STEVE

(Speed Trap Enforcement VehiclE)

Michael Hattermann
Final Report
EEL 5666 – Intelligent Machines Design Lab
Spring 2001
April 23, 2001

Table of Contents

Abstract

The purpose of this project is to construct an intelligent mobile robot to enforce a speed trap. This Speed Trap Enforcement VehiclE (STEVE) will sit in place and wait for an object to pass. When an object goes by, STEVE will calculate the speed the object was traveling. If the object was exceeding a predetermined speed limit, STEVE will turn on his lights and siren and will begin to follow the speeding object. If STEVE ever loses sight of the speeder, the siren will turn off and STEVE will perform object avoidance. STEVE will attempt to "pull over" the speeding object by bumping it from behind. Finally, if STEVE is able to pull the object over, he will signal success and then will shutdown, turning off his lights and siren, and will wait to be placed back in the speed trap.

Executive Summary

STEVE is an autonomous speed trap enforcement vehicle. He is designed to work in a very simple, specially designed speed trap. He must be placed in the speed trap to begin. STEVE will wait there until a speeder is detected using two laser break-beam sensors. Then STEVE will turn on his lights and siren and attempt to follow the speeding object. If STEVE ever loses the speeder, he will turn off his siren and avoid obstacles until he finds the speeder again. STEVE will then pull over the speeding object by bumping it.

STEVE's brain, which is a Motorola HC12 mounted and a board custom designed for EEL4744, is powered by eight 1.2V Ni-Cd batteries. Two output ports and 24k of SRAM have been added to the board. STEVE moves along using two DC gear head motors, which are powered by 12 1.2V Ni-Cd batteries, and a castor wheel for balance. The power supply and the control lines for the motors are completely isolated from the power supply of the electronics. All of this is mounted in a custom platform made of balsa wood and designed and painted to look like a real police car.

STEVE uses a specially designed speed trap sensor, consisting of two laser pointers, two phototransistors, two IR LED's, and two IR detectors modulated at 56.5kHz, to determine the speed of a passing object. STEVE uses three Sharp GP2D12 analog IR rangefinders, modulated at 40kHz, and 7 bump switches to avoid obstacles. STEVE also uses three hacked LiteOn analog IR detectors to follow an object. STEVE is equipped with red and blue flashing lights and a siren for feedback.

Introduction

Do you ever think cops are wasting time and money by just sitting on the side of the road waiting to catch speeders? Don't you think they should be off trying to catch bad guys? Well, now they can. STEVE was designed to pull over speeders autonomously and automatically. He won't be sympathetic to girls crying or sob stories from drivers who just don't want a ticket. STEVE can be much more efficient than your typical cop. Now real cops can do more important things like catching murders and drug dealers.

Integrated System

STEVE's brain is a Motorola HC12 microprocessor, running at 4Mhz, which is surface mounted on a custom board designed by Scott Kanowitz and Patrick O'Malley for EEL4744. The HC12 has four pulse width modulation (PWM) channels, eight analog-to-digital (A/D) channels and eight input capture/output compare channels. The board contains an Altera CPLD, which is used for memory expansion, 8k of EEPROM with a custom monitor program (used for downloading and debugging code) and an RS232 serial interface that connects to a computer through a 9-pin serial cable. The board also contains a large prototyping area (roughly 3" x 8"). A picture of the board can be seen in Figure 1.

A memory expansion was built in the prototyping area consisting of 24k of memory mapped SRAM (although only about 8k of it is usable due to unexplained technical difficulties). There are also two 8-bit output ports. The first one is constructed using two Fairchild DM9368 chips connected to a dual 7-segment display and is used to display the speed of objects passing through the speed trap. The other is consists of a 74HC574 setup to drive 8 LED's (which will be the flashing lights on top of the car).

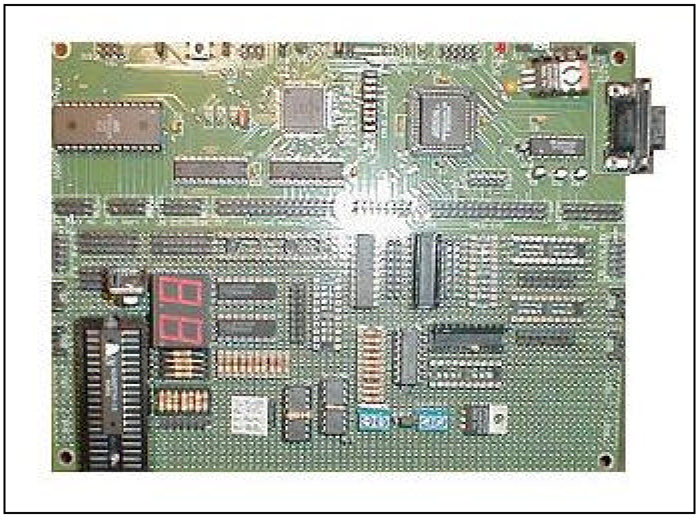

Figure 1

There is a low battery indicator circuit built onto the board that will light up when the battery voltage drops below 7.8V. The circuit schematics can be found in Figure 2.

