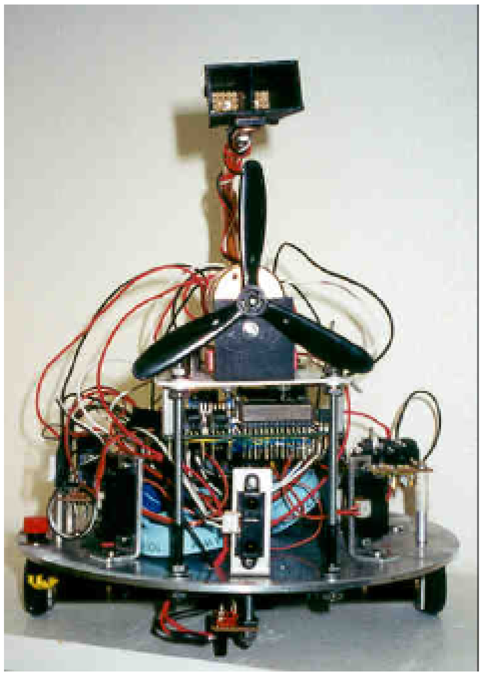
F.U.B.A.R



Group 3

Heather Bitsoi

David Byrd

Sam Field

Chris Reiten

EE382 Spring 2001

Abstract

As part of the undergraduate curriculum in Electrical Engineering at New Mexico Tech, students must work in groups of three or four to build a fire-fighting robot that is capable of competing in the Trinity College Home Fire-fighting Robot Contest. This is a project that encompasses all of the skills and knowledge the students have gained in their studies at Tech. To complete this project, our group had to accomplish many tasks. We started with drive system and motor selection, and moved on to chassis design and construction. To enable the robot to "see" its environment we had to incorporate a variety of sensors, some off the shelf and some custom built, into the design. At the core of the project is the Motorola 68HC12 microcontroller, which is responsible for turning the data from the sensors and control switches into control signals for the robot's motors. Programming the HC12 proved to be the most daunting task of the semester, with all four group members putting time in on the program. Other requirements included closed loop motor control and optoisolation between the robot's low power and high power systems. In the course of building the robot and fulfilling the various design requirements, as well as presenting our work at periodic design reviews, we gained valuable knowledge and skills that will be used throughout our careers.

Table of Contents

List of Figures

Introduction

Every spring, the New Mexico Tech Electrical Engineering Department offers a course called Introduction to Design. Taken by juniors in the Department, the purpose of this course is to allow the students to put to use the knowledge and skills they have accumulated in their studies at Tech.

The goal of the course is to build a firefighting robot following the guidelines set by the Trinity College Firefighting Robot Contest. By the rules of the contest, each robot must be completely autonomous, and must fit in a 12-inch cube. Beyond that, there are few rules set regarding robot design. After spending the first week of class learning about many different options regarding robot design, parts, and implementation, our group was set free to begin the project.

Drive System and Motor Selection

The first task was to figure out what type of drive system we would use, and what type of chassis to put it on. After briefly considering all of the options, we decided to use differential drive. A differential drive system consists of two motors, each driving one wheel, and a caster wheel for balance. By independently controlling the speed and direction of each motor, the robot can be made to go in a straight line, pivot on its vertical axis, or anything in between. This configuration has many advantages, including very good maneuverability and relative ease of programming. Another advantage is mechanical simplicity, an important consideration given the limited time for the project. Its primary disadvantage is potential balance problems, especially on rough surfaces. We judged this to be of minor importance in the robot's environment, and indeed we encountered no problems with balance throughout the semester.

Once the drive system had been determined, the next step was motor selection. The EE department had five types of motors in stock, ranging from a 5 watt Maxon to a 28 watt Pittman. We began by determining the power we would need using the equation

$$\text{power} = \text{force x speed} = mgv\sin(\theta)$$

where m is the mass of the robot, g is the acceleration of gravity, v is the velocity of the robot, and θ is the angle of incline (10° for Trinity ramps). Assuming a 3 kg robot moving at 1 m/s, we arrived at a total necessary power of 5.1 watts. Since there would be two motors, even the 5-watt Maxons would provide nearly twice the necessary power. Based on this calculation, we quickly ruled out the 28-watt Pittmans as too powerful for our project. Besides their high power rating, we were concerned that their relatively high gear ratio of 5.6:1 would make them difficult to control, and at 24 volts, they had the highest voltage rating of all the choices. Next to go were the 11-watt Maxons, for much the same reason as the Pittmans. Another disadvantage of these motors was their 100-count/rev encoders. We saw no need to sacrifice encoder resolution for power we did not need. This left three choices, all of them similar in power rating. The 5-watt Maxons, with a 128-count/rev single channel encoder, went first. We finally chose the green 6-watt Maxons for their 500 count/rev quadrature encoders, 14:1 gear ratio, and availability.

High and Low Power Battery Selection

For the low power supply, we decided to use 6 AA Nickel Metal-Hydride batteries. This gave us 7.8 volts, and with a 7805 voltage regulator, we ran all of our low power items at 5 volts. The batteries were rated at 1500mAh, so they gave us plenty of working time between charges.

For the high power side of the robot, we used a lead acid battery. This battery was rated at 12 volts and 200mAh. This battery worked well for our fan and motors. Its high capacity allowed us a couple of hours of operation between charges.

Chassis Construction

Having selected the motors and the power sources, the next step in the project was the construction of the chassis. We decided to base the chassis on a circular aluminum plate 10 inches in diameter that we obtained from the junk box. This had several advantages, the primary one being that a circular robot would not catch on corners in the maze. Also, aluminum is strong and easy to machine, and coming out of the junk box, the plate was free.

The next step was mounting the motors. To do this, we built two right angle brackets out of scrap aluminum from the instrument room. We drilled holes in these to match the drill pattern of the 14:1 gearhead on the motors and another hole to allow the motor shaft through. Once mounted on the main plate, these mounts proved very strong and gave us no problems during the semester. The motors were a close fit, with about ¼ inch between them at the center of the robot when mounted so that the wheels were completely under the main plate. For wheels we used the 1.5-inch wheels offered by the EE department. With aluminum hubs and rubber tires, these wheels were easy to mount on the motor shafts and gave plenty of traction. The encoder wires from the motors were run through a slot machined in the middle of the plate. A caster, also sourced from the junk box, was mounted at the rear of the robot.

To balance the robot, we decided to mount the batteries at the rear, behind the motors. Unfortunately, the size of the caster prevented us from mounting the lead acid battery under the main plate, and we had to put it on top, using Velcro to fasten it to the plate. The NiMH batteries were mounted on top of the lead acid battery and were held on by two screws run through the plate and the holes on the ends of the battery holder. Wing nuts on the screws kept everything in place and allowed easy removal of the batteries.

Also on the main plate, we mounted the distance sensors, the white line sensor, and the H-bridge and power distribution boards. The three Sharp distance sensors were mounted on right angle brackets that we machined from aluminum. The white line sensor was mounted under the plate on screws run through from the top. Using locknuts, we were able to securely and precisely position the flame sensor at a height that allowed for very reliable white line detection. The only problem was ground clearance. At about .25 inches from the floor, the flame sensor prohibited us from using ramps. The H-bridge and power distribution boards were mounted on top of the main plate, one on each side of the robot.

Next, we mounted the HC12 and the fan. These were both mounted on a rectangular aluminum plate held above the main deck with four pieces of all-thread. The fan determined the height of this plate. The rules for the Trinity competition state that the candle flame can be from 15 to 20 cm from the floor. Our fan blade had a diameter of 6 cm, so we decided that if its center was 17.5 cm from the floor, we would be able to extinguish any legal candle flame. This meant that the plate would be 3.6 inches above the main deck. The HC12 was mounted under this plate, a layout which had the advantage of protecting the microcontroller from any unforeseen impacts (a feature that came in handy). The only disadvantage was that it was relatively difficult to attach wires to the HC12. The fan was mounted on top of this plate, along with the Antex solid-state relay used to switch it, and the flame sensor. Also on top of the secondary plate was the Schmitt trigger board used to interface the flame and white line sensors to the HC12.

