# EE 382 – Junior Design

## Final Report

## Tail

May 7, 2001

Submitted to:

**New Mexico Tech**
**EE Department**

Dr. Stephen Bruder
Dr. Kevin Wedeward

Submitted By:

**Group 7**

Mike Berg
Seth Schuyler
Bill Willems
Thomee Wright

# Table of Contents

# Abstract

## *Overview*

In the beginning, there was the maze. The floors were of black and the walls of white and within the maze dwelt the demon known as Fire. The many peoples of the NMT EE Department cried out for a savior to deliver them from this demon. The governing body of the peoples pleaded to Group 7, “Send unto us an autonomous robot so that we might rid ourselves of this demon called Fire. Please find us worthy of optically isolation, closed loop speed control, and wall following. We stand before you in all humility to ask that it follow the laws of our land known as the Official Contest Rules for the 2001 Trinity College Fire Fighting Home Robot Competition.”

And Group 7 had mercy upon them and decreed, “I shall deliver unto thee four of My angles. They shall build your robot, and it will be known unto you as Tail. It will be all that you ask of it and more. Moreover, Tail will rid thee of the mischievous sprite known to you as Fire, who, being naughty in my sight, shall snuffeth.” And there was much rejoicing, “Yea!”

## *Book of Mike*

Group 7, in all His divine wisdom, made Mike Angel of Sensors and Low Level Software. For Tail was to be indwelled by an HC12, which doth speak in a strange and cryptic language that the other three angels were unfamiliar with since they knew only the HC11. So Mike set about on a communication layer with the HC12 to simplify the task before Thomee, the Angel of Software. And after many ANDs, ORs, NOT's, and shifting of bits in macros, it was tested, and

Group 7 saw that it was good.

And Group 7 said unto Mike, "Our robot cannot see, deliver unto Me sensors that Tail may see the walls to navigate and see where the demon Fire doth lurk to and fro within the maze." Now the HC12 has eight eyes with which to see analog data, and numerous digital eyes. Two of the analog eyes are blindfolded by DBug12 and are not meant to gaze upon this world. Group 7 had also decreed to use five GP2D12 sensors as eyes to see the maze, leaving but one analog eye to see where Fire doth dwell. And upon that issue, Mike spent much time thinking.

But the GP2D12 eyes did not see linearly, which would have caused much confusion in the code. So Mike created a table to reside in the HC12 that would translate what the GP2D12 eyes doth see into linear results so that all may know where the walls of the maze doth lay.

And Group 7 returned to Mike saying, "Tail doth see the walls and doth navigate well, but doth not see where Fire lurks." At this Mike took an analog MUX, variable resistors, and PN168 photo-transistors, and (with Seth, the Angle of Hardware) fashioned six eyes onto one board, so that using only one analog eye, Fire canst hide from Tail no longer. And Group 7 did look upon the fire sensor board and saw that it was good.

And Mike battled long hours along with Thomee, Bill, and Seth against the Demon Murphy who did torment Tail both day and night till Murphy was finally banished from the land. Mike had great faith in Group 7, so he was permitted to look upon "Tail," the robot made in Group 7's image.

## *Book of Seth*

Group 7, in all His divine wisdom, made Seth Angel of Hardware. No physical medium could withstand actual contact with Group 7, so whenever a circuit was made to take form, it was by the hand of Seth. Group 7 said onto Seth, "Build unto Me an H-bridge board. It will be optically isolated. It will handle 14V and spikes of 6A. It will handle a high frequency PWM." Seth set about his task with the only tools at his disposal, Protel, an iron, and some nasty blue stuff. Despite this resistance, he would not waiver from his duty. When he was done, Seth had shown his true devotion to Group 7 by offering Him a fully functional PCB on single sided copper.

Many more demands would be made to prove Seth's devotion to his Group 7. For a while, he was cast into despair having been told to build a board for which the specs kept changing. Seth never felt that Group7 had forsaken him, and realized that this was to humble Seth before Group 7, so he would know that only Group 7 is perfect. Henceforth, He handed down circuits of great complexity, but Seth was always able to produce them expediently on single sided copper. Along with these tasks for which Seth had been intended, he also fabricated many chassis components.

Seth traveled with Thomee, Angel of Software, to deliver the message of Group 7's greatness to faraway lands. While there, Seth conspired with Thomee to give Tail the gift of return home mode. This was not to pass on that journey. Tail was made to wonder the maze, led neither by a column of smoke nor a column of fire. Thomee had sympathy for tail, and wished to end it's suffering. Seth wished to behold Tail's mighty perseverance. Thomee agreed, and soon

thereafter came the fruition of Group 7's decree. Seth had faith in Group 7, so he was allowed to look upon "Tail," the robot made in Group 7's image.

## *Book of Bill*

Group 7, in all His divine wisdom, made Bill Angel of Motors. He said unto Bill, “Our robot cannot move, deliver unto Me two motors so that we may travel freely. They will be sleek and powerful. They will be powered with 14.4V or less, have a high gearhead ratio, good encoder resolution, and enough torque to climb the walls.”

Bill set about his task by applying equations to determine the requirements of motors that would be worthy of Tail. After finding the one of his liking, Bill appealed unto Maxon, the maker of the motors and gearhead, and unto Agilent, the maker of the encoders saying “Give unto us two free samples and we shall make praise unto thee by applying your likeness to our miraculous invention known as Tail.” And Group 7 did look upon the free motor assemblies and saw that they were good.

Group 7 then called upon Bill saying, “Design for Me a decoder so that I may know how fast the motors are actually going and respond accordingly. The decoder should be written in Altera. It will have both position and speed counters and make use of the quadrature signal given unto thee from the motor encoders.” And Bill did create a robust decoder assembly and presented it unto Group 7 and He saw that it was good.

Group 7 again called unto Bill saying, “Build unto Me sensor boards that I may see white lines and circles and so that I may have sound activation.” And

Bill did design a white line sensor board and sound activation with tone decoding and gave the schematics unto Seth, the Angel of Hardware, so that they may have form. Along with these tasks for which Bill had been intended, he also helped to produce many chassis components and to aid Thomee, the Angel of Software, during his time of trial and tribulation. Bill had great faith in Group 7, so he was permitted to look upon "Tail," the robot made in Group 7's image.

## *Book of Thomee*

Group 7, in all His divine wisdom, made Thomee Angel of Software. He said unto Thomee, "Deliver unto Me code so that the robot may control the body my other angels have fashioned." And Thomee set about this task by wrestling with the tool known as the IDEA C Compiler. And then he rested.

Soon, Group 7 called Thomee to Him and said, "Your closed loop speed control and wall following algorithms are good, but Tail has yet to recognize the demon known as Fire. Deliver unto Me code that Tail may recognize Fire and vanquish him from the maze." Thomee completed this task in time to travel with Seth, the Angel of Hardware, to that far away land known as Trinity. Whilst there, Group 7 said unto Thomee and Seth, "Add unto Tail so that it may return unto the room from whence it came for a further 0.8 multiplier." And Thomee did not rest.

Upon returning to their homeland, Thomee did complete this task and presented it unto Group 7 and the peoples of NMT and they saw that it was good. Thomee had great faith in Group 7, so he was permitted to look upon "Tail," the robot made in Group 7's image.

# Introduction

## *Scope*

This document starts with an Abstract to give the reader a sense of what each team member contributed to the project. We then give a short description of the requirements of the robot and move on to the specific components of the robot. The last two sections cover a Cost Analysis and then Results and Conclusions.

## *Purpose*

The purpose of this document is to provide a complete technical description of “Tail”.

# The Requirements

This Robot Project is designed to meet the requirements of the Trinity College Fire-Fighting Home Robot Competition. There is also a local competition held by the New Mexico Tech Electrical Engineering Department. Entry into either competition is optional.

The Trinity College Fire Fighting Home Robot Contest is an annual event that is held at Trinity College in Hartford, Connecticut. This contest is known to be the largest, public, true Robotics competition held in the U.S. that is open to entrants of any age, ability or experience from anywhere in the world.

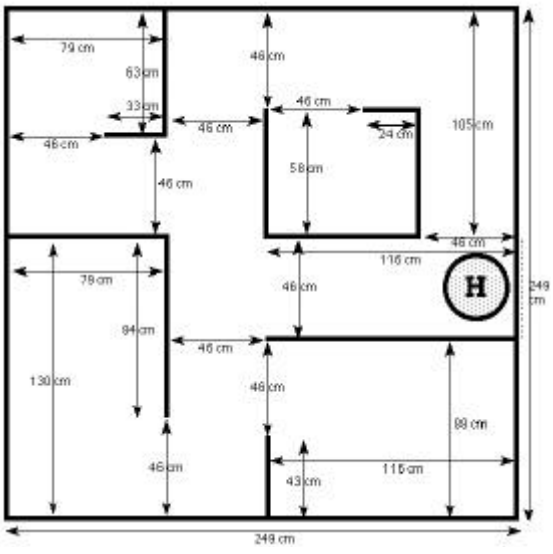


**Figure 1 - aMAZEing**

The goal of the Contest is “to build a Robot that can find and extinguish a fire in a house. The challenge for the entrants is to build a computerized (not radio-controlled) Robotic device that can move through a model of a single floor of a house [see Figure 1], detect fire (a lit candle) and then put it out. Robots that consistently accomplish this task in the shortest time win.”[1] Complete contest rules can be found on the Trinity Website.

