a surface mount package like a TQFP package, which is just over 1cm square in size, or a SSOP which is thinner but just as long (but contains less pins). As we were using the Atmel for the control unit, we decided it would be best to use a similar chip for the car circuit. We decided on the AVR AT90S8515 chip, which is very similar to the `8535 but doesn't have the ADC and has one less time. It is also a cheaper micro and the uni has development boards for these as well. As there were none of these chips available in the package we required, we decided to go with the `8535 as it was available. The best choice of microchip for the car circuit would probable be the PIC16F627 with 1Kb program memory and PWM or the PIC16F628, which has 2Kb memory, PWM and is available with a 20MHz operating frequency instead of 4MHz.

Brand	Model	Freq.	Prog Mem	RAM	EEPROM	Ports	Timers	Serial I/O	PWM	Packages	Cost
Atmel	AT90S8515	8	8K	512	512	32	1x8/1x16/WD	UART/SPI	3	PDIP40/TQFP44/MQFP44	40.00
Atmel	AT90S8535	8	8K	512	512	32	2x8/1x16/WD	UART/SPI	3	PDIP40/TQFP44/MQFP44	50.00
Microchip	PIC16F83	4	512	68		13	1x8/WD			DIP18	12.00
Microchip	PIC16F84	20	1K	68		13	1x8/WD			DIP18/SOIC18	15.00
Microchip	PIC16F627	4	1K	224	128	15	2x8/1x16/WD	UART/SPI	1	DIP18/SOIC18	11.00
Microchip	PIC16F628	4	2K	224	128	15	2x8/1x16/WD	UART/SPI	1	DIP18/SOIC18	13.00
Microchip	PIC16F628	20	2K	224	128	15	2x8/1x16/WD	UART/SPI	1	DIP18/SOIC18	15.00
Microchip	PIC16F870	20	2K	128	64	22	2x8/1x16/WD	UART/SPI/I2C	1	SOIC28/DIL28/SSOP28	15.00
Microchip	PIC16F871	20	2K	128	64	33	2x8/1x16/WD	UART/SPI/I2C	1	PLCC44/DIL40	20.00
Microchip	PIC16F872	20	2K	128	64	22	2x8/1x16/WD	UART/SPI/I2C	1	SOIC28/DIL28	15.00
Microchip	PIC16F873	4	4K	192	128	22	2x8/1x16/WD	UART/SPI/I2C	2	SOIC28/DIL28	24.00
Microchip	PIC16F874	4	4K	192	128	33	2x8/1x16/WD	UART/SPI/I2C	2	DIL40/MQFP44/TQFP44	30.00
Microchip	PIC16F876	4	8K	368	256	22	2x8/1x16/WD	UART/SPI/I2C	2	SOIC28/DIL28	26.00
Microchip	PIC16F877	4	8K	368	256	33	2x8/1x16/WD	UART/SPI/I2C	2	DIL40/MQFP44/TQFP44	36.00
ST	ST72T272		4K	256		22	2x16/WD	SPI/UART/I2C	1	SDIP32/SO28	20.00

Table 2. Selection of microcontrollers for Car circuit. Second entry is the selected option. Sixth entry is the preferred option.

The PWM signal is very simple to generate. After initialising the PWM generator on the chip all that needs to be done is send the data value to the PWM input register and the chip continually transmits a PWM signal. To receive the signal on the other hand is a bit more complicated. We trigger an interrupt on an external event (the value of the input signal changing), and capture the voltage level of the signal (high or low). We check how many bits the signal was in its previous state and use this to re-generate our signal. We check the signal to make sure we recognise the sync byte at the start and then send the data that corresponds to the car to the PWM register. The data value is determined by the ID number given to each car (i.e. a car with ID #1 would read the first byte after the sync byte as it's data value).

6.0 PRODUCT EVALUATION

The biggest step in the design of the Digital Slot Car System was to convert the system over from using an analogue signal to a digital signal. After researching different slot car systems and also some digital train sets, we tried to pull together the design concepts we thought would be useful for developing our system. We started by looking at the different communication methods that might suit our design. One method that we were looking at was to encode an AM data signal onto the power rails on the track. But after looking at the noise developed on the track while the cars were running and also because of the complex analogue circuit we would have to build we decided to use a digital signal to transmit the data. We came up with our solution for the digital signal encoded on the top of the power signal which we found had a few good design points. Firstly digital signals are more tolerant to noise which is a large problem in our transmission medium. Also the signal was easy to see on an oscilloscope in testing. We also found the signal easy to separate from the power on the rails which made it easy to break up the power and transmission signals in the car circuit.