Overall, our chassis design was fairly clean and very sturdy. Its integrity was tested before the final design review when some unknown person apparently dropped something on top of the robot. The only damage suffered was to the flame sensor, which was broken off its mount (and taped back on by the perpetrator) and to the Schmitt trigger board, which had a few bent pins, causing a short that took about an hour to track down and fix. Other than that, we did not have any problems with reliability during the semester.

H-Bridge

The H-bridge is the circuitry that drives the motors based on input from the HC12. There were several to choose from, each with different characteristics.

H-bridge Circuitry Requirements

The H-bridge used needed to meet these requirements:

1. Operate at 530mA continuous current per channel
2. Operate at a voltage range of 10-14 volts
3. Ability to operate at a 4k Hz PWM input signal
4. Reasonably priced at to fit into the budget.

H-Bridge Comparison

To determine which chip best suits our needs; we chose five different chips and compared the following characteristics:

- Voltage Range
- Output Current (Peak)
- Output Current (Continuous)
- H-Bridges per Package
- Price

The five H-bridges we chose to compare, and their electrical characteristics are shown in figure 1.

H-Bridge	Voltage Range	Output Current (Peak)	Output Current (Cont.)	H-bridges per Package	Price
Allegro 2998	10V-50V	3A	2A	2	7.00
National Semiconductor LMD18200	12V-55V	6A	3A	1	14.00
National Semiconductor LMD18201	12V-55V	6A	3A	1	12.00
Thomson ST L298	6V-46V	3A	2.5A	2	8.00
Thomson ST L293D	5V-36V	1.2A	.6A	2	6.00

Figure 1. Comparison of H-bridges

H-Bridge Specification Analysis and Final Choice

In order to meet the dropout voltage requirements, the H-bridge needed to operate at a range down to 10 volts. It also needed to be able to handle 530mA of continuous current, as that is the maximum continuous current the Maxon motors can draw. We are running the PWM signals at 4k Hz, and all three chips can operate at that speed. The two Thomson chips and the Allegro chip met these requirements. After narrowing our search down to these three chips, we chose to use the Thomson LM293D because it met all of our needs, and also was the best value.

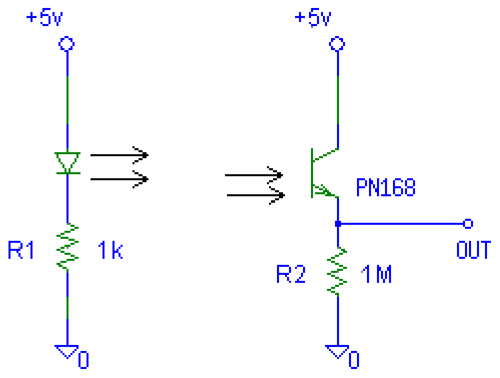
Details on Thomson H-bridge

Because the Thomson H-bridge was separated into four channels, we tied two channels on each side of the H-bridge together in order to be able to run the motors in both directions. In order for the H-bridge to operate properly, on each side of the H-bridge, one of the enable pins had to be a logic high, while the other had to be a logic low. We accomplished this by adding and inventor chip to our design.

We had four output lines going from the HC12 to the H-bridge, two PWM signals and a direction line for each motor. These signals were fed through optoisolators to reduce noise and to protect the HC12 from any voltage spikes. See Appendix A for a detailed schematic of the H-bridge and the optoisolators.

<u>White Line Sensor</u>

The white line sensor, while very important to the operation of the robot, is very simple in design. It consists of a matched infrared emitter/detector pair mounted in a single black plastic case. Mounted on a board under the main deck, the device has a focal length of about .25 inches.



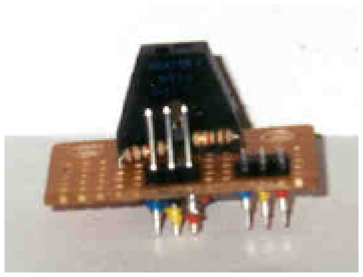


Figure 2 White line sensor schematic

Figure 3 White line sensor

Distance Sensors

In order for the robot to maneuver in the maze it had to be able to see the walls. We used the Sharp distance sensors GP2D12 and GP2D120. The GP2D120 sensor has a range of about 40-cm. We used those sensors for seeing the left and right walls. We mounted the sensors at a 45-degree angle (halfway between front and side) so that we could tell where the wall was with respect to the robot. We used the GP2D12 sensor for the front sensor so the robot could sense impending collisions. It has a range of about 80-cm. The following diagram shows the voltage response versus distance of the two sensors.

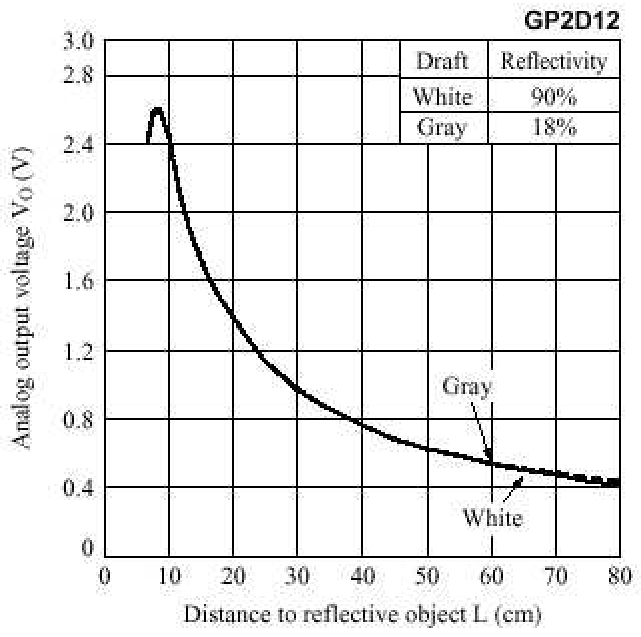


Figure 4 GP2D12 voltage/distance chart

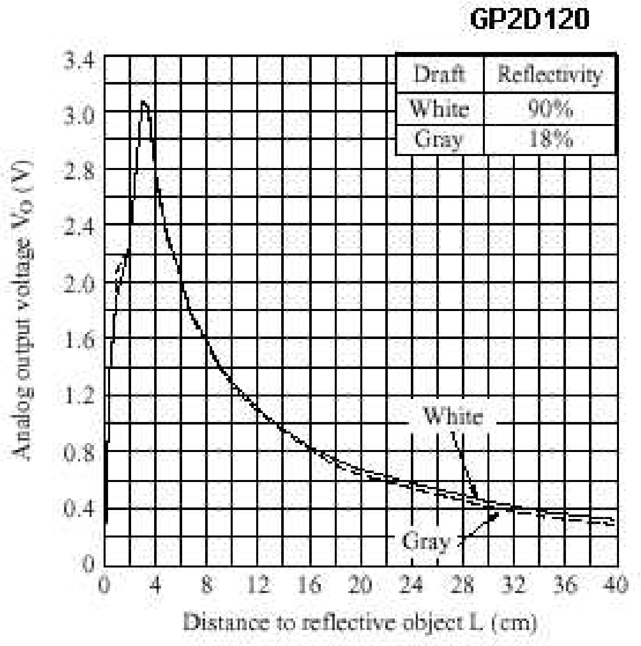


Figure 5 GP2D120 voltage/distance chart

Frequency to Voltage Converters

Our robot was required to have closed-loop speed control. In order to implement this we had to figure out the speed of the motors. The motor encoders put out a square wave with 500 rising edges per revolution. We used two National Semiconductor LM2907 Frequency to Voltage Converters, one for each motor, to determine our speed based on output voltage. We hooked the frequency to voltage converters to A/D ports six and seven of the HC12. The analog voltage ranged from about 0V to about 3.5V.