There were some additional requirements put forth by the instructors (Dr. Stephen Bruder and Dr. Kevin Wedeward) such as optical isolation between high-power and low-power electronics, motor speed and/or position measurements through Altera PLD, closed-loop speed and wall-following control, main program must be in 'C', and a neat and clean final design with good connectors.[2]

# Robot Components

## *Chassis*

The entire chassis is manufactured from aluminum. Weight was not a big issue, but aluminum is easy to work with. The two plates are parallel, and connected by 4.5” standoffs. Thought the motors have a lot of torque, there was never any sign of flex. The torque was delivered to the ground through 2.25" solid rubber tires which were held onto aluminum axles by two o-rings each. The axles griped the motors with setscrews. This kept the motor assembly flush with the plates, preventing hang-ups on corners.

The distance sensors, motor control, and all low power boards are connected to the battery through a voltage regulated, expansion board. This board was created using polarizing, locking connectors to prevent accidental shorts. It also gathers the data and delivers them to the HC12 or receives instructions for the peripherals through three ribbon cables. The H-Bridge Board and fan control board are connected in parallel to the high power battery when the in-line fuse and switch are closed.

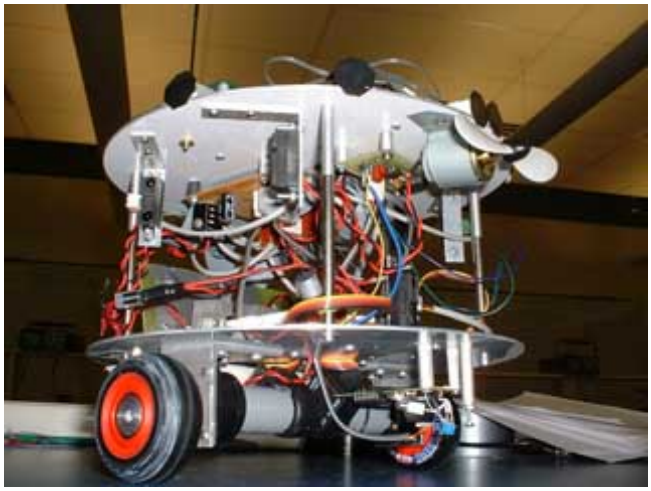

**Figure 2 - Tail**

## *Locomotion*

Our design uses a two-wheel differential drive. This consists of two 9V Maxon 6W Motors with 30:1 Gearhead and Agilent 500 count/rev quadrature encoders mounted on the horizontal centerline under the Lower Plate. At first, we were tinkering with the idea of mounting the motors slightly forward of center. This design does have its advantages (balance being one of them) but it also has its disadvantages (such as losing the ability to truly turn in place). After considering the pros and cons of each design, we decided to go with the centerline mounting points.

One of the design requirements was to have Closed Loop Speed Control (CLSC). Our CLSC Subsystem consists of four major components: the Maxon

Motor Assembly, the Signal Amplifier Assembly, the Altera Decoder Assembly and the HC12 Motor Control Subsystem as shown in Figure 3.

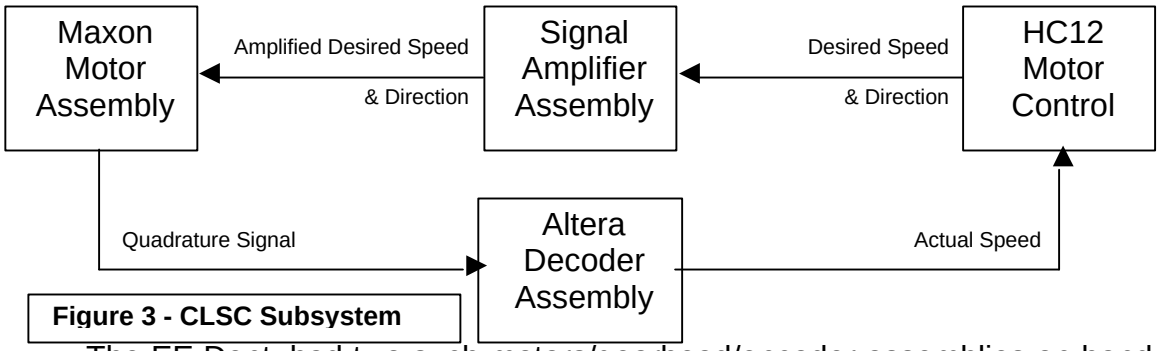


**Figure 3 - CLSC Subsystem**

The EE Dept. had two such motors/gearhead/encoder assemblies on hand. However (in an effort to alleviate costs), we were able to convince both Maxon Motors and Agilent Technologies to send us two complete motor assemblies and encoders. The Maxon Motor (A-max26-110947 9V 6W) and Gearhead assembly (GS 38-110454) is shipped from the factory as one unit and Dr. Bruder was kind enough to attach the Agilent Encoder assemblies for us. One of the conditions of our receipt of two free motors was to include the Maxon logo on our robot. Unfortunately, the logos they sent us were too huge for practical application to our robot (see Figure 4). However, we did print some smaller logos to display on our robot.


**Figure 4 - Have Some Logos**

The Signal Amplifier Assembly consists of two National LMD18200 H-Bridges and is optically isolated (as required) from the Low-Power System with Fairchild H11L1 Schmitt Trigger Optocouplers. These H-bridges were chosen

because, of the available options, they were the most powerful. They proved to be quite durable by never failing. Both the H-bridges and Optocouplers' spec sheets suggested an operating voltage of 15V was reasonable, but this was not the case. The Optocouplers were prone to fail after a couple of weeks at the unregulated 14.4V the battery was providing. The signal amplifier assembly was adjusted to include a Fairchild MC7805 5V voltage regulator on the Optocouplers' $V_{cc}$ line. The H-bridges were still able to understand this as logic high, so from then on the signal amplifier assembly was trouble free. The inverting property of the Optocouplers would make a power down of the HC12 look like a command to go full speed to the motors. This became a safety factor on more than one occasion, including an incident with orange juice. Each motor encoder outputs a quadrature signal (two signals 90° out of phase) the frequency of which directly corresponds to motor speed. There are several ways to calculate motor speed and direction given this signal – we considered three such solutions. The first is to implement a quadrature decoder in an Altera PLD. The second is to use frequency-to-voltage converters and feed the output voltage to an A/D port on the HC12. The third is to use the input capture pins on the HC12. The frequency-to-voltage converters as well as the input capture method would tie up already limited input pins as well as introduce a heavier processing load to the HC12. We already had an Altera PLD being used to control memory expansion and so it was decided to implement quadrature decoding in Altera. The Altera Decoder Assembly is illustrated below in Figure 5.

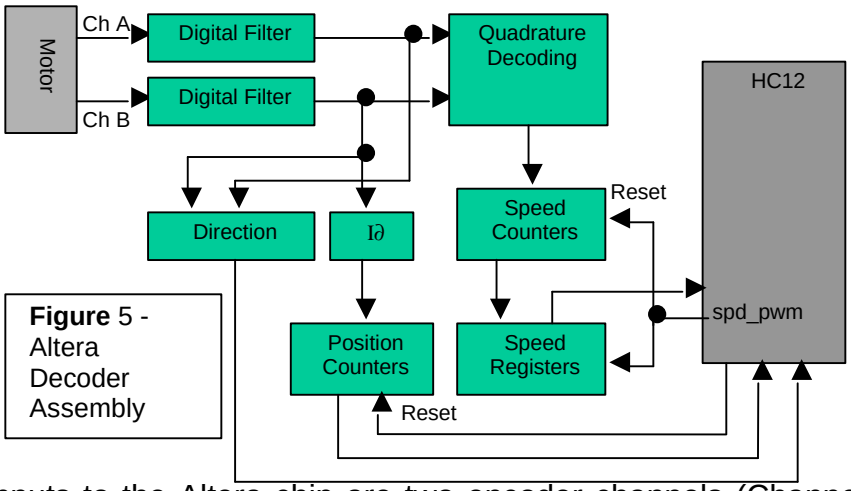


**Figure** 5 - Altera Decoder Assembly

The inputs to the Altera chip are two encoder channels (Channels A & B) from each motor and a low duty cycle PWM from the HC12. A Digital Filter is applied to both encoder channels with 6-state Moore Machines. The filtered signal is then used to determine direction – on the rising edge of Channel B, the motor direction corresponds to Channel A.

Channel B is also divided down and used to clock the Position Accumulators (if we don't divide it down, the 12bit counters would overflow about 14 times per revolution). The filtered signals are also fed into a 2-state Mealy Machine to decode the quadrature signal. The resulting clock signal is then used to clock the Speed Accumulators.

The low duty cycle PWM signal from the HC12 is used to latch the values from the Speed Accumulators into the Speed Registers and then to reset the accumulators. However, if this signal latches the registers and resets the accumulators at the same time, then we are latching the accumulator values into the registers while the accumulators are being reset. To get around this problem, we implemented a 4-state Moore machine to generate the accumulator reset signal one E clock cycle after the accumulator value is latched into the register.

Also, note that this figure describes the decoding process for only one motor. An identical assembly exists (in the same Altera chip) for the other motor (see Appendix for the underlying Altera Code).

## *Fire Extinguishing*

The fire extinguishing system is an optically isolated relay controlled fan. A single control line from the HC12 is connected to the low power side of the relay, with a 220 Ohm resistor in series to guarantee that at 5V, the HC12 will never exceed the 25mA that each port can supply. When 5V is supplied to the low power side of the relay, it closes the switch for the high power side, connecting the fan to the 12V power supply.

The fan is a 1.8A unit that can operate on 9-15VDC. The relay's high power side is rated at 2A, so can successfully handle the current needed by the fan. A very low resistor value was initially placed in series with the fan to prevent the possibility of excess current draw, but this prevented the fan from working, and was removed. The fan control circuit is in figure 6.