We were able to break the our design into small parts and test these parts. Most sections worked closely to how we had design them to. Although we did find some interesting results with the PWM signal to the motor. We were able to test this by developing a test circuit that allowed us to combine the input of control unit circuit (the ADC input) with the PWM output from the car circuit. We then sent this signal directly to the track to see the effect of PWM on the motor of the slot car (We also tested this using the motor driver circuit being driven from a function generator). We found that with high frequency PWM signals the motor would emit a high pitch noise, but with low frequencies the motor had poor low voltage response. We decided to go with a frequency that has some noise from the motor but most of the noise is drowned out by the noise of the car driving around the track.

Were the project wasn't successful was in the car receiving the signal from the track. We were able to detect the expected signal at the pin on the chip but the chip wouldn't detect signal. We think the problem may be due to the chip being damaged but we have not had time to test this as the problem was found on demo day. With this problem we had to take a backward step for our final demonstrated product. We took the system but to the most successfully functioning stage which was to use the control unit to output a PWM signal instead of the transmission signal. We then used the motor driver circuit from the car circuit to drive a signal on the track that that could power the car. So basically we had just replaced the analogue signal from the standard system with a digital PWM signal.

The project also falls a little short on the feature we had planed to have implemented. We had planned to have at least two cars running on one track with a place that they could change lanes. I think the main reason we didn't reach our goal was because a lack of adequate time planning. We started out at the beginning of the project with a basic time plan for the project, but we got caught up in the research stage and let the ourself spend too much time there. We then spent very little time between the end of first semester and the first weeks of second semester on the project when we should have been catching up on the work we hadn't done. Looking back on the way we spent our time, it is clear that we need to have a good time plan and try to stick to this as close as possible. Without proper time management, task get rushed that need to have time spent on them and other smaller tasks have too much time spent on them.

Another area that I could improve in as an Engineer would be to more thoroughly document the project. I had taken many notes during the project, but I should have used a workbook for all my notes and testing results. Without documenting enough information from the project you actually waste time in doing things more then once.

7.0 FUTURE DEVELOPMENTS

The main aim for the first version of this project was to implement the digital control for the slot car system. Most of the extra features of the digital system rely on this being done. Our system currently has a form of digital control to communicate between the control unit, that reads the data from the hand controllers, and the slot cars. The first thing to look at if continuing this project would be to do a little bit more work on the slot car to enable the receiving of transmission data to function properly. The problem is possibly from a damaged chip, but as we found the problem on demo day, we were unable to fix it. Once this problem is fixed, and with a couple of additions to the software to allow the full signal to be transmitted, the system will be able to run up to four cars independently on the track at the same time.

The next step from here would be to implement the lane changing feature so we can take full advantage of being able to run cars in the same lane and implement passing in our system. This is one of the features that we had worked on with a design almost ready to be implemented. Our design consisted of an extra button on the controller that could be read by the control unit. There would also need to be some sensors in the track to determine the closest lane switch is to the car that needs to be switched. To sense the cars we had decided to use an array of diodes underneath the cars. As each car has a unique ID number, the diodes would display a pattern that would be display the cars ID number. The sensor on the track would contain a series of optical sensors that would detect the ID number of a passing car and relay the data back to the control unit. The control unit could then signal a lane switch to switch when that car was approaching. The switching device could be one of two solutions that we have decided on. One solution would be a mechanical switch that would operate a small lever embedded into the track. This would force the guide pin on the slot car to change its path, hence forcing it to change to the other lane. The other solution is to use an electro-magnet to pull the guide pin onto a different path. The second solution is the preferred solution as there are no moving parts and therefore it should be less prone to break (after all it is a toy!).

Another feature to implement would be a lap counter. This lap counter could just be implemented with a series of TTL devices. We could use a counter for each car that was connected to a couple of seven segment displays to display the counter value. There would also have to be a reset button to reset the counter. If we placed a car position sensor at the start of the circuit, the control unit could detect when a car passed the start/finish line and signal the appropriate counter. Using this sensor at the start/finish line, we could also determine the cars lap time. This could be displayed on seven segment displays, an LCD module or sent to some software on a PC via the serial port.