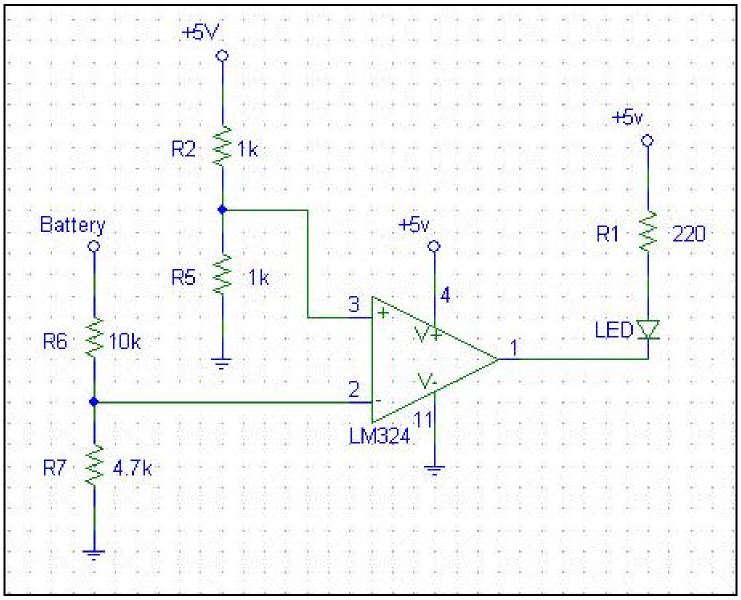

Figure 2

Finally, there are four 4N25 optoisolators used to optically isolate the motors from the electronics. There is a direction and a PWM signal for both the left and right motors. All four optoisolators are wiring in exactly the same way, except for the input signal. The schematic for one optoisolator is found in Figure 3.

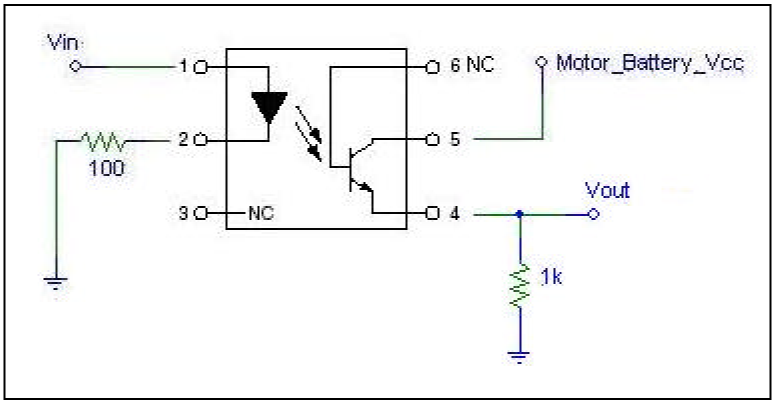

Figure 3

STEVE utilizes an off-board speed tracking system (which will be discussed in detail later) to calculate the speed of a passing objects. He uses two LiteOn digital IR receivers modulated at 56.5 kHz to communicate with the off-board speed tracking system. He uses 7 bump switches and 3 Sharp GP2D12 analog range finders to perform object avoidance. Finally he uses 3 analog hacked LiteOn IR receivers modulated at 56.5 kHz to follow the speeding object.

Mobile Platform

The mobile platform for the robot was custom built out of 1/8 inch thick balsa wood. The platform was designed using AutoCAD and cut out using a T-Tech machine in lab. There were two primary design goals for the platform: 1) The platform had to be large enough to mount the electronics board and all of the sensors and 2) The platform had to look similar to a police car.

The electronics board is fairly large (6” x 8”). This is the main reason the platform is fairly large. The wheels were placed toward the back of the robot to make it look more like a car. There is also a second layer to the platform, like the roof of a car, where the lights and siren are mounted. This second layer also helps to hide the wires and electronics. Finally, the platform was painted to resemble a University of Florida Police Department (UPD) car. A picture of the completed platform can be found in Figure 4.

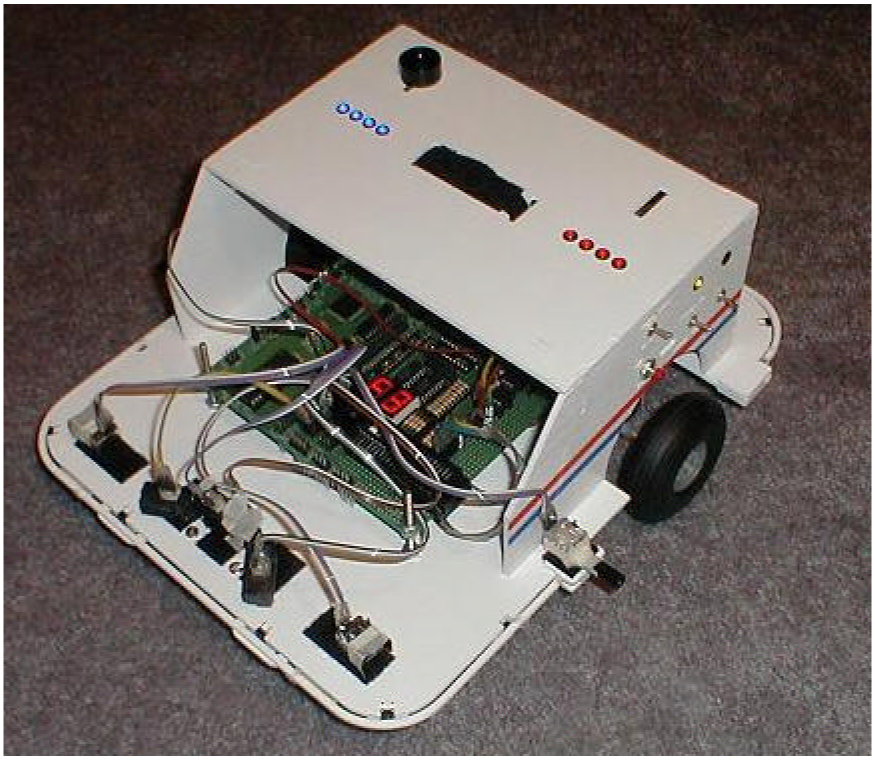

Figure 4

Actuation

STEVE needs to move fairly quickly if he is to catch speeding robots. To get the desired speed, STEVE moves using two DC gear-head motors. They have a no load speed of about 70 rpm and can deliver around 78 oz-in of torque. They each draw about 220mA of current with no load and about 1.1 amps stall current. The motors are driven by two National Semiconductor LMD18200T H-bridges located on a motor driver board originally designed for the OO-PIC, but still works for this application. The schematic for the relevant parts of the board can be found in Figure 5.