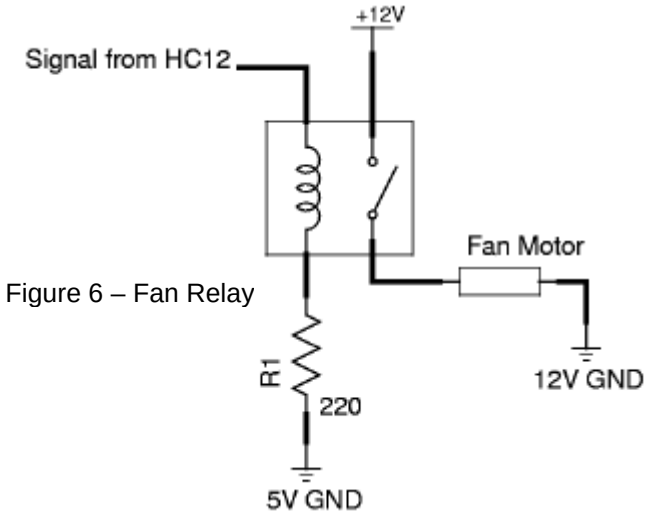


Figure 6 – Fan Relay

## *Microprocessor*

For our robot's control system, we were required to use a Motorola HC12. Mike had an HC12 from when he took EE308, and Bill purchased one to have spare for this class. Initially we used Mike's board, as it had a completed memory expansion, but we switched to Bill's board once he received a manufactured memory expansion board from the EE department. Bill purchased the memory expansion will the larger Altera chip to allow plenty of space for our intended motor interface. While running in Mike's Altera expansion, we were unable to have both speed and position monitoring. The HC12 proved to be a reasonable platform for developing the robot's control system, although it was not without flaws.

The HC12 has a built in PWM system with four separate PWM channels, which is ideal for running two motors. In addition, we used a third PWM output to drive the latch and accumulator reset in our Altera motor encoder interface. The PWM system on the HC12 proved to be both reliable and robust.

The HC12 has eight analog to digital converter ports, which are invaluable when interfacing to analog sensors. Unfortunately, the bootloader on the HC12 board prevents two of these channels from being used, leaving only six A/D ports for our use. We used five of these pins for inputs from our wall sensors, leaving only a single A/D port for our fire sensing system, leading us to the analog mux design.

Switching from the wire wrapped to manufactured memory expansion was supposed to allow us to run at full speed all the time, as the manufactured board included faster memory. After running the new board for a short time, we

discovered memory interface errors. We then found out that this particular revision of the HC12 makes some impossible timing demands on external memory when running at full speed. To correct this problem, we enabled the clock stretch that the EE308 class has been using for quite some time, which divides the clock in half when interfacing with external systems, including external RAM. This solved the memory read errors.

## *Power Distribution*

### High Power

The high power system (as illustrated in Figure 7) includes the motors, signal amplifier assembly, fan, and fan relay. The source is a 12V NiCd battery pack, and when it is fully charged, can deliver upwards of 14V and good surge currents. It could maintain this for around 1400mAhrs. This would easily destroy many components on the low power system, so the two are kept optically isolated. This is achieved with H11L1 Inverting Schmitt Trigger Optocouplers on the signal amplifier assembly and the internal isolation in the Antex Solid-State Opto-Isolated 2.5A DC relay which drives the fan. The motors draw an average of 10 A; the signal amplifier assembly, .66A; the fan, 1.8A; and the relay was negligible. This worked out fine because the high power battery held 1400mAhr, so we could run at full steam for over 7.5 minutes and still be able to put out the flame. This was never going to be the case in reality.

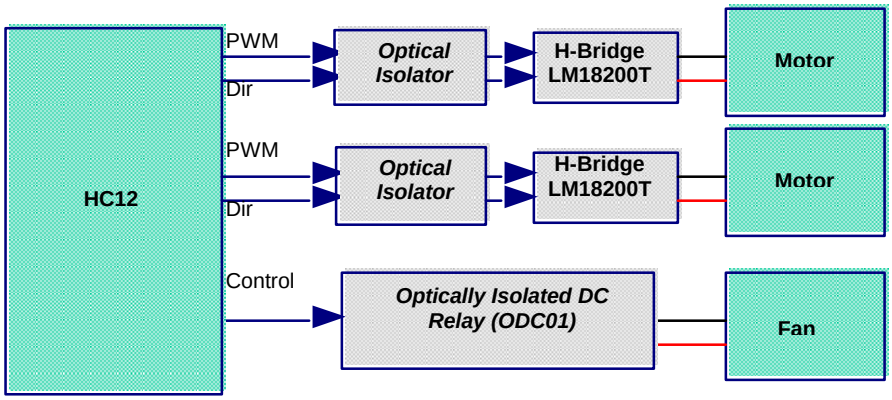


Figure 7 - High Power

## Low Power

The low power system includes the HC12, fire sensor assembly, white line sensor assembly, and distance sensors, and part of the signal amplification assembly (not shown). A Li-ion battery, Samsung BTL1310L, is the source, and supplies 7.2V. To step down the voltage to the 5V required by the HC12, a Fairchild MC7805 5V voltage regulator was put in line with the battery. This was able to hold the system at 4.99V until the battery dropped below this threshold. At that time, the battery dropped quickly to the mV range. The battery contacts could shift in an impact and cause a minor voltage change, which would cause memory read errors in the HC12. This was cured with a 1mF capacitor in parallel with the HC12, but the contacts were reinforced as well. The HC12 draws 100mA; the distance sensors together, 250mA; the signal amplification assembly, 200mA; the sound activation assembly, 15mA; the white line sensor assembly, 25mA; and the motor encoders together, 100mA. This totals to 690mA, so our 1350mAhr low power battery would last over 117 minutes.

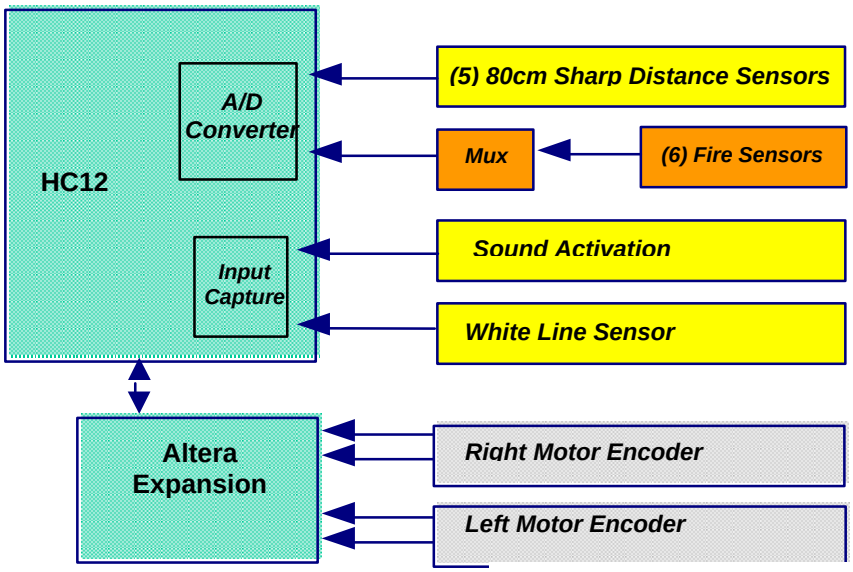


**Figure 8 - Low Power**

## *Sensors*

### Distance

We decided to use the Sharp GP2D120 or GP2D12 IR distance sensors on our robot. We chose these because of their insensitivity to ambient light level and color of the surface distance is being measured to. All necessary circuitry is also built into the units, so no external circuitry is needed. Power and Ground are supplied, and an analog voltage corresponding to the distance is produced on the signal line.

In our initial design we had selected the GP2D120 sensors which work over a 4cm-30cm range. Calibration measurements were made for the five sensors. The voltage to distance relationship for the GP2D120 can be modeled by:

$$dist = \frac{a}{(V + b)} + g$$

By using the measurements of table X.X to solve the resulting linear equations a = 9.4065, ß = -0.3615, and ? = -0.0133. Plugging these values into the program in gen_dist_table.c, a lookup table (dist_table.c) was generated which uses the result of the HC12's A/D converters as in index into the distance array to retrieve the distance detected by the sensor.

As different sensors configurations were tested and an angled configuration was finally chosen, the 30cm limit of the GP2D120 sensors was insufficient to see the walls of the maze when the robot was not parallel to the wall, resulting in random distance readings. So we switched to the GP2D12 distance sensors which are rated over the range from 10-80cm. The calibration measurements for the GP2D12 sensors are in table 3, and the same modeling equation was solved to return values of a = 19.5000, ß = -0.4900, and ? = 1.2600 (all after a small amount of manual tweaking).

This equation did not model the GP2D12 sensors as well as it did the GP2D120. The GP2D120 sensors deviated from the model only at the extreme values of below 5cm and above about 27cm. However, the GP2D12 sensors deviated by 1cm-3cm over most of the curve for distances from 10cm-50cm, beyond which the deviation increased as the measured distance approached 80cm.

## Fire

The fire sensors were PN168 phototransistors in a voltage divider configuration. We looked at the research done by one of the previous Jr. Design groups, showing that the media inside a 3.5” floppy disk filters most ambient light

while passing the peak output of the candle in the range of 750-800nm. In addition, the PN168's peak sensitivity is to 800nm wavelength light. Each sensor unit was made from a phototransistor was mounted inside a 3/8” x 3/8” x ¾” flat black enclosure, with the floppy light filter over the 3/8” x 3/8” open end.

Testing in the lab, a 1MO resistor in the voltage divider configuration provided output values of 0.20V when no candle was present, and 4.70V when 4” away from the candle. The units were also fairly directional with a field of view about 30° (15° in all directions from straight ahead). This matched the peak directional sensitivity range of 0-20° from centered for the PN168. Since each fire sensor can see about 30°, six fire sensors were placed around the front half of the robot to scan the 180° in front of the robot for a fire.