By enabling the serial port on the control unit, we have access to many other features that could be implemented. We could use the PC to log data such as lap time information, a lap counter and keep track of a cars position (using position sensors in the track). With this sort of information being fed to the PC, we can go a couple of steps further and include PC control of the slot cars. We have the position of the cars on the track; so all the PC needs to know is what the track looks like. We could develop a program that we could build a virtual track in. This virtual track could then become the model for the PC to use as its track. Then by telling the PC were the position sensors were the computer could then determine how fast to drive the car for a particular part of the track. The position sensor data would then be used to update the PC's model on where the cars are. This system would take a lot of work to develop, as there would be a lot of fine tuning of the computers model for safe cornering speeds and cars response. But, if this feature was

developed, it would allow the slot car set to be fun for a single player as they could then race the computer. There could also be multiple people racing the computer, which could add a lot of entertainment to using a slot car set.

8.0 CONCLUSION

a) Provide a brief overview of the project – what did you set out to do? What did you achieve? How well did it perform?

At the beginning of the project we set out to create a slot car set that would support up to four cars racing at a time with the ability to overtake. This would make a the standard slot car set more entertaining to use and also be a marketable product. The final product though ended up being a system that works very similar to the original except with a digital PWM signal on the track instead of the analogue. So we have developed a ‘digital’ slot car system, but with out the extra features implemented there isn’t much advantage in this system, except to be used a base for developing a slot car system that will be more entertaining.

The system is similar in feel and response to the original system, so from what we have developed so far the system does preform to specification. With a little more time to debug our system we would be able to run multiple cars per track with only the lane changing feature to implement to really make this Digital slot car system more entertaining to use then the original system.

9.0 Bibliography

About Slot Cars, *Slot Car World*, http://www.slotcarworld.com/

Slot Car Garage, *Slot Car Garage*, http://www.slotcargarage.com/index.htm

A Brief History Of Slot Car Racing, *Slot Car Enthusiast,* http://www.slotcar.org/sce/BriefHistory.htm

History, *Dundee Slot Car Club*, http://www.phsc11985.pwp.blueyonder.co.uk/newpage1.htm

Software, *Slot Side*, http://www.slotside.com/links/cat8.html

Identify You Slot Car Motor, *Scratchbuilt*, http://www.getyourwebsitehere.com/scratchbuilt/articles/motorid00.html

Mini Racing: Histoire Du Slot Racing, *Gérard Caupène - Mini Racing,* http://www.miniracing.fr/histoire_eng.htm

FALLER AMS Slot Cars Introduction & History, *Steve Cook*, http://www.geocities.com/steve.cook/ams/ams-intro.html

Controller Tips & Information, *Professor Motor*, http://www.professormotor.com/controltech.shtml