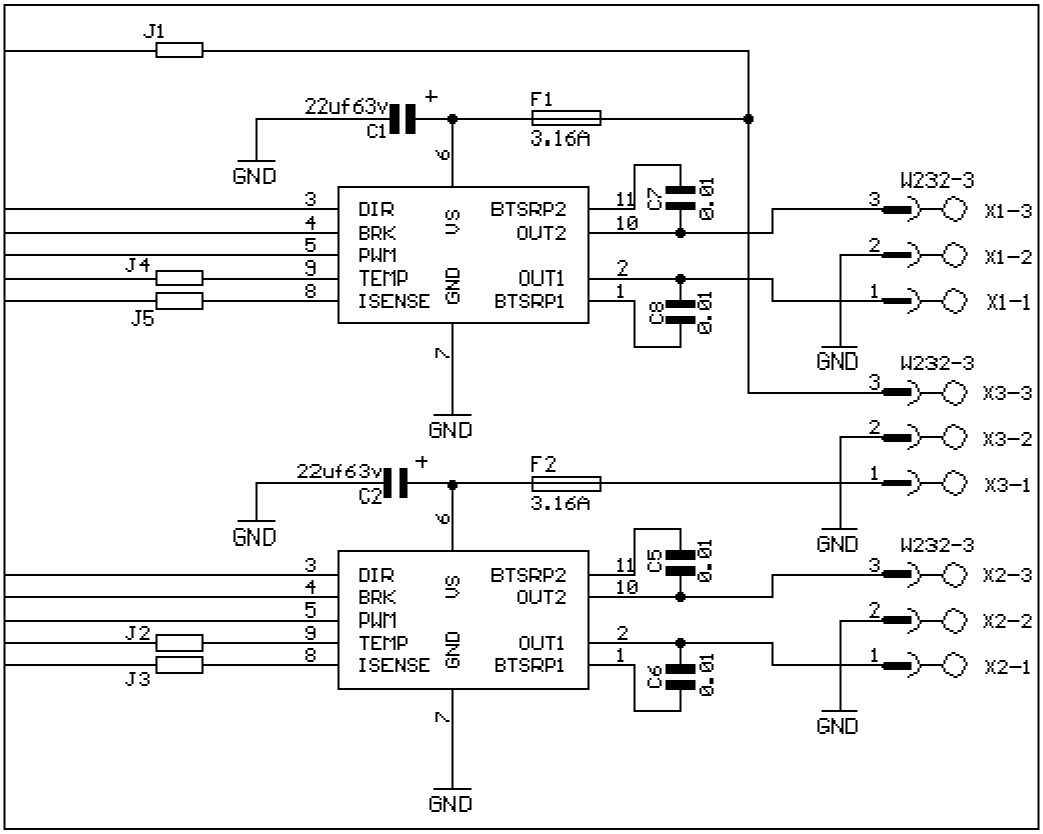

Figure 5

The H-bridges can deliver up to 3 amps of continuous current and 6 amps peak current. There are also two 3-amp fuses located on the board to ensure that the chips are not blown. Each chip uses three control signals to control the motor: a) a PWM signal to control speed b) a direction signal to indicate forward or backward and c) a brake signal. The brake has been hard wired to be off for both motors.

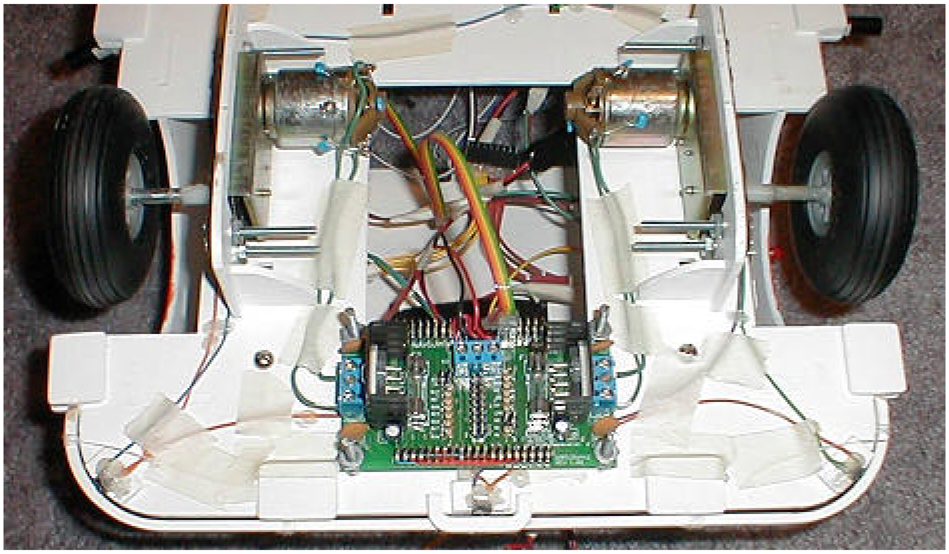

Figure 6

The motors and the motor driving board are both mounted on the bottom of the platform. Figure 6 contains a picture of both. A 3-inch wheel is attached to each motor. This was done by first super gluing roll pins into the center axis of each wheel and then using JB Weld to cold weld the roll pins to the motors. There is a castor wheel attached to the front of the platform to balance the robot and allow free movement.