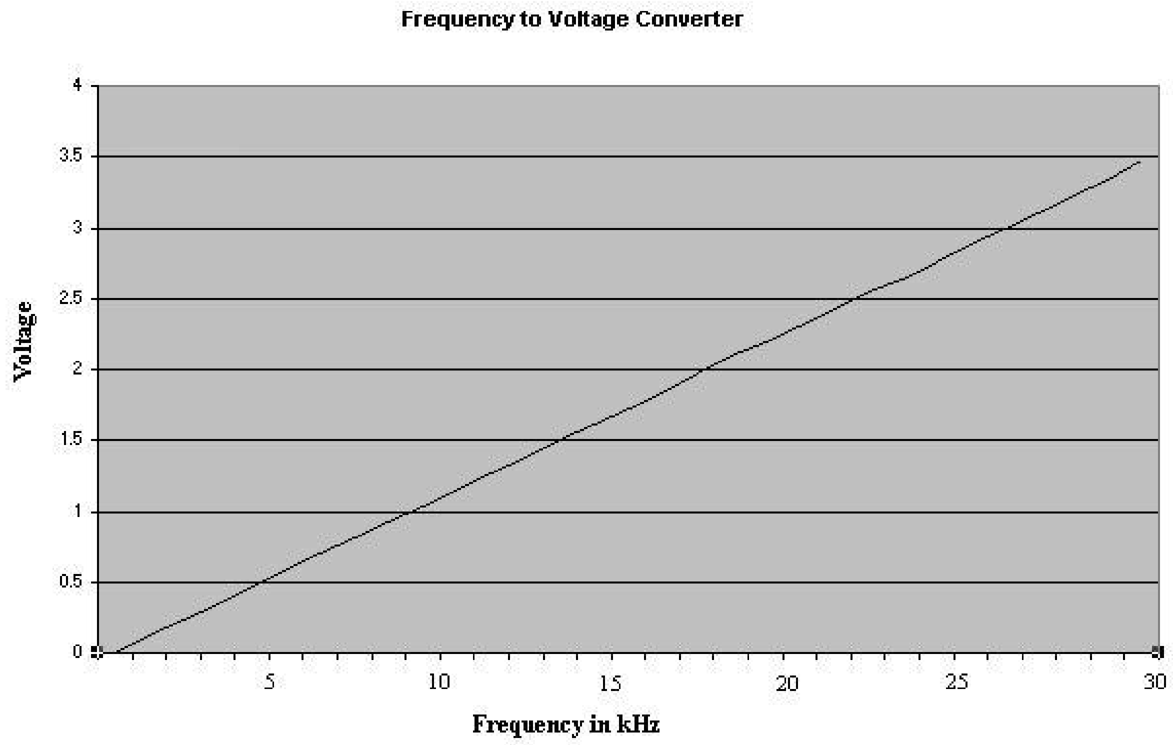


Figure 6 Frequency to Voltage Conversion Chart

In order for the frequency to voltage converter to have the maximum voltage swing over the desired range of frequency we had to choose the capacitors and resistors corresponding to

Vo = Vcc * Fin * C1 * R1

Where Fin is the maximum frequency, C1 is a capacitor, and R1 is a capacitor. With the motors running at full speed, the measured frequency on the encoders was about 30 kHz. The capacitor and resistor values were chosen accordingly.

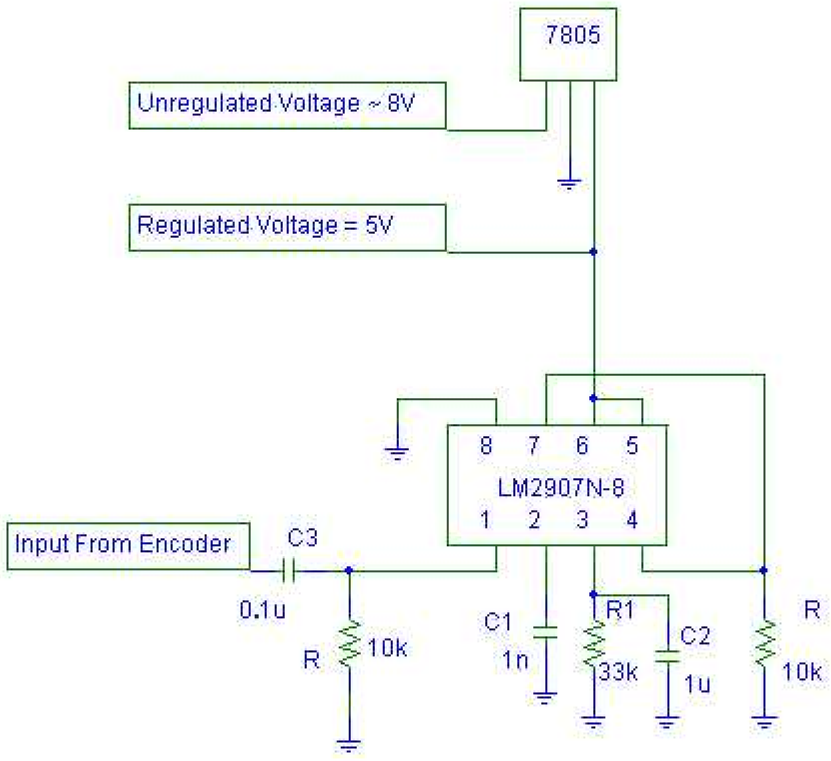


Figure 7 Circuit Design with Frequency to Voltage Converter and 5V Regulator

Flame Sensor

Our flame sensor consisted of two PN168 phototransistors. The phototransistors were concealed in a plastic hood to prevent interference from outside light. We had one phototransistor with a 33k resistor on the emitter for use as an analog fire signal. We found that the 33k resistor provided the most sensitivity to changes in light. The other phototransistor had a 100k resistor on the emitter. We ran the output from this transistor through a Schmitt trigger to get a digital signal.

Originally we were planning to use the analog signal to search the room and see if there was a flame present. If there was a flame in the room we were going to use the digital signal to hone in on the flame. We later simplified the program so that we would only look at the analog signal for flame detection.

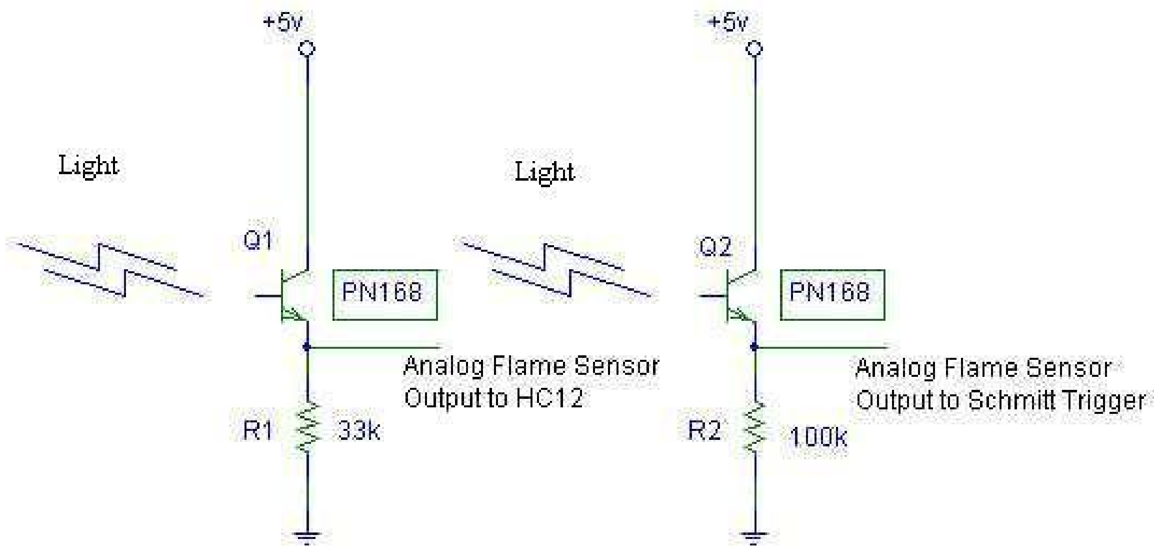


Figure 8 Analog and Digital Components of Flame Sensor

Schmitt Trigger

We used a Schmitt trigger to clean up the signal from the flame sensor and the white line sensor. The Schmitt trigger inverts the signal so we had to account for that change in the program.