Marklin Trains: Digital, *Marklin*, http://www.marklin.com/tech/digital1/

Appendix A - Hardware for control unit

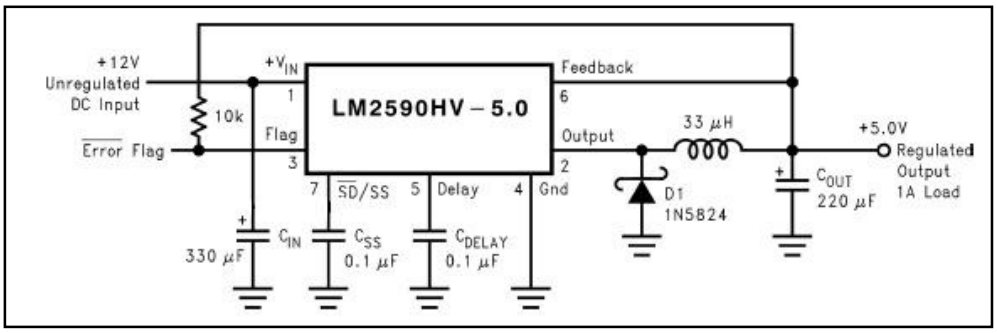

Figure 1. Buck converter for control unit circuit

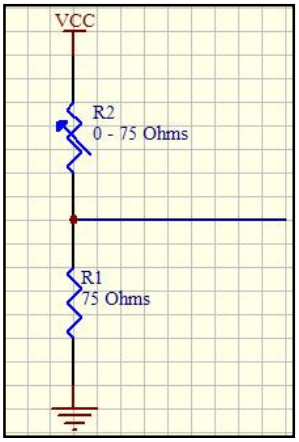

Figure 2. Hand controller resistor circuit

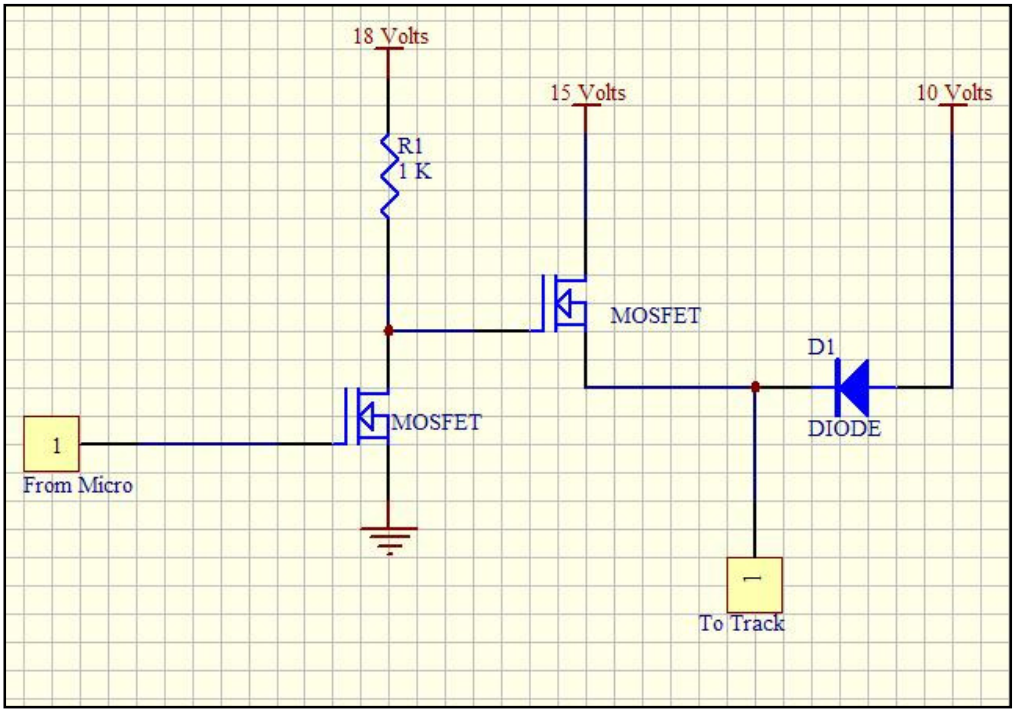

Figure 3. Line Driver circuit to encode transmission data from micro and power to send to rails on track.