Since all fire sensors would be operation on the one remaining unused A/D port on the HC12, a 4051 analog MUX was used to cycle through the six fire sensors. For flexibility, the resistor in each voltage divider configuration is a 1MO, 25-turn trim POT and a 10kO resistor. This allowed each fire sensor unit to be adjusted for inconsistencies in the PN168 phototransistor, as well as adjustments to be made for a wide range of ambient IR light levels. Tests were done outside in the shadows (high IR environment) and after adjusting the POT; a flame could still be detected at approximately one foot away. The 10kO resistor is included in each sensor to protect the PN168 from carrying too much current should the trim POT ever be turned to near 0 O. Three control lines from the HC12 are used to select the fire sensor to perform the A/D conversion on, and a real time interrupt every 2ms is used to control this sampling, result

storage, and sensor switching. The complete fire sensor assembly is illustrated in figure 9.

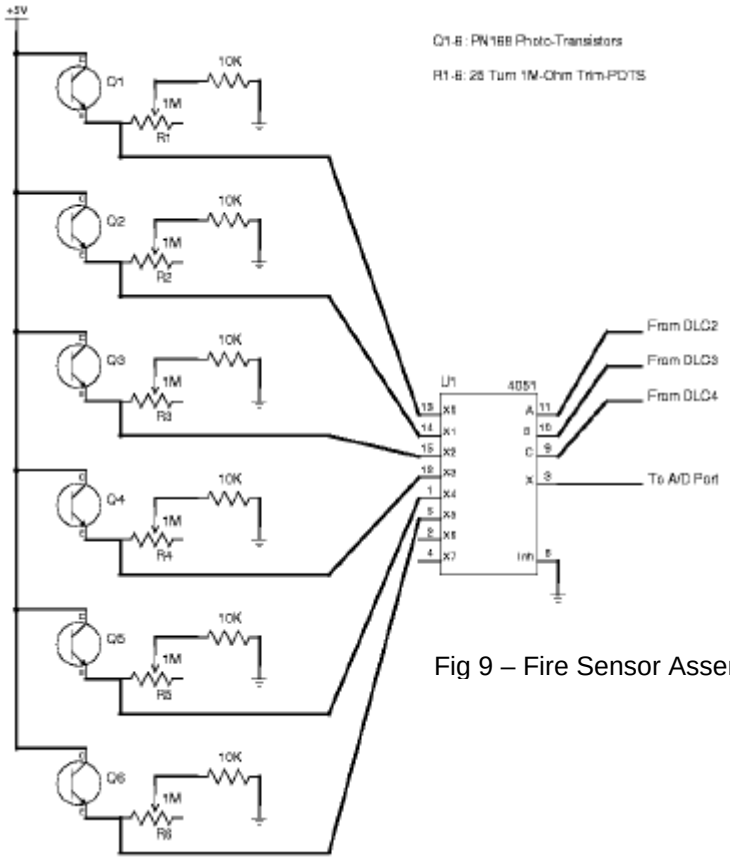


Fig 9 – Fire Sensor Assembly

## Sound Activation

One of the score multipliers is to have Sound Activation. The idea behind this being that our robot will be activated off the sound of a fire alarm (3-4kHz). To do this, we designed a circuit using the Phillips NE567 Tone Decoder/Phase-Locked Loop. The data sheet for the 567 gives the following design formulas:

$$f_o \cong \frac{1}{1.1R_1C_1} \text{ and } BW = 1070\sqrt{\frac{V_I}{f_oC_2}} \text{ in \% of } f_o$$

With $f_o$=3.33kHz and $V_I$=283mV (these values were obtained from an oscilloscope analysis of a 9V 3.4kHz buzzer) and BW=10. Use of these

equations and other information in the datasheet yielded $R_1$=27kO, $R_L$=1.2kO, $C_1$=0.01µF, $C_2$=0.68µF, $C_3$=0.022µF, and $C_4$=0.47µF as illustrated in Figure 10.

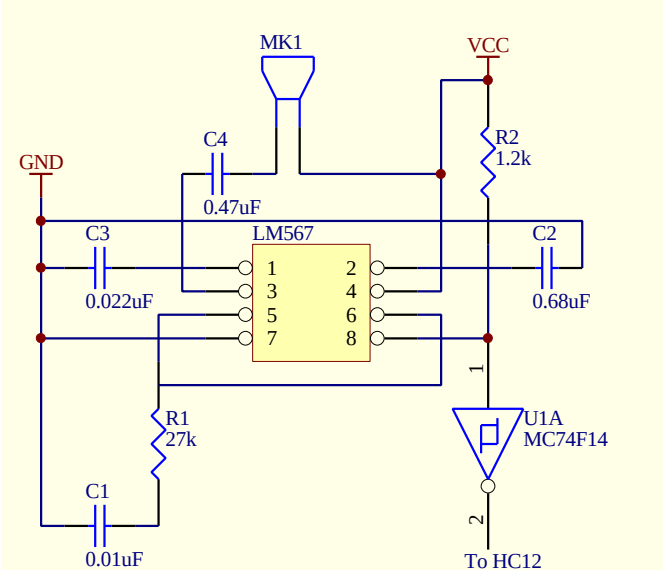


**Figure 10 - Sound Activation Schematic**

## White Line

The floor of the maze is black, the home and candle circles as well as the "door" lines are white. In order to see these we needed a White Line Sensor. The EE Department had several packaged Reflective IR Emitter/Detector Pair sensors (HOA708-1). These consisted of a matched LED and phototransistor pair. After playing around with several resistor values, we came up with the circuit illustrated in Figure 11.

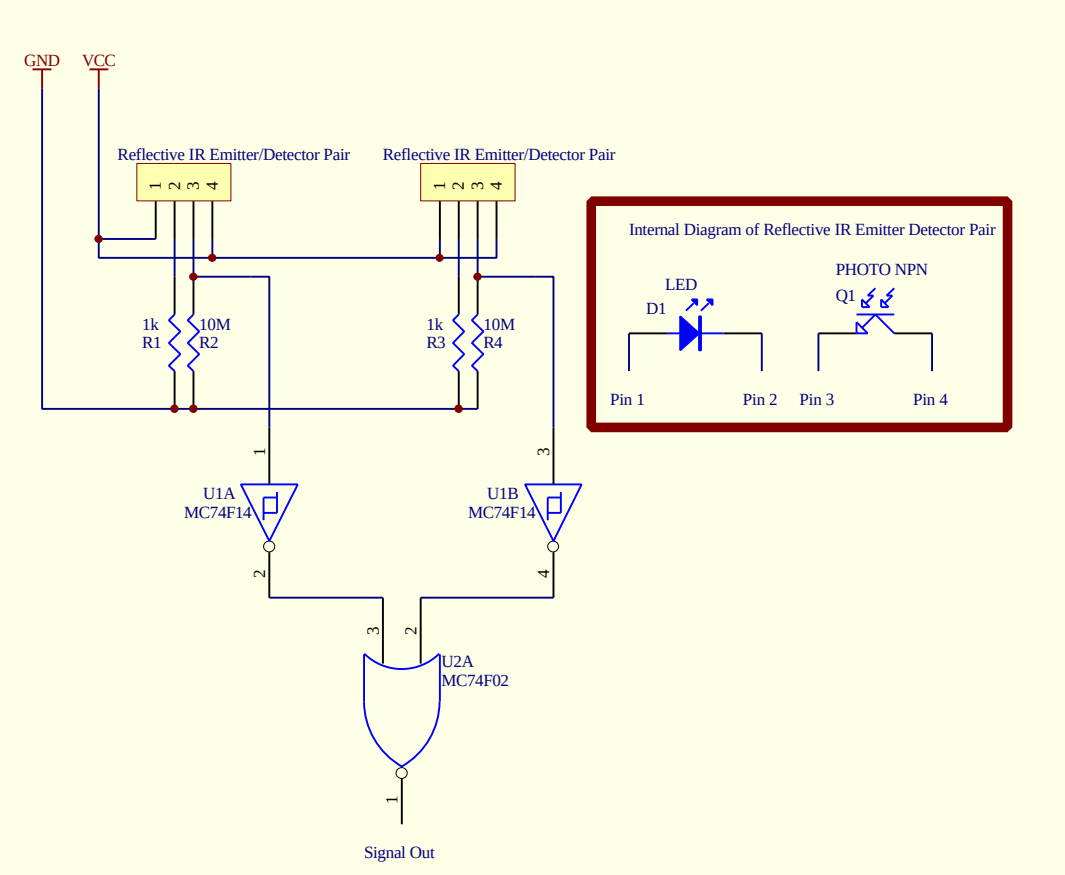


**Figure 11 - White Line Sensor Schematic**

With the lighting conditions in the Junior Design Lab, this configuration worked fine. However, once we got to Trinity and noticed that our robot was missing white lines, we discovered that our White Line Sensors only worked with a certain level of ambient light (the ambient light level at Trinity was significantly lower than that of our Junior Design Lab). To surmount this obstacle, we quickly wired a small incandescent bulb to our sensor board (see Figure 12) to increase the ambient light level and this solved our problem. Although this solution did work, it would not be efficiently mass-producible. That is why the final white line board does include a pot and bulb.