The motor control software is fairly complicated. The motor speed and direction values are updated every 65ms by an RTI interrupt using the following formula:

$$NewSpeed = \frac{k * OldSpeed + DesiredSpeed}{k + 1}$$

The value chosen for was 6. Then whenever a behavior wants to change the motor speeds, it just has to write to the DesiredSpeed (called NLEFTSPD and NRIGHTSPD in the code, for the left and right motors respectively) global variable, and the motors will be updated automatically.

There is also a function that provides the behaviors with basic control maneuvers. The maneuvers provided are: 1) go forward, 2) go backward, 3) hard left, 4) hard right, 5) soft left, 6) soft right, 7) stop, 8) backup left and 9) backup right. This function will set the motors to perform these maneuvers at whatever the current maximum speed is.

Sensors

Sharp GP2D12 Analog IR Sensors

Three Sharp GP2D12 analog range-finding sensors are used to perform obstacle avoidance. These sensors continuously emit a very narrow beam of IR, modulated at 40kHz. They also continuously check for the amount of IR returned to the sensor using an IR detector. By the amount of IR returned, you can get a fairly accurate reading as to how far away something is. This distance is returned as an analog value.

A test was conducted to find the values returned for an object being a certain distance away. The sensor was set in a stationary position, and an object was moved farther and farther away, taking readings along the way. Also, the width of the beam at each position was measured. Table 1 contains the data from the test.

Distance	Reading 1	Reading 2	Reading 3	Width of beam
4mm	38	35	37	-
8mm	55	55	54	-
12mm	6B	6C	6B	-
16mm	87	87	86	.55mm
20mm	78	7A	79	.6mm
24mm	66	66	66	.6mm
28mm	59	59	59	.6mm
32mm	4F	4F	50	.6mm
36mm	47	46	47	.6mm

40mm	40	40	40	.6mm
44mm	3B	3C	3B	.6mm

Table 1

The test shows that the measurements obtained from the sensor are very linear when the object 16mm (about 3 inches) away or farther. Also, it is important to notice that the width of the beam is narrow (about 1/4 inch).

<u>Bump Switches</u>

Seven bump switches are used to detect if STEVE has collided with an object. These switches have two purposes: 1) redundancy in obstacle avoidance and 2) to tell when a speeding object has been pulled over. If the GP2D12 analog IR sensors fail to find an obstacle, the bump switches will tell STEVE that he has hit something. Then STEVE can take the proper corrective action. Also, STEVE pulls over a speeding object by bumping it from behind. The bump switches will also let STEVE know when this has been accomplished.

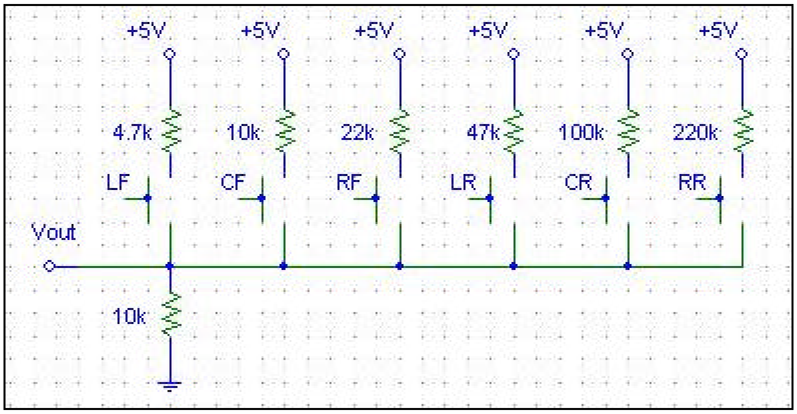

Figure 7

Figure 7 contains the circuit diagram of the bump sensor network. All of the bump switches are wired together through a resistor network and are connected to an analog port on the HC12.

When a bump switch (or any combination of bump switches) is pressed, a unique value will be read from the analog port. A test was conducted to determine the values for each combination of switches. Table 2 contains the data from that test.

Bumper(s)	**Resistance**	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**
LF	4.7k	AE	AE	AF	AE	AF
CF	10k	82	82	82	82	82
RF	22k	50	50	50	50	50
LF+CF	3.2k	C3	C3	C3	C3	C4
RF+CF	6.9k	99	99	9A	9A	9A
LF+RF	3.9k	BA	BA	B9	B9	B9
LF+CF+RF	2.8k	CA	CA	CA	CA	CA
LR	47k	2C	2C	2C	2C	2C
CR	100k	17	17	17	17	17
RR	220k	0B	0B	0B	0B	0B
LR+CR	32k	3C	3C	3D	3C	3C
RR+CR	68.8k	20	20	20	20	21
LR+RR	38.7k	34	34	34	34	34
LR+CR+RR	27.9k	43	43	43	43	43

Table 2

Speed Trap Sensor

STEVE determines the speed of passing objects using a special speed trap sensor. This sensor operates much like a real speed trap. In a real speed trap, there are two white lines painted on the road. The police then fly a plane above the road and time cars as they go by. They start the timer when the car crosses the first line and stop it when it crosses the second line. Then, by knowing the distance between the lines and the time it took to travel that distance, the average velocity of the car is calculated. If the car is speeding, the plane will radio down to police officers waiting farther down the road and they will pull over the car.