Appendix B - Hardware for Car Circuit

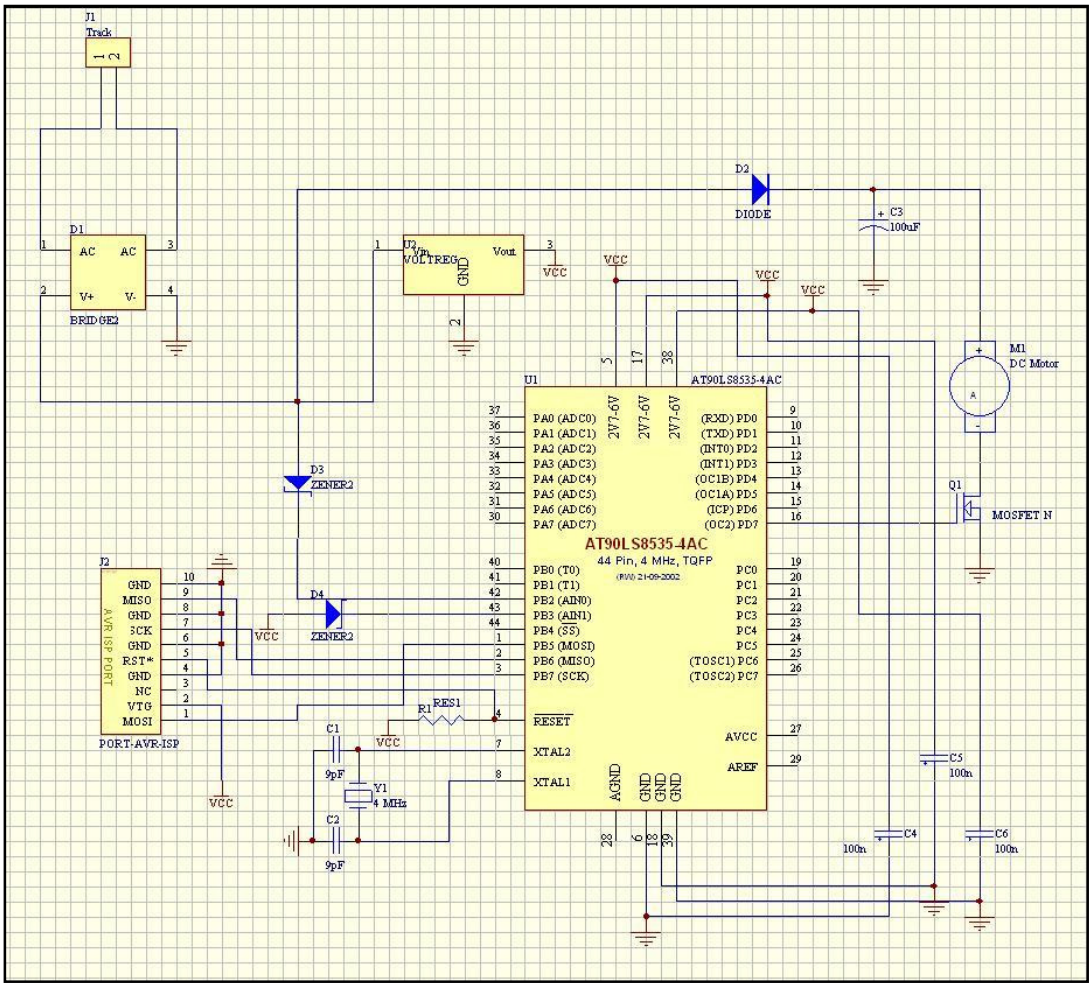

Figure 4. Car Circuit Schematic

Appendix C - Software Flow Charts

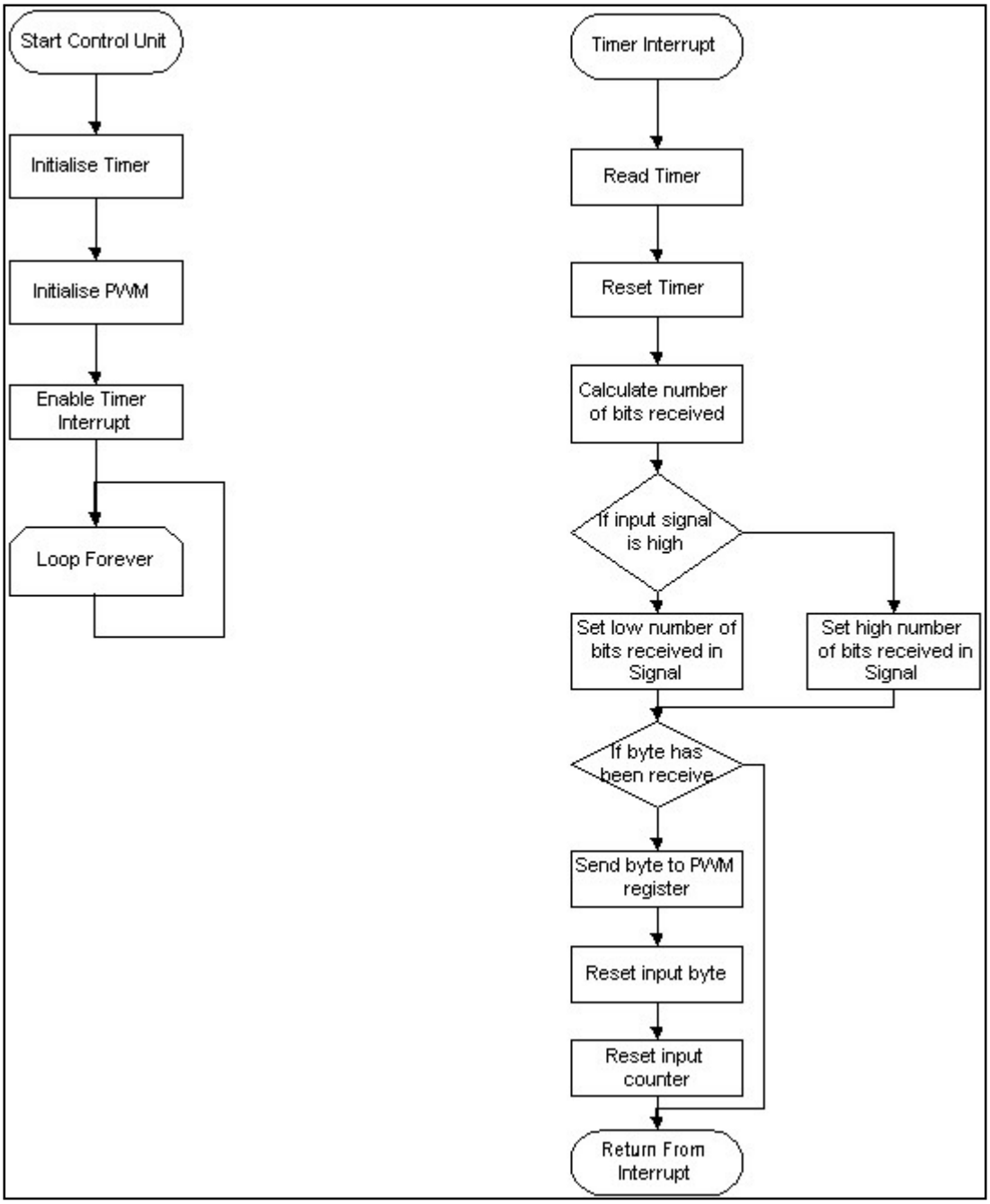

Figure 5. Flow Charts for the Car Software

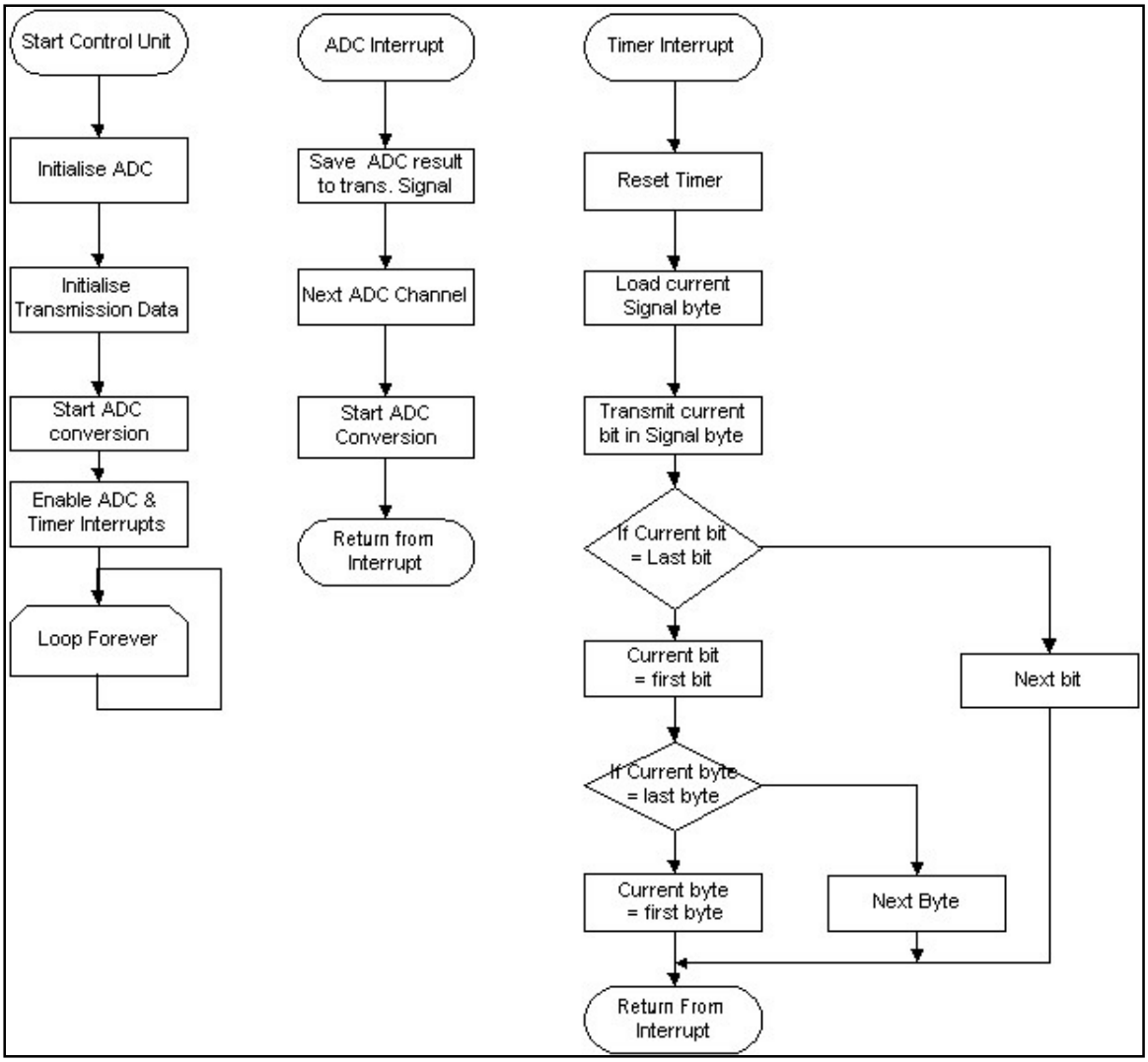

Figure 6. Flow Charts for the Control Unit Software

Appendix D - Software for Control Unit

```
/*********************************************
Project : Digital Slot Car System
Version : Controller Code
Date    : 21/10/2003
Author  : Scott Bremner
Comments: CodeWizardAVR V1.23.8c was used to generate the initial structure for
                        this code. The code was then changed so it would be compatable with
                        the AVR-GCC compiler.

Chip type           : AT90S8535
Clock frequency     : 4.000000 MHz
Memory model        : Small
Data Stack size     : 128
*********************************************/
#include <interrupt.h>
#include <sig-avr.h>
#include <io.h>
#include <io8535.h>

#define FIRST_ADC_INPUT 0
#define LAST_ADC_INPUT 3

static unsigned int adc_data[LAST_ADC_INPUT-FIRST_ADC_INPUT+1];
static unsigned char index=0;
static unsigned char signal[6];
static unsigned char data = 0;
static int tCount = 7; // timer counter
static int iCount = 0; // timer counter

/*
 * Timer 0 overflow interrupt service routine -
 * When interrupt triggers shift register to send next bit.
 * If byte sent load next byte to transfer.
 */
SIGNAL(SIG_OVERFLOW0)
{
        // Reinitialize Timer 0 value
        outp(0x10, TCNT0);

        outp(data, PORTD);
        data = data >> 1;