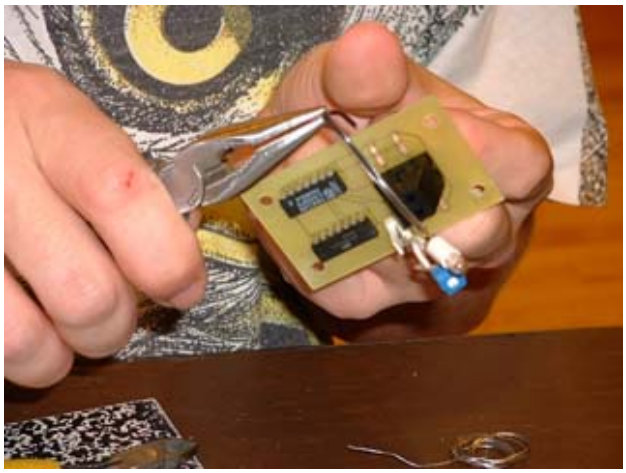

**Figure 12 - White Line Sensor Hack**

## *Software*

### Background

We found that the foundation for a good control system is closed loop speed control. Closed loop speed control provides accurate, reliable control of motor speed by monitoring the actual speed, comparing it to the desired speed, and adjusting power output accordingly. Without being able to rely on the robot behaving almost exactly as requested, the complexity of a useful navigation system increases significantly. In an algebraic control system, closed loop speed control can easily be represented by the equation $P_{new} = P_{old} + (S_{desired} - S_{actual})$ where P is power, and S is speed.

Navigation in an environment composed primarily of hallways is most easily implemented by following a wall on one side of the robot. Wall following, like closed loop speed control, can be simply represented in an algebraic control system: $L = S + (D_{desired} - D_{actual})$; $R = S + (D_{desired} - D_{actual})$ implements left wall following, where L and R are left and right motor speed, S is desired travel speed, and D is distance from the wall. These equations assume a drive system

composed of independent drive wheels on the left and right sides, which is the system used by our robot.

Designing the navigation system beyond wall following requires taking the layout of the operating environment into account. When operating in arbitrary starting point mode, the starting point cannot be directly taken into account in the navigation algorithm. It is known, however, that the robot will start in a room and that the fire will not be in that room. A basic search pattern can be implemented by simply leaving the start room and left wall following, scanning for fire when a white line is seen, and turning around if no fire is found. However, when starting in a room other than the island, the island will not be searched, and when starting in the island, only the island will be searched. This flaw can be remedied by turning around every time the home circle is encountered.

When a fire is found, there are two ways to approach it. The first is to use fire sensors to determine the direction of the flame, then travel directly towards it until the candle circle is detected. The second is to follow the wall of the room until the circle around the candle is detected. While traveling directly towards the flame is faster, there are some potential flaws to this tactic. First, the white walls of the maze reflect the candle well, and it is possible to "lock on" to a bright spot on the wall and travel towards it, rather than the candle. In addition, returning home requires leaving the room, and retracing the path to the candle from the doorway is much harder to do reliably if you don't wall follow to the candle. This is important because if the exit path does not closely retrace the entry path, it is possible to pass the candle circle again and mistake it for the white line marking

the room exit.

Finally, returning home when running in arbitrary starting point mode is not always a simple task. The task is made simpler because the search pattern visits each room sequentially in a clockwise order. Because of this, when the candle is found in the third room searched, home can be reached by simply continuing the search pattern one more room after leaving the room with the fire. Similarly, when the candle is found in the first room searched, home can be reached by reversing the search pattern after leaving the room with the fire. The remaining situation, finding the candle in the second room searched, is more difficult.

In order to return to a starting room that is 2 rooms behind (or ahead), it is necessary to determine which one of four possible return trips must be made. If the home circle has not yet been encountered, the start room must have been the lower right room, and the robot found the candle in the upper left room. In this case, home may be reached by left wall following after leaving the room where the fire was found. If the home circle has been encountered twice, the opposite trip must have been made, and home can be reached by right wall following.

If the home circle has only been encountered once, the robot must have either started or found the candle in the island room at the upper right. If the robot started in the island room, the home circle would have been encountered before any other rooms; on the other hand, if the robot started in the room at the lower left, the room at the upper left would have been reached before the home circle. Based on this, it is possible to determine which return trip should be made. Both return trips require traveling down the center hall without a consistent wall

on one side.

## Implementation

Initially, we designed a fully heuristic control system for the robot. While we were able to successfully implement closed loop speed control and wall following this way, we were unable to fully adjust cornering without destabilizing the system. After deciding that system was not a success, we attempted to design a fully algebraic control system. This time as well, we were able to implement both closed loop speed control and wall following. However, we again encountered the problem of not being able to properly adjust cornering without destabilizing the system. In both cases, inside corners, i.e. left turns when right wall following, were the problem. On our third attempt, we implemented closed loop speed control and wall following using an algebraic control system, but the wall following only handled outside corners. We were able to adjust this version very well. We then added a heuristic element to the control system, waiting until the front distance sensor dropped below a certain level, then rotating away from the wall for a fixed amount of time in order to execute a 90° turn. This solution, while not elegant, was able to fulfill basic navigation requirements very well, and therefore was used in the final version of the control system.

After implementing the left wall follow and turn around at the home circle searching algorithm, we found that it worked very well. As such, we kept it and didn't try anything else. One minor complication was properly locating a wall to follow out of the starting room. we tried simply rotating and watching for a relative minimum on my front distance sensor, assumed it was the closest point on the

wall we were facing, and drove towards it. This brought the robot nearly perpendicular to a wall, so we stopped once we got close enough, turned, and began following the wall. This worked well enough that it was used in the final version of the control system.

Initially, we implemented a system that attempted to travel straight to the candle, but encountered problems with both reflections and re-crossing the candle circle when exiting. For the second try, we simply wall followed until we found the candle, then extinguished it. This system found the candle reliably, and could also leave the room reliably. The second solution worked well enough, and at a small enough cost in time, that we used it in the final version of the control system.

We implemented the return home system described above. Return trips after the candle was found in the first or third rooms worked very well. Return trips from the upper left to lower right and vice versa also worked very well. We were able to implement returning home from the lower left to the island at the upper right by left wall following until that wall turned, then going straight for a short distance, then right wall following until we entered a room. This return trip worked reliably. The final return trip, from the island at the upper right to the room at the lower left was the most difficult. We attempted to simply mirror the mirror return trip, left wall following until the wall turned, then going straight for a short distance before right wall following into the room. While good conceptually, this design had a tendency to end in the upper left room rather than the lower left, because the robot would switch to right wall following while only partially done

making the initial turn out of the room. We attempted to correct this error by monitoring the front distance as well, and only switching from left wall following when the front distance sensor read a long distance in front of it. Unfortunately, even when optimally adjusted, that solution would sometimes switch too early and follow into the upper left room, and other times switch too late and run into the wall between the island at the upper right and the room at the lower right (see Appendix for underlying HC12 code).

# Cost Analysis

**Table 1 – Detail Budget**

| Item Description | Price/ea | quantity | cost | notes |
|---|---|---|---|---|
| H-BridgeLMD 18200T | $14.00 | 2 | $28.00 | |
| Perf-board 2.25”X1.8”: | $1.25 | 1 | $1.25 | |
| Tone decoder: NE567 | $1.00 | | $0.00 | donated |
| Schmitt trigger optoisolators | $1.50 | 6 | $9.00 | |
| 5V regulators: 7805 | $0.75 | | $0.00 | donated |
| Terminal housing (with pins) 2-pin (pair): | $0.75 | 1 | $0.75 | |
| Terminal housing (with pins) 3-pin (pair): | $1.00 | 7 | $7.00 | |
| Terminal housing (with pins) 5-pin (pair): | $1.75 | 1 | $1.75 | |
| 3.4kHz 9V buzzer: | $1.50 | 1 | $1.50 | |
| 3/32” shrink tubing (price per inch) | $0.10 | 10 | $1.00 | |
| IR distance sensors:0-80cm: GP2D12: | $10.00 | 5 | $50.00 | |
| 10” diam. 1/16” (thick) Al. disks: | $7.00 | 2 | $14.00 | |
| Maxon 6-Watt 22mm motors (grey body): | $60.00 | 2 | $0.00 | Donated |
| 12V lead-acid chargers: | $9.00 | 1 | $9.00 | |
| 9V Alkaline batteries: | $2.00 | 1 | $2.00 | |
| 12V 1.4Ahr NiCad battery pack (Powerline) | $5.00 | 1 | $5.00 | |
| Fuse holder (GMA fuses): | $1.00 | 1 | $1.00 | |
| GMA fuses(1,2,4,6A): | $0.20 | 2 | $0.40 | |
| 20-pin male header | $0.50 | 3 | $1.50 | |
| IDC connectors (female)2X10 | $1.00 | 4 | $4.00 | |
| 2x10 (female) header housing: | $2.00 | 2 | $4.00 | |
| crimp contacts: (i.e. pins for above) | $0.10 | 37 | $3.70 | |
| Electret mic (with-leads): | $1.50 | 1 | $1.50 | |
| Polarized single-row housing:5-pin housing: | $0.70 | 2 | $1.40 | |
| Reflective IR emitter/detector pair: | $1.50 | 4 | $6.00 | |
| Misc. switches: | $0.50 | 2 | $1.00 | |
| Caster wheel assembly | $10.00 | 1.5 | $15.00 | |
| 10-pin ribbon cable (price per foot) | $0.50 | 6 | $3.00 | |
| Misc. Resistors, Capacitors, Ics | $0.00 | | $0.00 | donated |

| | | | |
|---|---|---|---|
| **Items from Norton's Shop** | | | |
| Terminal housing (with pins) 2-pin (pair): | $2.00 | 3 | $6.00 |
| Terminal housing (with pins) 3-pin (pair): | $2.00 | 11 | $22.00 |
| Terminal housing (with pins) 4-pin (pair): | $2.00 | 4 | $8.00 |
| 6-32 spa`cers from machine shop (G5111) | $0.10 | 4 | $0.40 |
| 6-32 spacers from machine shop (G5090) | $0.25 | 4 | $1.00 |
| **Electronic Parts Inc.** | | | |
| Analog Mux (NTE4051B) | $4.10 | 1 | $4.10 |
| 25-turn pots 1M Ohm (2) | $3.52 | 4 | $14.08 |
| 2x10 pin ribbon connectors (5) | $4.34 | 1 | $4.34 |
| GRAND TOTAL | | | $232.67 |

# Results and Conclusion

Well, our robot placed 4th at Trinity and 1st at the local competition so we are very happy about how our robot performs.