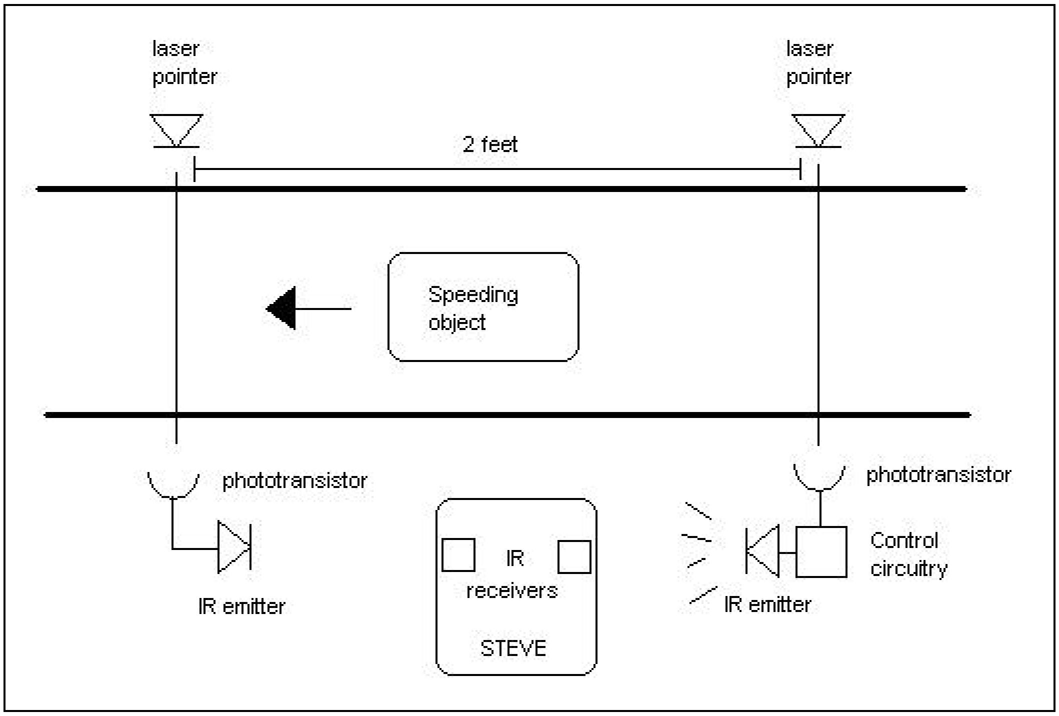

Figure 8

STEVE's speed trap operates in the same way. Figure 8 contains an overhead diagram of the speed trap. The white lines of the speed trap are created using two laser break beams. When a beam is broken, the control circuitry will "radio" to STEVE using an IR emitter modulated at 56.5kHz that the beam is broken. STEVE then records the time the beam was broken. Once both beams are broken, STEVE calculates the time it took the object to travel from one beam to the other. Then, knowing the beams are two feet apart, STEVE calculates the average velocity of the moving object and determines if it was exceeding some predetermined speed limit. If the object is speeding, STEVE turns on his lights and siren and tries to pull it over. If the object is not speeding, STEVE lets it go and continues to wait for a speeding object.

The control circuitry for the break beam sensor is very simple. It consists of three parts: a timer to generate a 56.5kHz waveform and two control circuits to monitor the break beam. The circuit diagram of the timer can be found in Figure 9.

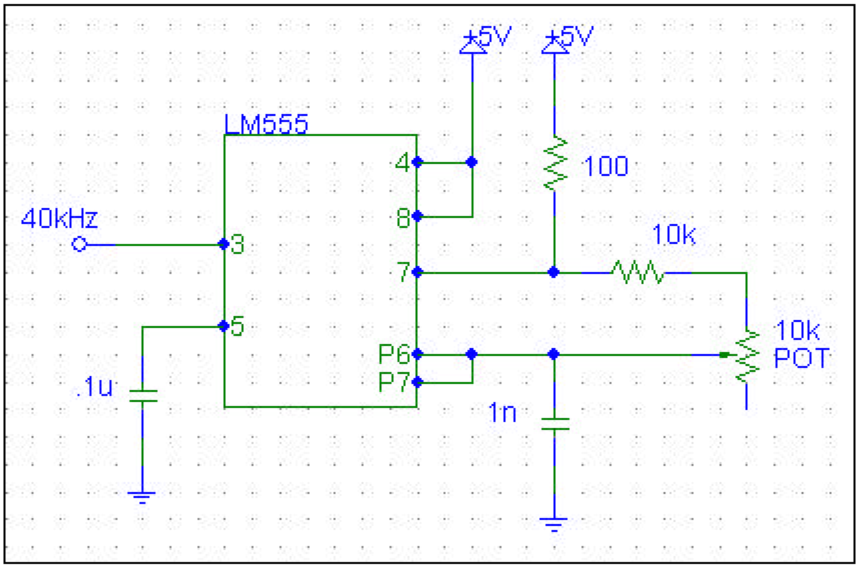

Figure 9

A 555 timer chip was used to generate the 56.5kHz signal needed to modulate the IR emitter. The timer is running in the astable mode of operation and has a duty cycle of 48.4%. The potentiometer can be used to change the frequency from 56.7kHz down to 29.9kHz. This allows IR detectors at different frequencies to be used if there are interference problems.