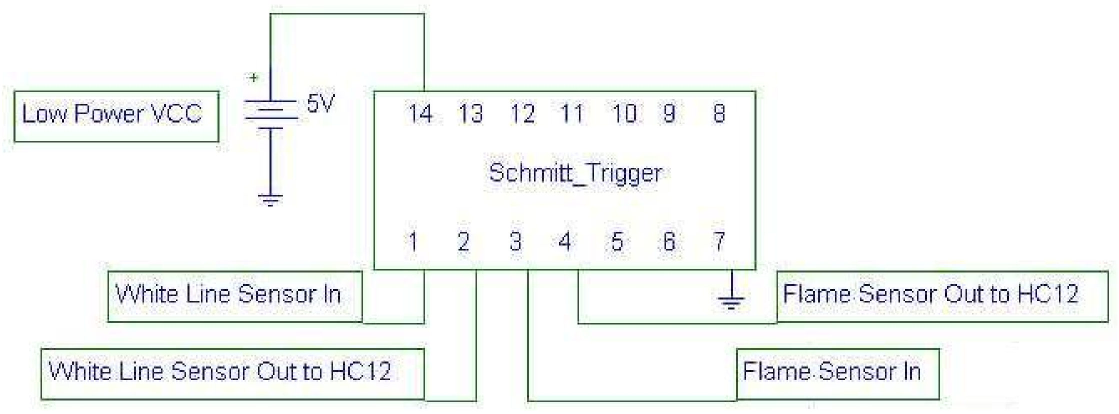


Figure 9 Schmitt Trigger Wiring Diagram

Flame Suppression

To extinguish the flame we used a small motor and a fan, both of which we found in the spare parts bin. The fan was wired to an Antex solid-state relay, which helped isolate the high power and low power supplies. When the HC12 sent the relay a 5V signal the fan would run off of the high power supply and extinguish the flame.

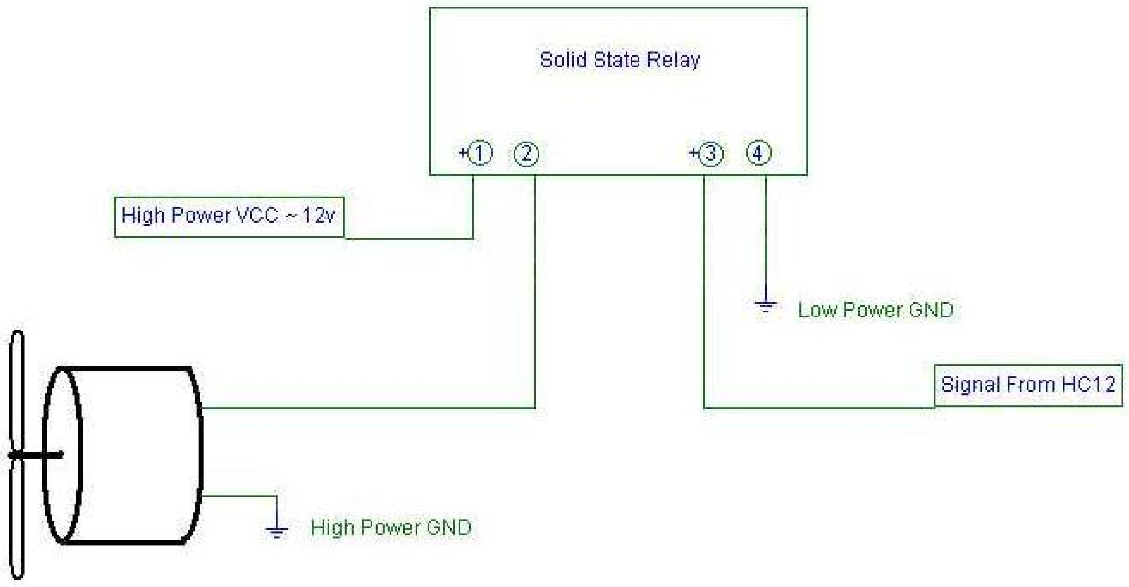


Figure 10 Solid State Relay and Fan Connection Diagram

<u>Optoisolation</u>

Optoisolators between the H-bridge and HC12 helps to insure that the high power does not damage the HC12 in the event of a voltage spike. Optoisolators consist of a light emitting diode (LED) and a phototransistor. A Schmitt trigger is also built into the chip to help with noise immunity. Optoisolation is necessary to protect the H-bridge from sending a signal into the microcontroller and damaging HC12. If a high voltage signal tries to travel back toward HC12, the Optoisolators will prevent this signal from reaching the HC12.

We used four Fairchild H11L1 Schmitt Trigger Optoisolators. They were placed on the PWM lines and the direction lines coming from the HC12. These chips have a 100ns switching time. When wiring up these chips we made sure to isolate the high and low grounds. Because the optoisolators have a built in Schmitt trigger, we never had any issues with signal noise.

Analysis of Power Usage

For the robot we used a high and low power supply. In order to figure out what batteries to use for the robot, we needed to compute the power composition of all of the components. We did this by creating a power budget.

Power Budget

Component	Max. Cont. Current	Quantity	Total Current
Drive Motors	530 mA	2	1060 mA
Fan	150 mA (est.)	1	150 mA
HC12	100 mA	1	100mA
Altera	500 mA	1	500 mA
GP2D120 Sensors	50 mA	2	100 mA
H-Bridge	66 mA	1	66 mA
Optocoupler input LEDs	60 mA	4	240 mA
White Line Sensor	50mA	1	50 mA
Fire Sensor	100 mA	1	100mA
GP2D12 Sensor	35 mA	1	35 mA

Figure 11 Power Budget

We used this power budget chart and divided our items into high and low power items. The high power supply needed to drive the motors, the fan, and the H-bridges. These totaled to a maximum current usage of 1276 mA. The low power side needed to power the HC12, and all of the sensors. The low power side totaled to a maximum current draw of 1125 mA.