Figure 13 - Trinity Plaque

# References

1. Trinity College Fire Fighting Home Robot Contest Official Website (http://www.trincoll.edu/events/robot/)
2. NMT EE382 Junior Design Website (http://www.ee.nmt.edu/~wedeward/EE382/SP01/ee382.html)
3. Annual NMT Robot Drag Race (http://www.ee.nmt.edu/vtour/temp/dragrace/dragrace.html)

# Appendices

## *List of Figures*

**Table 2**



**Table 3 – GP2D12 Calibration**

| Distance (cm) | average (V) | sensor #1 (V) | sensor #2 (V) | sensor #3 (V) |
|---|---|---|---|---|
| **8.00** | **2.62** | 2.64 | 2.56 | 2.66 |
| **10.00** | **2.46** | 2.45 | 2.45 | 2.47 |
| **12.00** | **2.10** | 2.10 | 2.10 | 2.10 |
| **14.00** | **1.83** | 1.83 | 1.82 | 1.83 |
| **16.00** | **1.63** | 1.64 | 1.61 | 1.65 |
| **18.00** | **1.50** | 1.50 | 1.48 | 1.51 |
| **20.00** | **1.41** | 1.41 | 1.38 | 1.43 |
| **22.00** | **1.34** | 1.34 | 1.32 | 1.37 |
| **24.00** | **1.31** | 1.30 | 1.29 | 1.33 |
| **26.00** | **1.27** | 1.27 | 1.25 | 1.28 |
| **28.00** | **1.24** | 1.23 | 1.23 | 1.25 |
| **30.00** | **1.19** | 1.20 | 1.18 | 1.20 |
| **32.00** | **1.16** | 1.17 | 1.14 | 1.16 |
| **34.00** | **1.12** | 1.13 | 1.10 | 1.12 |
| **36.00** | **1.08** | 1.09 | 1.07 | 1.08 |
| **38.00** | **1.05** | 1.06 | 1.03 | 1.05 |
| **40.00** | **1.01** | 1.03 | 1.00 | 1.01 |
| **42.00** | **0.99** | 1.01 | 0.98 | 0.98 |
| **44.00** | **0.95** | 0.98 | 0.94 | 0.93 |

| | | | | |
|---|---|---|---|---|
| **46.00** | **0.92** | 0.95 | 0.90 | 0.91 |
| **48.00** | **0.88** | 0.91 | 0.86 | 0.87 |
| **50.00** | **0.85** | 0.88 | 0.84 | 0.84 |
| **52.00** | **0.82** | 0.86 | 0.80 | 0.81 |
| **54.00** | **0.80** | 0.84 | 0.78 | 0.78 |
| **56.00** | **0.78** | 0.82 | 0.76 | 0.76 |
| **58.00** | **0.75** | 0.80 | 0.73 | 0.73 |
| **60.00** | **0.73** | 0.78 | 0.71 | 0.71 |
| **62.00** | **0.71** | 0.74 | 0.70 | 0.70 |
| **64.00** | **0.68** | 0.72 | 0.68 | 0.65 |
| **66.00** | **0.67** | 0.70 | 0.65 | 0.65 |
| **68.00** | **0.65** | 0.68 | 0.64 | 0.62 |
| **70.00** | **0.63** | 0.66 | 0.62 | 0.60 |
| **72.00** | **0.61** | 0.63 | 0.60 | 0.59 |
| **74.00** | **0.57** | 0.58 | 0.58 | 0.56 |
| **76.00** | **0.55** | 0.58 | 0.54 | 0.54 |
| **78.00** | **0.51** | 0.52 | 0.52 | 0.50 |
| **80.00** | **0.51** | 0.51 | 0.52 | 0.49 |
| **82.00** | **0.49** | 0.50 | 0.50 | 0.48 |