There are two identical control circuits to control the break beam sensors. The circuit diagram for one control circuit can be found in Figure 10. The LM358 op amp is used to convert the signal from the phototransistor to a digital signal. This is signal is then passed through a 74'02 which AND's it with the 56.5kHz signal generated from the timer circuit. This will generate a 56.5kHz signal at the output of the 74'02 whenever the beam is broken.

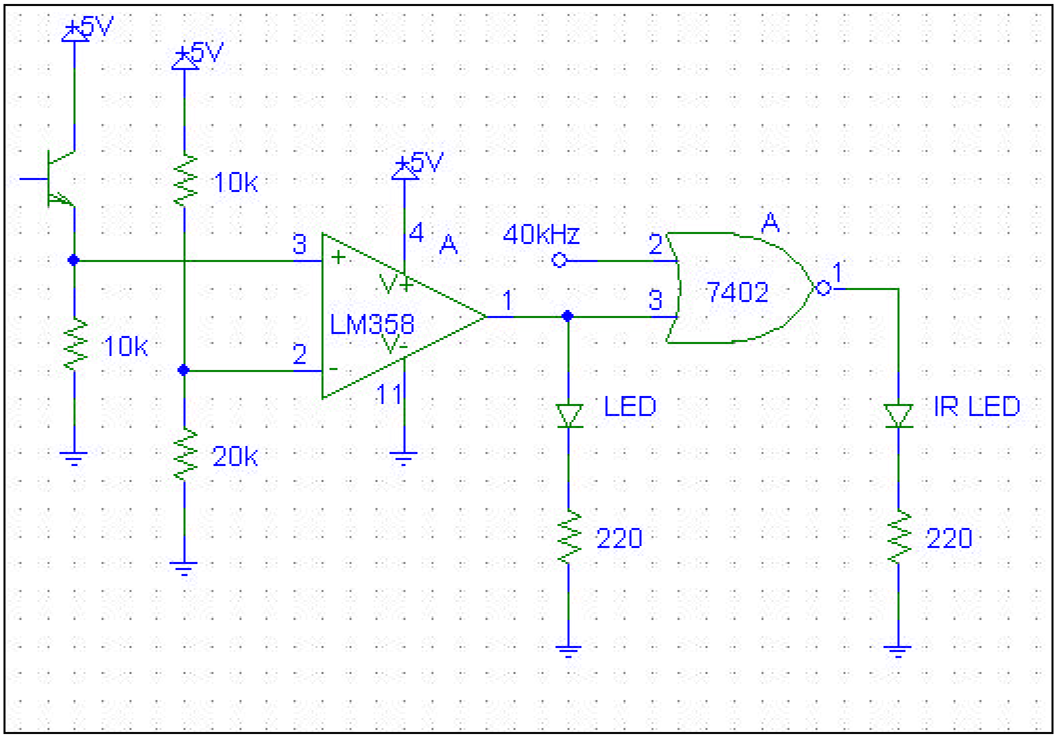

Figure 10

This signal is then fed through the IR emitter to “radio” to STEVE that the beam has been broken. There is also a feedback LED to indicate when the beam is connected to help in the setup of the speed trap.

An experiment was designed to find the maximum distance between the laser pointer and the phototransistor. A photo transistor was set in place and a laser pointer was set up in front of it. Then the laser pointer was moved back gradually, recording the distance the status of the beam at each step. The results of this experiment can be found in Table 3.

Distance(feet)	Test 1	Test 2	Test 3
½	Yes	Yes	Yes
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	Yes	Yes	Yes

Table 3

The results of the experiment show that the laser can be placed very far from the phototransistor and the break beam will still work. This experiment was limited by the length of wire I had to connect to the laser pointer. Also, the farther away from the phototransistor the laser pointer was moved, the more difficult it was to keep the laser pointer aimed at the phototransistor. This was mainly due to the poor design of the laser pointer holders. However, for the intended use of the sensor, three feet of separation is more than sufficient.

An experiment was designed to see how the break beam sensor worked under different lighting conditions. It is difficult to measure the actual amount of light in a room, so a rough scale was made ranging from dark to bright. Then, as the amount of light in the room was changed, the connectedness of the beam was recorded. Table 4 contains the results from the experiment.

Brightness	Test 1	Test 2	Test 3
Dark	Yes	Yes	Yes
Below Average	Yes	Yes	Yes
Average	Yes	Yes	Yes
Above Average	Yes	Yes	Yes
Bright	Yes	Yes	Yes
Very Bright	No	No	No

Table 4

Under the most extreme lighting conditions (i.e. a 60 Watt light bulb being shined from 1-2 feet away), the break beam sensor failed. Under all other conditions, the beam was working. These results are acceptable, because the break beam is very insensitive to ambient light. This is more than sufficient because STEVE will always run indoors.

An experiment was designed to test the maximum distance the IR emitter can communicate with STEVE. A break beam sensor and IR emitter were set in place, and STEVE was move progressively farther away. At each step, the break beam was broken and STEVE was checked to see if he received the communication. Table 3 contains the data from the experiment.

Distance(inches)	**Test 1**	**Test 2**	**Test 3**
2	Yes	Yes	Yes
4	Yes	Yes	Yes
6	Yes	Yes	Yes
8	Yes	Yes	Yes
10	Yes	Yes	No
12	No	No	No
14	No	No	No

Table 5

The results of the experiment show that STEVE can be up to 8 inches away from the IR emitter and safely receive the broken beam signal. Any farther away from that, and he is not guaranteed to accurately receive the signal. For the intended use of the sensor, 8 inches is more than enough distance to safely communicate with STEVE.