Budget

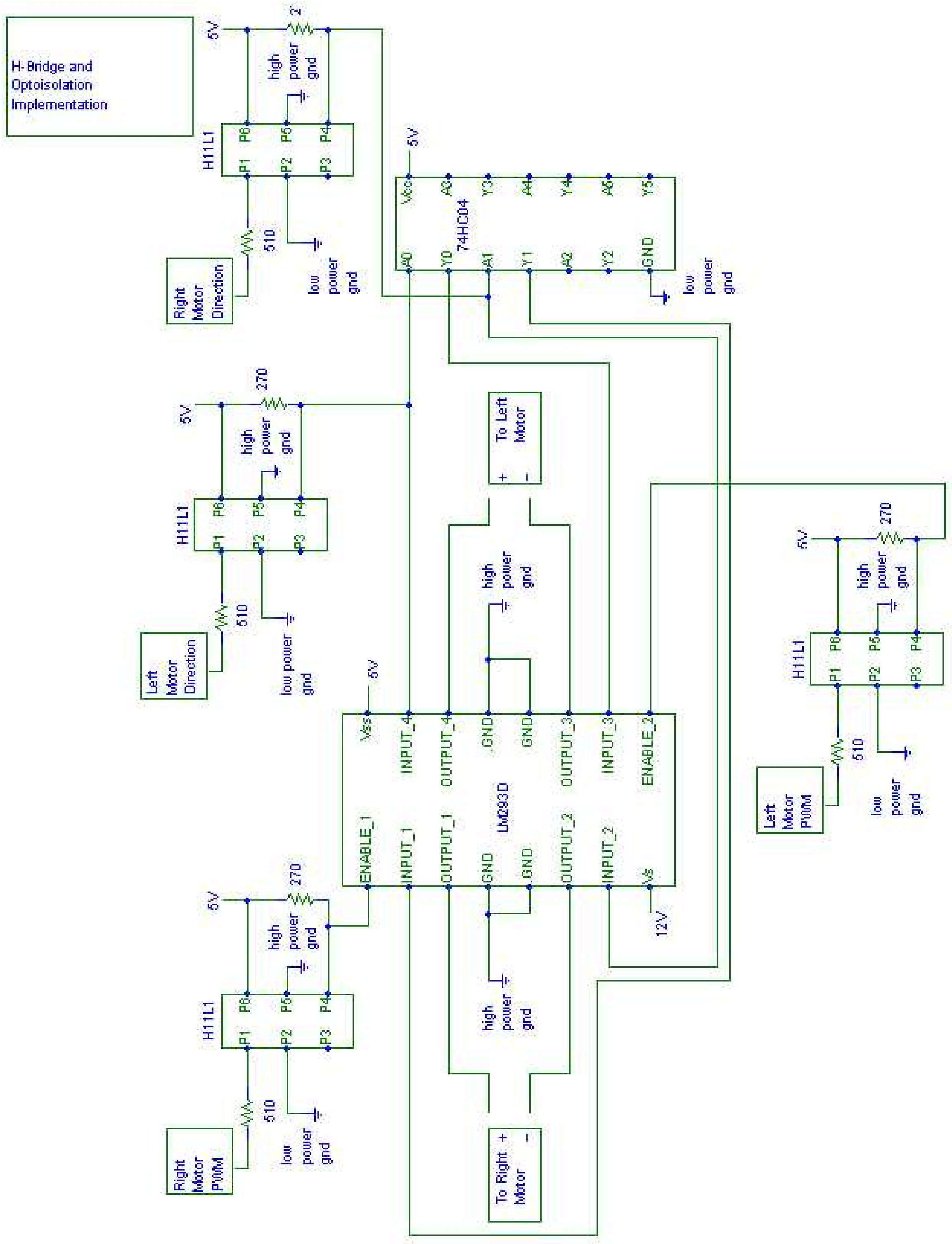
Item Description	price/ea	quantity	cost
H-Bridge (Thompson) L293D	$6.00	1	$6.00
Battery Holder: 8-AA holder	$2.00	2	$4.00
Perf-board 2.5"X3.2":	$2.00	1	$2.00
Perf-board 2.25"X1.8":	$1.25	2	$2.50
Perf-board 1.5"X1.75":	$1.00	1	$1.00
Freq-voltage converters: LM2907 or 17	$2.50	2	$5.00
Schmitt trigger optoisolators	$1.50	4	$6.00
5V regulators: 7805	$0.75	2	$1.50
3/32" shrink tubing (price per inch)	$0.10	40	$4.00
IR distance sensors:0-30cm: GP2D120:	$8.00	2	$16.00
IR distance sensors:0-80cm: GP2D12:	$10.00	1	$10.00
Maxon 6-Watt 22mm motors (green body):	$50.00	2	$100.00
12V 2.0Ahr lead battery:	$30.00	1	$30.00
12V lead-acid chargers:	$9.00	1	$9.00
9-Volt battery connector	$0.50	1	$0.50
Wire-wrap DIP socket2x4-pin:	$1.00	2	$2.00
Wire-wrap DIP socket 2x7-pin:	$1.25	3	$3.75
Wire-wrap DIP socket2x8-pin:	$1.50	2	$3.00
Boxed headers2X5-pin (male)	$1.00	2	$2.00
crimp contacts: (i.e. pins for above)	$0.10	80	$8.00
Antex solid-state opto-isolated2.5A DC relays:	$5.00	1	$5.00
Reflective IR emitter/detector pair:	$1.50	3	$4.50
Misc. capacitors:	$0.10	3	$0.30
Misc. LEDS	$0.10	2	$0.20
Misc. switches:	$0.50	3	$1.50
Misc. spacers/standoffs 4-40 spacers:	$0.10	4	$0.40
Misc. spacers/standoffs 6-32 spacers:	$0.10	8	$0.80
1.5" rubber wheels:	$2.50	2	$5.00
motor screws	$0.30	6	$1.80
NiMH batteries	$2.00	6	$12.00
pin headers	$2.00	2	$4.00
crimp motor connectors	$0.80	4	$3.20
Instrument room supplies(est.)	$10.00	1	$10.00
GRAND TOTAL			$264.95

Figure 12 Final Budget

Conclusion

This project has been a new and useful experience for everyone in Group 3. We are all satisfied with the robot's performance. Despite lagging behind other groups for much of the semester, we were able to accomplish enough in the last few weeks to place a respectable 5th in the New Mexico Tech Firefighting Robot Competition. The robot's primary strengths are its strong chassis and simple design. Its main weakness is the wiring, along with a programming glitch or two. With another month, we could make both the wiring and the program substantially more robust, and it would be interesting to see how we would fare at Trinity. The fact that we did not try to get to Trinity leads to some of the lessons we learned. The primary problems we had during the semester involved communication within the group and division of tasks. When the decision was made as to which groups would go to Trinity, our group was not in the running for the simple reason that lack of organization had put us behind. It is safe to say that in the course of this semester every member in our group has learned something about organization and communication and their importance in a group project. These lessons in working with others, along with the engineering and presentation skills we have gained in this course should prove very useful next year in Senior Design and in our careers beyond graduation.

Appendix A



H-Bridge and Optoisolation Implementation
H11L1
P1
P2
P3
P4
P5
P6
5V
2
270
510
high power gnd
low power gnd
Right Motor Direction
Left Motor Direction
Right Motor PWM
Left Motor PWM
74HC04
Vcc
A0
Y0
A1
Y1
A2
Y2
A3
Y3
A4
Y4
A5
Y5
GND
LM293D
Vss
INPUT_4
OUTPUT_4
GND
OUTPUT_3
INPUT_3
ENABLE_2
ENABLE_1
INPUT_1
OUTPUT_1
OUTPUT_2
INPUT_2
Vs
12V
To Left Motor
To Right Motor
+
−