## *Altera Code*

```
% Quadrature Encoding with Digital Filters, 12 bit position and   %
% speed registers.                                                %
% This is written to use the daughter board to the HC12 with the  %
% 256 Altera Chip.                                                %
% Bill Willems                                                    %
% 4/3/01                                                          %
% I know you don't like it all in one file but I do...            %
SUBDESIGN motors
(
      EXP_ENn     :  INPUT;   % Memory expansion enabled if low   %
                              % (active low – anything with an 'n'%
                              % at the end of it is active low)   %
      E           :  INPUT;   % E-Clock %
      R_W         :  INPUT;   % R/W Line %
      RESETn      :  INPUT;   % Reset line %
      LSTRBn      :  INPUT;   % 8-bit or 16-bit access? %
      PA[7..0]    :  BIDIR;   % Address and Data (15-8) from HC12 %
      PB[7..0]    :  BIDIR;   % Address and Data (7-0) from HC12 %
      WEn         :  OUTPUT;  % Write Enable to memory %
      OEn         :  OUTPUT;  % Output Enable to memory %
      CEn         :  OUTPUT;  % Chip Select for SRAM Memory %
      CS_MEM_ODDn :  OUTPUT;  % Chip Select for odd addresses %
      CS_MEM_EVENn:  OUTPUT;  % Chip Select for even addresses %
      A[15..0]    :  OUTPUT;  % Demultiplexed address bits %
      IRQn        :  BIDIR;   % Open-drain output for future use %

      % Motor Stuff %
      R_chA       :  INPUT;   % Right motor - CHannel A %
      R_chB       :  INPUT;
      L_chA       :  INPUT;   % Left motor %
      L_chB       :  INPUT;
      spd_pwm     :  INPUT;   % resets the speed counters & registers %
                              % PWM from HC12 %
)

VARIABLE
      % Memory Expansion %
      demux[15..0]:  DFF;     % Demultiplexed address internal %
      CS_MEM      :  NODE;    % Tell's when address is in our range %
                              % of memory %
      IDB[15..0]  :  NODE;    % Internal Data Bus to send to HC12 %
      PA_OE       :  NODE;    % High when we want to send data to     %
                              % HC12 on the HC12's Port A             %
      PB_OE       :  NODE;    % Port B %

      % state machines and associated signals %
      spd_machine       :  MACHINE  % generates narrow reset_sig_n %
                           WITH STATES (s0,s1,s2,s3);
      reset_sig_n       :  NODE;    % to reset speed accumulators %

      L_chA_DF_machine  :  MACHINE  % Digital Filter for L_chA %
                           WITH STATES (la0,la1,la2,la3,la4,la5);
      L_chB_DF_machine  :  MACHINE  % Digital Filter for L_chB %
                           WITH STATES (lb0,lb1,lb2,lb3,lb4,lb5);
```

```
L_chA_DF    :  NODE; % Signals generated from L_chX_DF_machine %
L_chB_DF    :  NODE; % This is channel X digitally filtered.   %

R_chA_DF_machine: MACHINE
                  WITH STATES (ra0,ra1,ra2,ra3,ra4,ra5);
R_chB_DF_machine: MACHINE
                  WITH STATES (rb0,rb1,rb2,rb3,rb4,rb5);
R_chA_DF    :  NODE;
R_chB_DF    :  NODE;

L_quad_machine    :  MACHINE  % to generate a quadrature clock %
                     WITH STATES (lq0,lq1);
L_quad_clk        :  NODE;    % generated by L_quad_machine %
L_AxorB           :  NODE;    % Obviously, this is A xor B for %
                              % the L_quad_machine %

R_quad_machine    :  MACHINE
                     WITH STATES (rq0,rq1);
R_quad_clk        :  NODE;
R_AxorB           :  NODE;

E_div_2     :  DFF; % For Dividing the E clk down.              %
E_div_4     :  DFF; % Used to drive L_ and R_chB_DF_machines    %
E_div_8     :  DFF;

R_pos_div_2 :  DFF; % This gives me less resolution on      %
R_pos_div_4 :  DFF; % the position accumulators. If I don't %
R_pos_div_8 :  DFF; % do this, then the accumulators will   %
R_pos_div_16:  DFF; % overflow about 14 times / revolution. %
R_pos_div_32:  DFF;
L_pos_div_2 :  DFF;
L_pos_div_4 :  DFF;
L_pos_div_8 :  DFF;
L_pos_div_16:  DFF;
L_pos_div_32:  DFF;

% Motor Stuff %
R_speed[11..0]    :     DFF;  % Accumulators    %
L_speed[11..0]    :     DFF;
R_position[11..0] :     DFF;
L_position[11..0] :     DFF;
R_spd_reg[11..0]  :     DFF;  % Registers                        %
L_spd_reg[11..0]  :     DFF;  % Position doesn't get registers. %
R_direction       :     DFF;  % Direction: GND = Forward;    %
L_direction       :     DFF;  % VCC = Reverse. NOTE: Forward %
                              % is RELATIVE to the ROBOT!    %
% Motor Addressing %
motor_addr  :     NODE; % We use these when accessing       %
L_dir_addr  :     NODE; % direction, speed and position     %
L_spd_addr  :     NODE; % stuff. Using nodes saves some     %
L_pos_addr  :     NODE; % space instead of running demux    %
R_dir_addr  :     NODE; % lines all over the place.         %
R_spd_addr  :     NODE;
R_pos_addr  :     NODE;
R_pos_resetn      :     NODE; % Reset position counters %
L_pos_resetn      :     NODE;
```

```
BEGIN

%*********************************************************************%
% Address decoding and demultiplexing                                 %
% Latch address on rising edge of E clock                             %
%*********************************************************************%
      demux[15..8].d = PA[7..0];
      demux[7..0].d = PB[7..0];
      demux[15..0].clk = E;
      demux[15..0].clrn = RESETn;
      A[15] = VCC; % Let's use the upper 32k of the 16x64kB SRAM chip %
      A[14..0] = demux[15..1].q;    % Needed since the SRAM chip  %
                                    % addresses 16-bit words.     %

% Enable writes when E high and R/W low %
      IF (EXP_ENn == GND) & (E == VCC) & (R_W == GND) THEN
            WEn = GND;
      ELSE
            WEn = VCC;
      END IF;

% Enable reads when E high and R/W high %
      IF (EXP_ENn == GND) & (E == VCC) & (R_W == VCC) THEN
            OEn = GND;
      ELSE
            OEn = VCC;
      END IF;

% Access external memory when addresses in the range 0x1000 to 0x7fff %
      IF (EXP_ENn==GND) & (E==VCC) &
         ((demux[15..0].q >= H"1000") & (demux[15..0].q <= H"7fff"))
      THEN
            CS_MEM = VCC;
      ELSE
            CS_MEM = GND;
      END IF;
      CEn = !CS_MEM;

% Access even memory locations when address is even %
      IF (CS_MEM == VCC) & (demux[0].q == GND) THEN
            CS_MEM_EVENn = GND;
      ELSE
            CS_MEM_EVENn = VCC;
      END IF;

% Access odd memory locations when ... %
      IF (CS_MEM == VCC) & (LSTRBn == GND) THEN
            CS_MEM_ODDn = GND;
      ELSE
            CS_MEM_ODDn = VCC;
      END IF;
```

```
% ------------------------------------------------------------------- %
% LSTRBn (Low Byte Strobe) works as follows:                          %
%                                                                     %
% LSTRBn    A0    Type of Access                                      %
% ------------------------------------------------------------------- %
%   0       0     16-bit access of even address                       %
%                 (even 0xXXXX and odd 0xXXXX+1)                      %
%   0       1     8-bit access of odd address                         %
%                                                                     %
%   1       0     8-bit access of even address                        %
%                                                                     %
%   1       1     16-bit access of odd address                        %
%                 (NOT USED FOR EXTERNAL ADDRESSING)                  %
% ------------------------------------------------------------------- %

% HC12 will access the stuff we are interested like so:     %
% An 8bit READ from address  0x0C01 => L_direction          %
% A 16bit READ from address  0x0C02 => L_spd_reg            %
% A 16bit READ from address  0x0C04 => L_position           %
% Any access to/from address 0x0C06=> R_pos_resetn          %
% An 8bit READ from address  0x0C07 => R_direction          %
% A 16bit READ from address  0x0C08 => R_spd_reg            %
% A 16bit READ from address  0x0C0A => R_position           %
% Any access to/from address 0x0C0C=> L_pos_resetn          %

% Notice that 8bit reads are odd addresses and 16bit reads are even.  %
% I set it up this way because it makes handling LSTRBn easier. Here, %
% LSTRBn is always low.                                               %

% These will be handy... %
      IF (demux[15..8].q == H"0C") THEN motor_addr = VCC;
      ELSE motor_addr = GND; END IF;
      IF (demux[3..0].q == H"1") THEN L_dir_addr = VCC;
      ELSE L_dir_addr = GND; END IF;
      IF (demux[3..0].q == H"2") THEN L_spd_addr = VCC;
      ELSE L_spd_addr = GND; END IF;
      IF (demux[3..0].q == H"4") THEN L_pos_addr = VCC;
      ELSE L_pos_addr = GND; END IF;
      IF (demux[3..0].q == H"6") & (motor_addr == VCC) THEN
            R_pos_resetn = GND;
      ELSE R_pos_resetn = VCC; END IF;
      IF (demux[3..0].q == H"7") THEN R_dir_addr = VCC;
      ELSE R_dir_addr = GND; END IF;
      IF (demux[3..0].q == H"8") THEN R_spd_addr = VCC;
      ELSE R_spd_addr = GND; END IF;
      IF (demux[3..0].q == H"A") THEN R_pos_addr = VCC;
      ELSE R_pos_addr = GND; END IF;
      IF (demux[7..0].q == H"0C") & (motor_addr == VCC) THEN
            L_pos_resetn = GND;
      ELSE L_pos_resetn = VCC; END IF;
```

```
% If 16 bit reading from addresses related to spd and pos, we need to %
% drive the Port A lines when E is high.                               %
      IF (EXP_ENn == GND) & (R_W == VCC) & (E == VCC) &
         (motor_addr == VCC) &
         ( (L_spd_addr == VCC) # (L_pos_addr == VCC)
         # (R_spd_addr == VCC) # (R_pos_addr == VCC) )
      THEN
            PA_OE = VCC;
      ELSE
            PA_OE = GND;
      END IF;

% When 8 bit accessing related to speed or position or direction      %
% stuff, we need to drive Port B lines when E is high.                 %
      IF (EXP_ENn == GND) & (R_W == VCC) & (E == VCC) &
         (LSTRBn == GND) & (motor_addr == VCC) &
         ((L_dir_addr == VCC)#(L_spd_addr == VCC)#(L_pos_addr == VCC)
         # (R_dir_addr == VCC)#(R_spd_addr == VCC)#(R_pos_addr == VCC))
      THEN
            PB_OE = VCC;
      ELSE
            PB_OE = GND;
      END IF;

% Put the internal data bus values onto Port A %
      PA[7] = TRI(IDB[15], PA_OE);
      PA[6] = TRI(IDB[14], PA_OE);
      PA[5] = TRI(IDB[13], PA_OE);
      PA[4] = TRI(IDB[12], PA_OE);
      PA[3] = TRI(IDB[11], PA_OE);
      PA[2] = TRI(IDB[10], PA_OE);
      PA[1] = TRI(IDB[9],  PA_OE);
      PA[0] = TRI(IDB[8],  PA_OE);

%  onto Port B... %
      PB[7] = TRI(IDB[7], PB_OE);
      PB[6] = TRI(IDB[6], PB_OE);
      PB[5] = TRI(IDB[5], PB_OE);
      PB[4] = TRI(IDB[4], PB_OE);