Hacked LiteOn Analog IR Sensors

Three analog IR sensors are needed to perform object following. These sensors also had to be at a different frequency than the obstacle avoidance IR sensors. The only Sharp cans I found were at 38kHz or 40kHz, which would interfere with the obstacle avoidance sensors. So I purchased several LiteOn digital IR receivers modulated for 56.5kHz and set out to hack them to be analog. After several hours and a lot of probing, I found the hack. It is pictured in Figure 11.

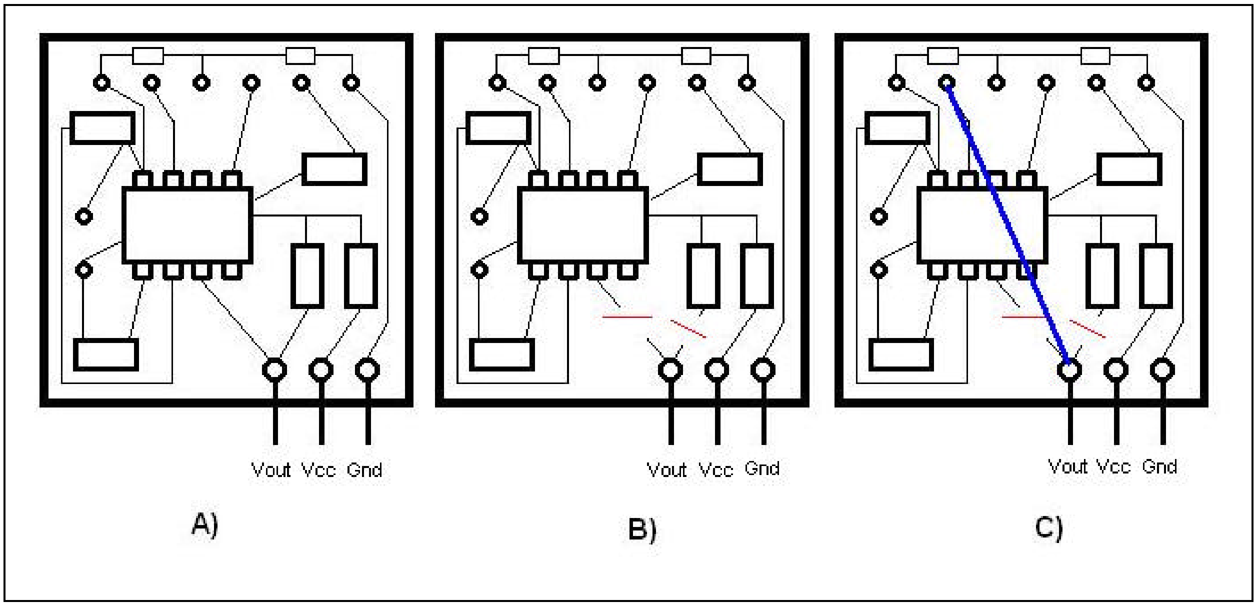

Figure 11

The first step is to open the can to get access to the electronics inside. When the can is open, the electronics board inside should look like Figure 11A. There is on main black chip with eight pins, several surface mount resistors, the bottom of solders for components on the other side of the board, and traces connecting them. The next step is to cut the two traces that connect to the output pin. This is shown in Figure 11B. The final step is to solder a jumper (i.e. a piece of wire wrap wire) between the output pin and the bottom of a solder, as shown in Figure 11C.

An experiment was designed to test the range of the hacked LiteOn IR receiver. The receiver has held stationary and an IR LED was moved incrementally farther away. Several readings were taken from the sensor a each step along the way. The results of the test can be found in Table 4. It was found that the voltage on the out put pin varied from 1.575V when there was no IR to 2.530V when the IR LED was less than one inch from receiver.

Distance	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**
2”	7F	7F	7F	7F	7F
4”	7F	80	80	7F	80
6”	7D	7D	7D	7D	7D
8”	76	76	76	76	76
10”	73	73	73	73	73
12”	6F	6F	6E	6F	6F
14”	6A	6A	6A	6A	6A
16”	67	67	67	67	67
18”	63	63	63	63	63
20”	5F	5F	5F	5F	5E
22”	5C	5C	5C	5C	5C
24”	5A	5A	5A	5A	5A
26”	57	57	57	57	57
28”	55	55	55	55	55
30”	54	54	54	54	54
32”	53	53	53	53	53
34”	53	53	53	53	53
36”	52	52	53	52	53
38”	52	52	52	52	52
40”	52	52	52	52	52
Nothing	50	50	50	50	50

Table 6

Behaviors

Object Avoidance

STEVE is able to move around while avoiding obstacles. He does this using the three Sharp GP2D12 analog IR sensors and the 7 bump switches. Figure 12 shows a diagram of where the sensors are located on the robot.