Appendix B

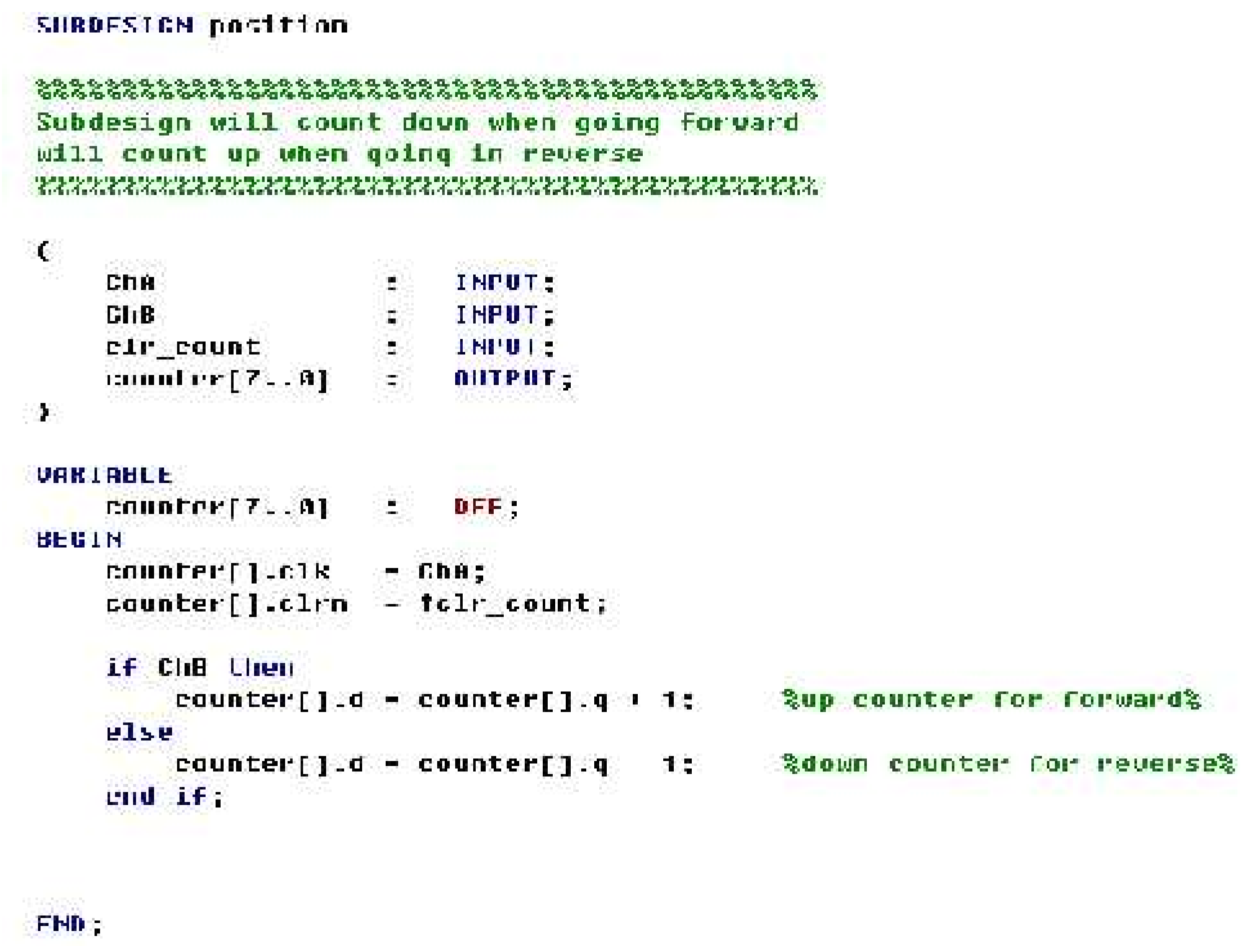
```
#include "hc12.h"
#include "DBug12.h"

// Define macros used in the program. Used to initializes ports to work
// as we desired. Made program more readable and understandable.
#define RIGHTFORWARD  PORTP = PORTP & ~0x40  //sets right wheel forward
#define LEFTFORWARD   PORTP = PORTP | 0x80   //sets left wheel forward
#define RIGHTBACKWARD PORTP = PORTP | 0x40   //sets right wheel
backward
#define LEFTBACKWARD  PORTP = PORTP & ~0x80  //sets left wheel backward
#define NOWHITELINE   PORTS & 0x04           //port set to no whiteline
seen
#define NOFLAME       PORTS & 0x08           //port set to no flame
seen
#define R_MOTOR_SPEED PWDTY3                 //right motor speed
#define L_MOTOR_SPEED PWDTY2                 //left motor speed
#define FAN_ON        PORTP = PORTP | 0x20   //sets port to turn on fan
#define FAN_OFF       PORTP = PORTP & ~0x20  //sets port to turn off
fan
#define START_ROBOT   PORTP & 0x10           //starts the robot with a
switch

//Global variables used through program. All are signed integers to
allow
//negative and positive numbers.

signed int kr1, kr2, kr3, kl1, kl2, kl3; //constants for motors
signed int rmsd, lmsd;                   //desired motor speed
signed int rmsm, lmsm;                   //measured motor speed
signed int rmdir, lmdir;                 //direction of each motor
signed int ldd;                          //desired distance to wall
signed int ldm;                          //measured distance to wall
signed int rmd,lmd;
signed int r_duty_cycle, l_duty_cycle;   //duty cycles for left and
right motors
signed int frontsensor;                  //measurement from front wall
sensor
signed int leftsensor;                   //measurement from left wall
sensor
signed int rightsensor;                  //measurement from right wall
sensor
signed int flamesensor;                  //measurement from flame
sensor
signed int line_count = 0;               //line count of white lines
sensed
signed int x,y, temp;                    //used for delay functions and
                                         //temporary placement of
variables
int rti_test, rmd, lmd;                  //real time interrupt
int line, flame, fan, start;             //variables used to
communicate w/fan
//variables used to troubleshoot the program
signed int volt,fire_line, threshold;
```

```
//start of main program
void main(void)
{
  //set up of port p and port s => initialize the input and output pins
  DDRP = 0xe0;                  //sets port P[7..5] as output, and PP4
as input
  DDRS = 0x00;                  //PS2 and PS3 are inputs

  //set up the pulse width modulation
  PORTP = 0x80;
  PWCLK = PWCLK & ~0xc0;        //8 bit mode
  PWCTL = PWCTL & ~0x08;        //left aligned
  PWPOL = 0x00;                 //polarity->reversed due to the
inverters in
                                //opto-isolators
  PWPOL = PWPOL | 0xc0;         //clock mode 1 for channel 3
  PWCLK = (PWCLK | 0x02);       // N = 2
  PWPER2 = 249;                 //set the period of channel 2
  PWPER3 = 249;                 //set the period of channel 3
  PWEN = PWEN | 0x0c;           //enable PWM channels 2 and 3

  ATDCTL2 = ATDCTL2 | 0x80;     //turn on a/d converter
  ATDCTL4 = 0x01;               //2MHz a/d clock
  ATDCTL5 = 0x70;               //select 8 channel mode and scan

  ldd = 60;                     //set desired distance from left wall
  rti_test = 0;                 //intialize rti_test variable to zero

  kr1=2;                        //initializes right constants
  kr2=1;
  kr3=2;
  kl1=2;                        //initializes left constants
  kl2=1;
  kl3=2;

  rmsd = 80;                    //set desired motor speeds for left and
right
  lmsd = 80;

  frontsensor = ADR5H;          //update the sensor for initial use
  rightsensor = ADR4H;
  leftsensor = ADR3H;
  flamesensor = ADR2H;
  FAN_OFF;                      //initialize fan to be off

  threshold = 50;               //initialize threshold for flame sensor
  RTICTL= 0x84;                 //enable rti and sample every 16ms
  RTIFLG= 0x80;                 //clear rti flag

  x = 0;                        //initialize variable to zero
  while(x == 0)                 //while loop waits for a high on pin 5
of PORT P
    {
      x = START_ROBOT;          //x is update with value on pin5 of
port p
      R_MOTOR_SPEED = 0;        //motor remain off until a one is on
pin 5
```

```
      L_MOTOR_SPEED = 0;
    }

  delay(1000);                  //delay for 1000 ms after the pin
receives a 1
  start_moving();               //calls start moving function
  enable();                     //enables rti interrupt

  //infinite loop to run robot
  while(1)
    {
      rmsm = ADR7H;             //updates all sensor values
      lmsm = ADR6H;
      flamesensor = ADR2H;
      frontsensor = ADR5H;
      leftsensor = ADR3H;
      rightsensor = ADR4H;
      flame = PORTS & 0x08;     //updates the flame port to current
value

      left_wall_follow();     //calls left wall follow function
      check_white_line();       //calls check white line function

      if(frontsensor > 90)      //turns right if wall in front
      {
        right_turn();         //calls right turn function
      }
      LEFTFORWARD;              //turn polarity of wheel both forward
      RIGHTFORWARD;
    }                           //end of infinited loop
}                               //end of main function

/***********************************************************************
********
* Function : void stop(int stop_time)
* This function receives a value from calling function and puts the
robot
* at a stop for the amount of time(in milliseconds) that is specified
***********************************************************************
*******/
void stop(int stop_time)
{
  LEFTBACKWARD;                 //reverses the wheels
  RIGHTBACKWARD;
  L_MOTOR_SPEED = 50;           //sends a PWM to brake the robot
  R_MOTOR_SPEED = 50;
  delay(60);                    //delays for 60 ms to assure stop of
robot
  L_MOTOR_SPEED = 0;            //sends a zero PWM to remain at a stop
  R_MOTOR_SPEED = 0;
  delay(stop_time);             //calls delay function and send the
stop time
}

/***********************************************************************
********
* Function : void right_turn()
* This function provides a right turn for the robot using the
frontsensor and
* left sensor.
**********************************************************************
*******/
void right_turn()
{
  //stop the robot first
  LEFTBACKWARD;
  RIGHTBACKWARD;
  L_MOTOR_SPEED = 200;
  R_MOTOR_SPEED = 200;

  //updates the sensor values
  frontsensor = ADR5H;
  leftsensor = ADR3H;

  //turns until the frontsensor is the desired distance away from the
wall
  while(frontsensor > 75)
    {
      LEFTFORWARD;
      RIGHTBACKWARD;
      L_MOTOR_SPEED = 110;      //provides a constant speed
      R_MOTOR_SPEED = 110;
      frontsensor = ADR5H;      //updates the sensors again
      leftsensor = ADR3H;
    }
}

/*********************************************************************
********
* Function : void right_turn()
* This function provides a right turn for the robot using the
frontsensor and
* left sensor.
**********************************************************************
*******/

void left_turn()
{

  //frontsensor = ADR5H;
  leftsensor = ADR3H;
  delay(100);
  LEFTBACKWARD;
  L_MOTOR_SPEED = 150;
  delay(10);

  while(leftsensor < 65)
    {

      LEFTFORWARD;
      RIGHTFORWARD;
      L_MOTOR_SPEED = 50;
      R_MOTOR_SPEED = 200;
      leftsensor = ADR3H;
      check_white_line();
      //DBug12FNP->printf("stuck in turn :  %d\r\n", leftsensor);

    }
}

/***********************************************************************
********
* Function : void left_wall_follow()
* This function is the left wall following function. It provides the
closed loop
* control and keeps the initial desired distance to the wall. The
function is
* uses three different sets of constants to assure that the robot does
not hit
* the wall when there is a significant difference in the sensor values.
***********************************************************************
*******/

void left_wall_follow()
{

  rmsm = ADR7H;                  //update motor speed and wall distance
  lmsm = ADR6H;
  ldm  = ADR3H;

  if(ldm < 15)                   //executed when robot is too close to
the wall
    {
      kr1=1;                     //sets constants for near low voltages
from
      kr2=2;                     //left wall sensor
      kr3=1;
      kl1=1;
      kl2=1;
      kl3=1;

      ldd = 45;                  //sets the desired distance farther
than it
                                 //should anticipate
      //calculates the motors speed based upon the distance desired and
the
      //distance measured
      lmsd = 120 + (kl3 * (ldd - ldm));
      rmsd = 100 - (kr3 * (ldd - ldm));

      //adjusts the duty cycle if needed based upon the calculation of