      PB[3] = TRI(IDB[3], PB_OE);
      PB[2] = TRI(IDB[2], PB_OE);
      PB[1] = TRI(IDB[1], PB_OE);
      PB[0] = TRI(IDB[0], PB_OE);

%****************************%
% Motor Encoder Decoder Stuff %
%****************************%

      % Set clocks and clears %
      R_speed[].clk = R_quad_clk;   % from quad_machine %
      R_speed[].clrn = reset_sig_n; % from spd_machine  %

      R_spd_reg[].clk = spd_pwm;    % PWM from HC12 %
      R_spd_reg[].clrn = RESETn;

      R_position[].clk = R_pos_div_32.q;
      R_position[].clrn = R_pos_resetn;
```

```
R_direction.clk = R_chB_DF;   % on rising edge of B, direction  %
                              % corresponds to A                %
R_direction.clrn = RESETn;

L_speed[].clk = L_quad_clk;
L_speed[].clrn = reset_sig_n;

L_spd_reg[].clk = spd_pwm;
L_spd_reg[].clrn = RESETn;

L_position[].clk = L_pos_div_32.q;
L_position[].clrn = L_pos_resetn;

L_direction.clk = L_chB_DF;
L_direction.clrn = RESETn;

% Divide down the Digital Filter outputs to clock the Position  %
% Accumulators so we get less resolution. Otherwise, the        %
% accumulators would overflow about 14 times per revolution.    %
R_pos_div_2.clk = R_chA_DF;
R_pos_div_4.clk = R_pos_div_2.q;
R_pos_div_8.clk = R_pos_div_4.q;
R_pos_div_16.clk = R_pos_div_8.q;
R_pos_div_32.clk = R_pos_div_16.q;

R_pos_div_2.d = !R_pos_div_2.q;
R_pos_div_4.d = !R_pos_div_4.q;
R_pos_div_8.d = !R_pos_div_8.q;
R_pos_div_16.d = !R_pos_div_16.q;
R_pos_div_32.d = !R_pos_div_32.q;

L_pos_div_2.clk = L_chA_DF;
L_pos_div_4.clk = L_pos_div_2.q;
L_pos_div_8.clk = L_pos_div_4.q;
L_pos_div_16.clk = L_pos_div_8.q;
L_pos_div_32.clk = L_pos_div_16.q;

L_pos_div_2.d = !L_pos_div_2.q;
L_pos_div_4.d = !L_pos_div_4.q;
L_pos_div_8.d = !L_pos_div_8.q;
L_pos_div_16.d = !L_pos_div_16.q;
L_pos_div_32.d = !L_pos_div_32.q;

% Divide down the E_clk to clock the Digital Filters. %
E_div_2.clk = E;
E_div_4.clk = E_div_2.q;
E_div_8.clk = E_div_4.q;

E_div_2.d = !E_div_2.q;
E_div_4.d = !E_div_4.q;
E_div_8.d = !E_div_8.q;

% Implement L_chA_DF_machine                                    %
% This is a digital filter for L_chB. If we don't               %
% do this, L_chB is NOISY!                                      %
L_chA_DF_machine.clk = E_div_8.q;
L_chA_DF_machine.reset = !RESETn;
```

```
CASE L_chA_DF_machine IS
	WHEN la0 =>
		L_chA_DF = VCC;
		IF L_chA == GND THEN L_chA_DF_machine = la1;
		ELSE L_chA_DF_machine = la0;
		END IF;
	WHEN la1 =>
		L_chA_DF = VCC;
		IF L_chA == GND THEN L_chA_DF_machine = la2;
		ELSE L_chA_DF_machine = la0;
		END IF;
	WHEN la2 =>
		L_chA_DF = VCC;
		IF L_chA == GND THEN L_chA_DF_machine = la3;
		ELSE L_chA_DF_machine = la0;
		END IF;
	WHEN la3 =>
		L_chA_DF = GND;
		IF L_chA == VCC THEN L_chA_DF_machine = la4;
		ELSE L_chA_DF_machine = la3;
		END IF;
	WHEN la4 =>
		L_chA_DF = GND;
		IF L_chA == VCC THEN L_chA_DF_machine = la5;
		ELSE L_chA_DF_machine = la3;
		END IF;
	WHEN la5 =>
		L_chA_DF = GND;
		IF L_chA == VCC THEN L_chA_DF_machine = la0;
		ELSE L_chA_DF_machine = la3;
		END IF;
END CASE;

% Implement L_chB_DF_machine %
L_chB_DF_machine.clk = E_div_8.q;
L_chB_DF_machine.reset = !RESETn;
CASE L_chB_DF_machine IS
	WHEN lb0 =>
		L_chB_DF = VCC;
		IF L_chB == GND THEN L_chB_DF_machine = lb1;
		ELSE L_chB_DF_machine = lb0;
		END IF;
	WHEN lb1 =>
		L_chB_DF = VCC;
		IF L_chB == GND THEN L_chB_DF_machine = lb2;
		ELSE L_chB_DF_machine = lb0;
		END IF;
	WHEN lb2 =>
		L_chB_DF = VCC;
		IF L_chB == GND THEN L_chB_DF_machine = lb3;
		ELSE L_chB_DF_machine = lb0;
		END IF;
	WHEN lb3 =>
		L_chB_DF = GND;
		IF L_chB == VCC THEN L_chB_DF_machine = lb4;
		ELSE L_chB_DF_machine = lb3;
		END IF;
```

```
	WHEN lb4 =>
		L_chB_DF = GND;
		IF L_chB == VCC THEN L_chB_DF_machine = lb5;
		ELSE L_chB_DF_machine = lb3;
		END IF;
	WHEN lb5 =>
		L_chB_DF = GND;
		IF L_chB == VCC THEN L_chB_DF_machine = lb0;
		ELSE L_chB_DF_machine = lb3;
		END IF;
END CASE;

% Implement R_chA_DF_machine %
R_chA_DF_machine.clk = E_div_8.q;
R_chA_DF_machine.reset = !RESETn;
CASE R_chA_DF_machine IS
	WHEN ra0 =>
		R_chA_DF = VCC;
		IF R_chA == GND THEN R_chA_DF_machine = ra1;
		ELSE R_chA_DF_machine = ra0;
		END IF;
	WHEN ra1 =>
		R_chA_DF = VCC;
		IF R_chA == GND THEN R_chA_DF_machine = ra2;
		ELSE R_chA_DF_machine = ra0;
		END IF;
	WHEN ra2 =>
		R_chA_DF = VCC;
		IF R_chA == GND THEN R_chA_DF_machine = ra3;
		ELSE R_chA_DF_machine = ra0;
		END IF;
	WHEN ra3 =>
		R_chA_DF = GND;
		IF R_chA == VCC THEN R_chA_DF_machine = ra4;
		ELSE R_chA_DF_machine = ra3;
		END IF;
	WHEN ra4 =>
		R_chA_DF = GND;
		IF R_chA == VCC THEN R_chA_DF_machine = ra5;
		ELSE R_chA_DF_machine = ra3;
		END IF;
	WHEN ra5 =>
		R_chA_DF = GND;
		IF R_chA == VCC THEN R_chA_DF_machine = ra0;
		ELSE R_chA_DF_machine = ra3;
		END IF;
END CASE;

% Implement R_chB_DF_machine %
R_chB_DF_machine.clk = E_div_8.q;
R_chB_DF_machine.reset = !RESETn;
CASE R_chB_DF_machine IS
	WHEN rb0 =>
		R_chB_DF = VCC;
		IF R_chB == GND THEN R_chB_DF_machine = rb1;
		ELSE R_chB_DF_machine = rb0;
		END IF;
```

```
        WHEN rb1 =>
              R_chB_DF = VCC;
              IF R_chB == GND THEN R_chB_DF_machine = rb2;
              ELSE R_chB_DF_machine = rb0;
              END IF;
        WHEN rb2 =>
              R_chB_DF = VCC;
              IF R_chB == GND THEN R_chB_DF_machine = rb3;
              ELSE R_chB_DF_machine = rb0;
              END IF;
        WHEN rb3 =>
              R_chB_DF = GND;
              IF R_chB == VCC THEN R_chB_DF_machine = rb4;
              ELSE R_chB_DF_machine = rb3;
              END IF;
        WHEN rb4 =>
              R_chB_DF = GND;
              IF R_chB == VCC THEN R_chB_DF_machine = rb5;
              ELSE R_chB_DF_machine = rb3;
              END IF;
        WHEN rb5 =>
              R_chB_DF = GND;
              IF R_chB == VCC THEN R_chB_DF_machine = rb0;
              ELSE R_chB_DF_machine = rb3;
              END IF;
END CASE;

% Implement the L_quad_machine                                    %
% This generates a pulse (one E_clk wide) whenever chA or         %
% chB changes.                                                    %
L_quad_machine.clk = E;
L_quad_machine.reset = !RESETn;
L_AxorB = L_chA_DF $ L_chB_DF;
TABLE
L_quad_machine,   L_AxorB     =>    L_quad_clk, L_quad_machine;
lq0,              0           =>    0,          lq0;
lq0,              1           =>    1,          lq1;
lq1,              0           =>    1,          lq0;
lq1,              1           =>    0,          lq1;
END TABLE;

% Implement the R_quad_machine %
R_quad_machine.clk = E;
R_quad_machine.reset = !RESETn;
R_AxorB = R_chA_DF $ R_chB_DF;
TABLE
R_quad_machine,   R_AxorB     =>    R_quad_clk, R_quad_machine;
rq0,              0           =>    0,          rq0;
rq0,              1           =>    1,          rq1;
rq1,              0           =>    1,          rq0;
rq1,              1           =>    0,          rq1;
END TABLE;
```

```
% Implement the spd_machine:                                    %
% This generates a narrow reset signal one E_clk AFTER          %
% spd_pwm latches the value of the accumulators into            %
% spd_regs.                                                     %
% Otherwise the registers are latching the same time the        %
% accumulators are reseting.                                    %
% spd_pwm clocks the regs                                       %
% reset_sig_n resets the accumulators                           %
spd_machine.clk = E;
spd_machine.reset = !RESETn;
CASE spd_machine IS
      WHEN s0 =>
            reset_sig_n = VCC;
            IF spd_pwm == GND THEN spd_machine = s0;
            ELSE spd_machine = s1;
            END IF;
      WHEN s1 =>
            reset_sig_n = VCC;
            spd_machine = s2;
      WHEN s2 =>
            reset_sig_n = GND;
            spd_machine = s3;
      WHEN s3 =>
            reset_sig_n = VCC;
            IF spd_pwm == VCC THEN spd_machine = s3;
            ELSE spd_machine = s0;
            END IF;
END CASE;

% Wire speed accumulators to their registers %
R_spd_reg[].d = R_speed[].q;
L_spd_reg[].d = L_speed[].q;

% Which way are we going? %
R_direction.d = R_chA_DF;
L_direction.d = !L_chA_DF;

% Increment Speed counters %
R_speed[].d = R_speed[].q + 1;
L_speed[].d = L_speed[].q + 1;

% Increment or Decrement pos counters dependent upon direction. %
IF (R_direction.q == GND) THEN
      R_position[].d = R_position[].q + 1;
ELSE
      R_position[].d = R_position[].q - 1;
END IF;
IF (L_direction.q == GND) THEN
      L_position[].d = L_position[].q + 1;
ELSE
      L_position[].d = L_position[].q - 1;
END IF;
```

```
      % NOTE: We don't need to worry about the position and speed %
      % counters being on a different clock because the data is   %
      % only latched on the rising edge of channel B. We can keep %
      % putting [].q + 1 on the [].d lines all we want but that   %
      % won't be latched in until we get a rising edge. See, I    %
      % thought about this – it wasn’t just pure luck that it     %
      % works!!                                                   %

      % Now that all of the counters are counting - put their     %
      % values on the bus when the appropriate _addr nodes are    %
      % high.                                                     %

      IF (motor_addr == VCC) THEN
            IF (L_dir_addr == VCC) THEN % L_direction %
                  IDB[15..1] = 0;
                  IDB[0] = L_direction.q;
            END IF;
            IF (L_spd_addr == VCC) THEN % L_spd_reg %
                  IDB[15..12] = 0;
                  IDB[11..0] = L_spd_reg[].q;
            END IF;
            IF (L_pos_addr == VCC) THEN % L_position %
                  IDB[15..12] = L_position[11].q; % sign extension %
                  IDB[11..0] = L_position[].q;
            END IF;
            IF (R_dir_addr == VCC) THEN % R_direction %
                  IDB[15..1] = 0;
                  IDB[0] = R_direction.q;
            END IF;
            IF (R_spd_addr == VCC) THEN % R_spd_reg %
                  IDB[15..12] = 0;
                  IDB[11..0] = R_spd_reg[].q;
            END IF;
            IF (R_pos_addr == VCC) THEN % R_position %
                  IDB[15..12] = R_position[11].q; % sign extension %
                  IDB[11..0] = R_position[].q;
            END IF;
      ELSE  % Anything else concerns the memory chips - put 0's on %
            % the internal data bus. %
            IDB[15..0] = H"0000";
      END IF;

% IRQn is an open drain output; this will make it high impedance  %
% always. IRQn is wired to the IRQ pin on the HC12 - high         %
% impedance means no interrupts.                                  %
      IRQn = OPNDRN(VCC);
END;
```

## *HC12 Code*

*See attached files code.ps (full program) and code2.ps (system include files).*


code.ps


code2.ps