        tCount--;
        if (tCount < 0)
        {
                tCount = 7;
                data = signal[iCount];
                (iCount == 0) ? (iCount = 1) : (iCount = 0);
        }
}

/*
 * ADC interrupt service routine -
 * Scans first four input pins for the ADC and does a conversion
 * on each pin.
 */
SIGNAL(SIG_ADC)
{
        // Read the AD conversion result
        adc_data[index] = (inp(ADCL) + (inp(ADCH) << 8));

        // Select next ADC input
        if (index == LAST_ADC_INPUT)
                index = FIRST_ADC_INPUT;
        else
                index = LAST_ADC_INPUT;

        outp(FIRST_ADC_INPUT+index, ADMUX);
        signal[1] = (adc_data[FIRST_ADC_INPUT] >> 2);
```

```
        // Start the AD conversion
        //ADCSR|=0x40;
        sbi(ADCSR, 6);
}

int main(void)
{
        // Input/Output Ports initialization
        // Port A initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTA);
        outp(0x00, DDRA);

        // Port B initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTB);
        outp(0x00, DDRB);

        // Port C initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTC);
        outp(0xFF, DDRC);

        // Port D initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=Out
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=0
        outp(0x00, PORTD);
        outp(0xFF, DDRD);

        // Timer/Counter 0 initialization
        // Clock source: System Clock
        // Clock value: 500 kHz
        outp(0x02, TCCR0);
        outp(0x00, TCNT0);

        // Timer/Counter 1 initialization