the
      //previous equation
      r_duty_cycle = (kr1 * rmsd) + (kr2 * (rmsd - rmsm));
      l_duty_cycle = (kl1 * lmsd) + (kl2 * (lmsd - lmsm));

      //these if statements are included to correct the duty cycle if
it to
      //were ever overflow.
      if(r_duty_cycle < 0)
      {
        r_duty_cycle = 0;
      }
      if(r_duty_cycle > 249)
      {
        r_duty_cycle = 249;
      }
      if(l_duty_cycle < 0)
      {
        l_duty_cycle = 0;
      }
      if(l_duty_cycle > 249)
      {
        l_duty_cycle = 249;
      }

      //the calculated duty cycles are then set to the left and right
motor
      PWDTY3 = r_duty_cycle;
      PWDTY2 = l_duty_cycle;

      //the distance to the wall is re-initialized to the our previous
distance
      ldd = 60;
    }
  else if(ldm < 65)       //executed if robot is in the range of desire
distance
    {
      kr1=1;              //set constants for approriate distance
      kr2=1;
      kr3=2;
      kl1=1;
      kl2=1;
      kl3=2;

      //calculates the motors speed based upon the distance desired and
the
      //distance measured
      lmsd = 225 + (kl3 * (ldd - ldm));
      rmsd = 225 - (kr3 * (ldd - ldm));

      //adjusts the duty cycle if needed based upon the calculation of
the
      //previous equation
      r_duty_cycle = (kr1 * rmsd) + (kr2 * (rmsd - rmsm));
      l_duty_cycle = (kl1 * lmsd) + (kl2 * (lmsd - lmsm));

      //these if statements are included to correct the duty cycle if
it to
      //were ever overflow.
      if(r_duty_cycle < 0)
      {
        r_duty_cycle = 0;
      }
      if(r_duty_cycle > 249)
      {
        r_duty_cycle = 249;
      }
      if(l_duty_cycle < 0)
      {
        l_duty_cycle = 0;
      }
      if(l_duty_cycle > 249)
      {
        l_duty_cycle = 249;
      }

      //the calculated duty cycles are then set to the left and right
motor
      PWDTY3 = r_duty_cycle;
      PWDTY2 = l_duty_cycle;
    }
  else                    //robot is too far away from wall
    {
      kr1=1;              //set constants to compensate for robot being
too
      kr2=2;              //far from wall
      kr3=2;
      kl1=1;
      kl2=2;
      kl3=2;

      //calculates the motors speed based upon the distance desired and
the
      //distance measured
      lmsd = 150 + (kl3 * (ldd - ldm));
      rmsd = 150 - (kr3 * (ldd - ldm));

      //adjusts the duty cycle if needed based upon the calculation of
the
      //previous equation
      r_duty_cycle = (kr1 * rmsd) + (kr2 * (rmsd - rmsm));
      l_duty_cycle = (kl1 * lmsd) + (kl2 * (lmsd - lmsm));

      //these if statements are included to correct the duty cycle if
it to
      //were ever overflow.
      if(r_duty_cycle < 0)
      {
        r_duty_cycle = 0;
      }
      if(r_duty_cycle > 249)
      {
        r_duty_cycle = 249;
      }
      if(l_duty_cycle < 0)
      {
        l_duty_cycle = 0;
      }
      if(l_duty_cycle > 249)
      {
        l_duty_cycle = 249;
      }
      //the calculated duty cycles are then set to the left and right
motor
      PWDTY3 = r_duty_cycle;
      PWDTY2 = l_duty_cycle;
      rmsm = ADR7H;            //update measurements again
      lmsm = ADR6H;
      ldm  = ADR3H;
    }
}

/*********************************************************************
********
* Function : void candle_white_line()
* This function looks for the white line around the candle after the
flamesensor
* senses the candle. We use it so our white line count is not
incremented.
**********************************************************************
*******/
void candle_white_line()
{
  fire_line = 0;               //line around candle not found
  if (rti_test == 1)           //checks rti interrupt
    {
      fire_line = 1;           //line around candle found
    }
}

/*********************************************************************
********
* Function : check_white_line()
* This function checks for the white lines that are in the entrance of
the rooms.
* If the line_count is either a 0,2,4,or 7 it will initiate a room
search and
* will precede on from there.
**********************************************************************
*******/
void check_white_line()
{
  if (rti_test == 1)          //if there is a white line
    {
      //executed if the line_count is 0,2,4 or 7
      if(line_count == 0 | line_count == 2 | line_count == 4 |
line_count == 7)
      {
        delay(200);           //provide delay to make robot roll over
white line
        stop(100);
        room_search_lwf();
      }
      if(line_count == 6)
        {
        stop(200);
        while(frontsensor > 85)
          {
            stop(200);
            left_wall_follow();
            frontsensor = ADR5H;
          }
        flip_turn();
        }
      else
      {
        delay(100);
        stop(100);
      }
      line_count = line_count + 1;

    }
  else
    {
    }
}
/*********************************************************************
********
* Function : void flip_turn()
*This function turns the robot around until the right sensor sees the
wall, then
* moves forward.
*********************************************************************
*******/
int flip_turn()
{
  rightsensor = ADR4H;
  while(rightsensor < 50)
    {
      RIGHTBACKWARD;
      LEFTFORWARD;
      L_MOTOR_SPEED = 90;
      R_MOTOR_SPEED = 90;
      rightsensor = ADR4H;
    }
  RIGHTFORWARD;
  LEFTFORWARD;
  return (1);
}

/*********************************************************************
********
* Function : void delay(unsigned int ms)
*Provides a delay
*
*********************************************************************
*******/
void delay(unsigned int ms)
{
  int i;
  while(ms>0)
    {
      i = 2000;
      while (i > 0)
        {
        i = i - 1;
        }
      ms = ms-1;
    }
}

/*********************************************************************
********
* Function : void room_search_lwf()
*This function goes into the room and then scans until either the flame
sensor
*sees a flame or the front sensor sees the wall
*********************************************************************
*******/
void room_search_lwf()
{
  int y, z;   /* set variables to zero */
  flamesensor = ADR2H;
  frontsensor = ADR5H;

  y = 0;
  z = 0;
  flame = 0;

  while(y < 100)    /* scan right for about 400ms */
    {

      frontsensor = ADR5H;
      flamesensor = ADR2H;
      RIGHTBACKWARD;
      LEFTFORWARD;
      L_MOTOR_SPEED = 80;
      R_MOTOR_SPEED = 80;
      if(frontsensor > 80)
      {
        y = 100;
      }
      if (flamesensor > 55)  /* if flame is seen, set variables to jump
out of while loops */
      {
        RIGHTFORWARD;
        LEFTBACKWARD;
        L_MOTOR_SPEED = 130;
        R_MOTOR_SPEED = 130;
        delay(10);
        y = 100;
        z = 100;
        flame=1;
      }
    }
  while(z < 100)     /* scan left for about 800ms */
    {
      frontsensor = ADR5H;
      flamesensor = ADR2H;
      RIGHTFORWARD;
      LEFTBACKWARD;
      L_MOTOR_SPEED = 80;
      R_MOTOR_SPEED = 80;
      if(frontsensor > 80)
      {
        z = 100;
      }
      if(flamesensor > 55)
      {
        z = 100;
        RIGHTBACKWARD;
        LEFTFORWARD;
        L_MOTOR_SPEED = 130;
        R_MOTOR_SPEED = 130;
        delay(10);
        flame = 1;
      }
    }


  if(flame == 1)
    {
      flame_extinguish();
    }
  stop(50);
}

/*********************************************************************
********
* Function : void flame_extinguish()
* Robot follows left wall until white line is seen.  Fan is turned on
and
* flame is extinguished
*********************************************************************
*******/
void flame_extinguish()
{
  y = 0;
  leftsensor = ADR3H;
  frontsensor = ADR5H;
  candle_white_line();
  temp = fire_line;

  LEFTFORWARD;
  RIGHTFORWARD;
  left_wall_follow();

  while(temp == 0)
    {
      left_wall_follow();
      candle_white_line();
      temp = fire_line;
      if(frontsensor > 90)
      {
        right_turn();
        LEFTFORWARD;
        RIGHTFORWARD;
      }
      frontsensor = ADR5H;
    }
  stop(500);
  FAN_ON;
  delay(2500);
  FAN_OFF;
  stop(20000);
}

/**********************************************************************
********
* Function : void start_moving()
* Gives robot a boost to get off of the home circle so that no white
line
* will be counted until first room is found
**********************************************************************
*******/
void start_moving()
{
  RIGHTFORWARD;
  LEFTFORWARD;
  R_MOTOR_SPEED = 150;
  L_MOTOR_SPEED = 150;
  delay(200);
}

/**********************************************************************
********
* Function : @interrupt void rti(void)
* Interrupt for finding white line
*
**********************************************************************
*******/

@interrupt void rti(void)
{
  line = NOWHITELINE;
  if (line == 0)
    {
      rti_test = 1;
    }
  else
    {
      rti_test = 0;
    }
  RTIFLG= 0x80; /*clear rti flag*/
}
```

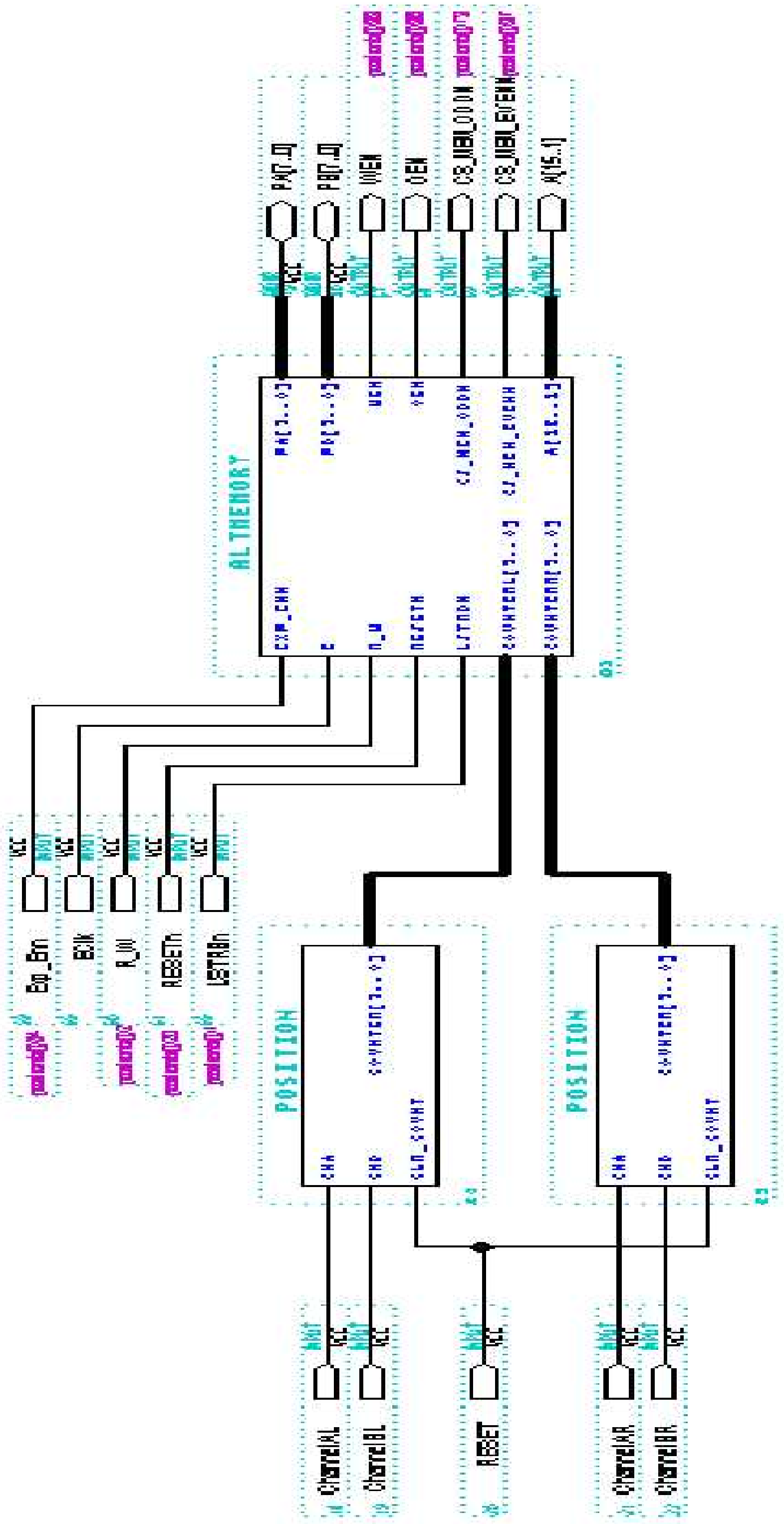
Appendix C

ALTERA is one possible way to determine motor position. To determine the motor position we could have used a basic 8 bit up down counter. The inputs to this up-down counter would come from the encoders on the motors (channel A and channel B). If channel A led channel B then the counter would count up. If channel A lagged channel B then counter would count down. Here is the basic program to implement the up-down counter.



```
SUBDESIGN position

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Subdesign will count down when going forward
will count up when going in reverse
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

(
     ChA                 :    INPUT;
     ChB                 :    INPUT;
     clr_count           :    INPUT;
     counter[7..0]       :    OUTPUT;
)

VARIABLE
     counter[7..0]       :    DFF;
BEGIN
     counter[].clk    = ChA;
     counter[].clrn   = !clr_count;

     if ChB then
          counter[].d = counter[].q + 1;     %up counter for forward%
     else
          counter[].d = counter[].q - 1;     %down counter for reverse%
     end if;


END;
```

This sub design was then created into a symbol and implemented with the memory expansion. The following page provides a complete schematic of the GDF implementation of the motor position.



Our final program did not use any approach for determining position.

References

Fairchild Semiconductors Website (2001)

http://www.fairchildsemi.com

Flynn, A., & Jones, J., & Seiger, B. (1999). *Mobile Robots Inspiration to Implementation*. Massachusetts: A K Peters.

New Mexico Tech EE 382 Course Website (2001)

http://www.ee.nmt.edu/~wedeward/EE382/SP01/ee382.html