        // Clock source: System Clock
        // Clock value: Timer 1 Stopped
        // Mode: Normal top=FFFFh
        // OC1A output: Discon.
        // OC1B output: Discon.
        // Noise Canceler: Off
        // Input Capture on Falling Edge
        outp(0x00, TCCR1A);
        outp(0x00, TCCR1B);
        outp(0x00, TCNT1H);
        outp(0x00, TCNT1L);
        outp(0x00, OCR1AH);
        outp(0x00, OCR1AL);
        outp(0x00, OCR1BH);
        outp(0x00, OCR1BL);

        // Timer/Counter 2 initialization
        // Clock source: System Clock
        // Clock value: 125.000 kHz
        // Mode: Phase correct PWM top=FFh
        // OC2 output: Non-Inverted PWM
        outp(0x00, ASSR);
        outp(0x63, TCCR2);
        outp(0x00, TCNT2);
        outp(0x00, OCR2);

        // External Interrupt(s) initialization
        // INT0: Off
        // INT1: Off
```

```
            outp(0x00, GIMSK);
            outp(0x00, MCUCR);

            // Timer(s)/Counter(s) Interrupt(s) initialization
            outp(0x01, TIMSK);

            // Analog Comparator initialization
            // Analog Comparator: Off
            // Analog Comparator Input Capture by Timer/Counter 1: Off
            // Analog Comparator Output: Off
            outp(0x80, ACSR);

            // ADC initialization
            // ADC Clock frequency: 125.000 kHz
            outp(FIRST_ADC_INPUT, ADMUX);
            outp(0xCD, ADCSR);

            // Global enable interrupts
            sei();
            signal[0] = 0x01;

            while (1)
            {
    // Loop forever

            }
   return 0;
}
```

Code for Slot Car Controller Unit

Appendix E - Software for Control Unit

```
/*********************************************
Project : Digital Slot Car System
Version : Car Code
Date    : 21/10/2003
Author  : Scott Bremner
Comments: CodeWizardAVR V1.23.8c was used to generate the initial structure for
                        this code. The code was then changed so it would be compatable with
                        the AVR-GCC compiler.

Chip type           : AT90S8535
Clock frequency     : 4.000000 MHz
Memory model        : Small
Data Stack size     : 128
*********************************************/
#include <interrupt.h>
#include <sig-avr.h>
#include <io.h>
#include <io8535.h>

/* Length is 1 less then for controller as the controller sends a blank byte
for syncronisation */
#define TRANS_LENGTH      5
#define TRANS_TIME                  4

//unsigned char signal[TRANS_LENGTH];
unsigned char data = 0;
unsigned char mask = 0x80;
int start = 0;
int lastTime = 0;
int time = 0;

SIGNAL(SIG_COMPARATOR)
{
        int bits;

        //outp(inp(ACSR), PORTA);

        time = inp(TCNT0);
        lastTime = time - lastTime;
        bits = lastTime / TRANS_TIME;
        if ((inp(ACSR) & 0x20) == 0x20)
        {
                for (; bits > 0; bits--)
                {
                        data = data | mask;
                        mask = mask >> 1;
                        if (mask == 0)
                        {
                                outp(data, OCR2);
                                mask = 0x80;
                        }
                }
        }
        else
        {
                for (; bits > 0; bits--)
                {
                        data = data & ~mask;
                        mask = mask >> 1;
                        if (mask == 0)
                        {
                                outp(data, OCR2);
                                //PORTC = data;
                                mask = 0x80;
                        }
                }
        }
```

```
        lastTime = time;
}


int main(void)
{
        // Input/Output Ports initialization
        // Port A initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTA);
        outp(0xFF, DDRA);

        // Port B initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTB);
        outp(0x80, DDRB);

        // Port C initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        outp(0x00, PORTC);
        outp(0xFF, DDRC);

        // Port D initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=Out
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=0
        outp(0x00, PORTD);
        outp(0x80, DDRD);

        // Timer/Counter 0 initialization
        // Clock source: System Clock
        // Clock value: 500 kHz
        outp(0x00, TCCR0);
        outp(0x00, TCNT0);

        // Timer/Counter 1 initialization
        // Clock not used
        outp(0x00, TCCR1A);
        outp(0x00, TCCR1B);
        outp(0x00, TCNT1H);
        outp(0x00, TCNT1L);
        outp(0x00, OCR1AH);
        outp(0x00, OCR1AL);
        outp(0x00, OCR1BH);
        outp(0x00, OCR1BL);

        // Timer/Counter 2 initialization
        // Clock source: System Clock
        // Clock value: 125.000 kHz
        // Mode: Phase correct PWM top=FFh
        // OC2 output: Non-Inverted PWM
        outp(0x00, ASSR);
        outp(0x63, TCCR2);
        outp(0x00, TCNT2);
        outp(0x00, OCR2);

        // External Interrupt(s) initialization
        // INT0: Off
        // INT1: Off
        outp(0x00, GIMSK);
        outp(0x00, MCUCR);

        // Timer(s)/Counter(s) Interrupt(s) initialization
        outp(0x00, TIMSK);

        // Analog Comparator initialization
        // Analog Comparator: On


        // Interrupt on Output Toggle

        // Analog Comparator Input Capture by Timer/Counter 1: On
        // Analog Comparator Output: Off
```

```
        outp(0x0C, ACSR);

        // Global enable interrupts
        sei();

        while (1)
        {
        // Loop forever
        }

        return 0;
}
```