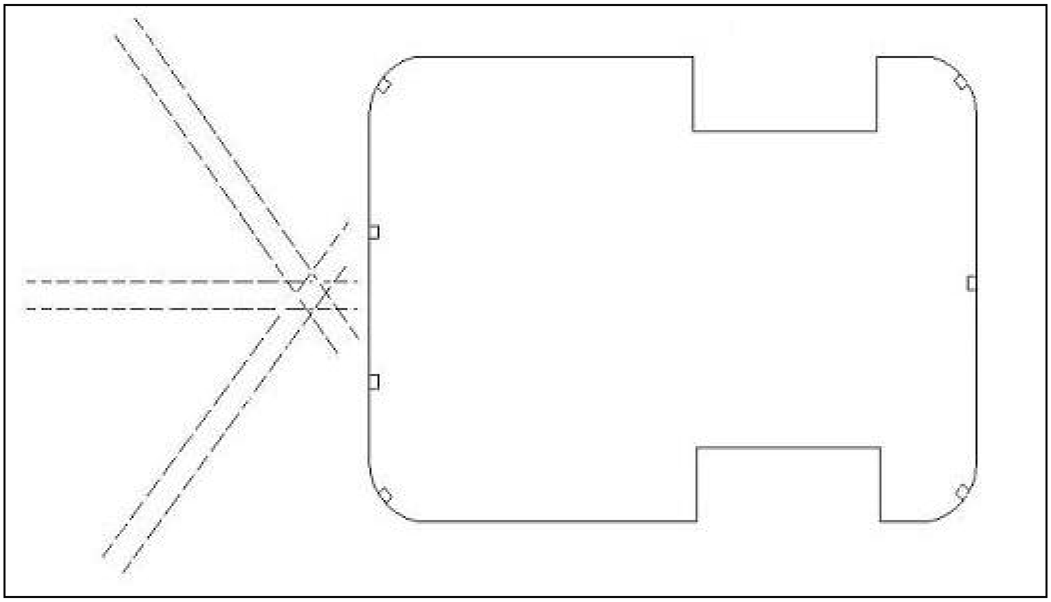

Figure 12

The three IR sensors form a grid and tell STEVE how far away on object is. Then, depending on how far away the object is, STEVE will react using the preset maneuvers. Figure 13 shows an approximate view of STEVE reactions to an object at varying distances. Once a reaction is determined, then the decision to react left of right needs to be made. To do this, STEVE looks at the values of the left and right IR sensors. If one value is greater than the other by 16, then STEVE will turn away from that sensor. Otherwise, if the left and right sensors are about equal, STEVE will turn in a random direction. Once a random direction is chosen, STEVE will continue to turn that way until he changes reactions (i.e. changing from react soft to react hard). Every time a new reaction starts, a new random direction is chosen.

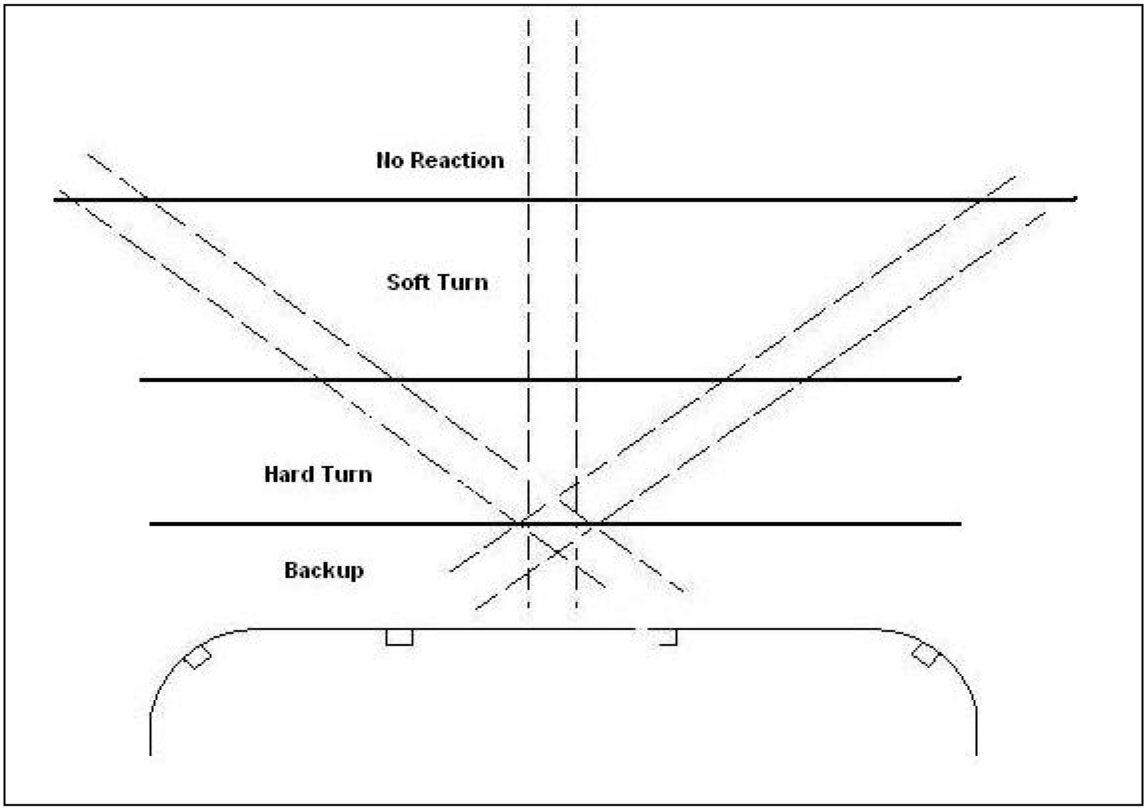

Figure 13

Object Following

STEVE is able to follow a moving object, provided the object is emitting IR modulated at 56.5kHz. STEVE uses three hacked LiteOn analog IR sensors to do this. The sensors are aligned so that they overlap to create five zones. Each zone is then assigned a reaction value. The alignment of the sensors and the associated reactions can be seen in Figure 14. This makes the logic for following very simple. If STEVE is ever in doubt as to which zone the object is in, he will react the same way he did last time (it is assumed the object wont move very far in the time between measurements). Also, if the front bumper is pressed during following, the analog IR sensors for obstacle avoidance are checked. If they indicate that an object is close, we assume that we ran into the speeding object. At this point we terminate the program and wait to be placed back in the speed trap.

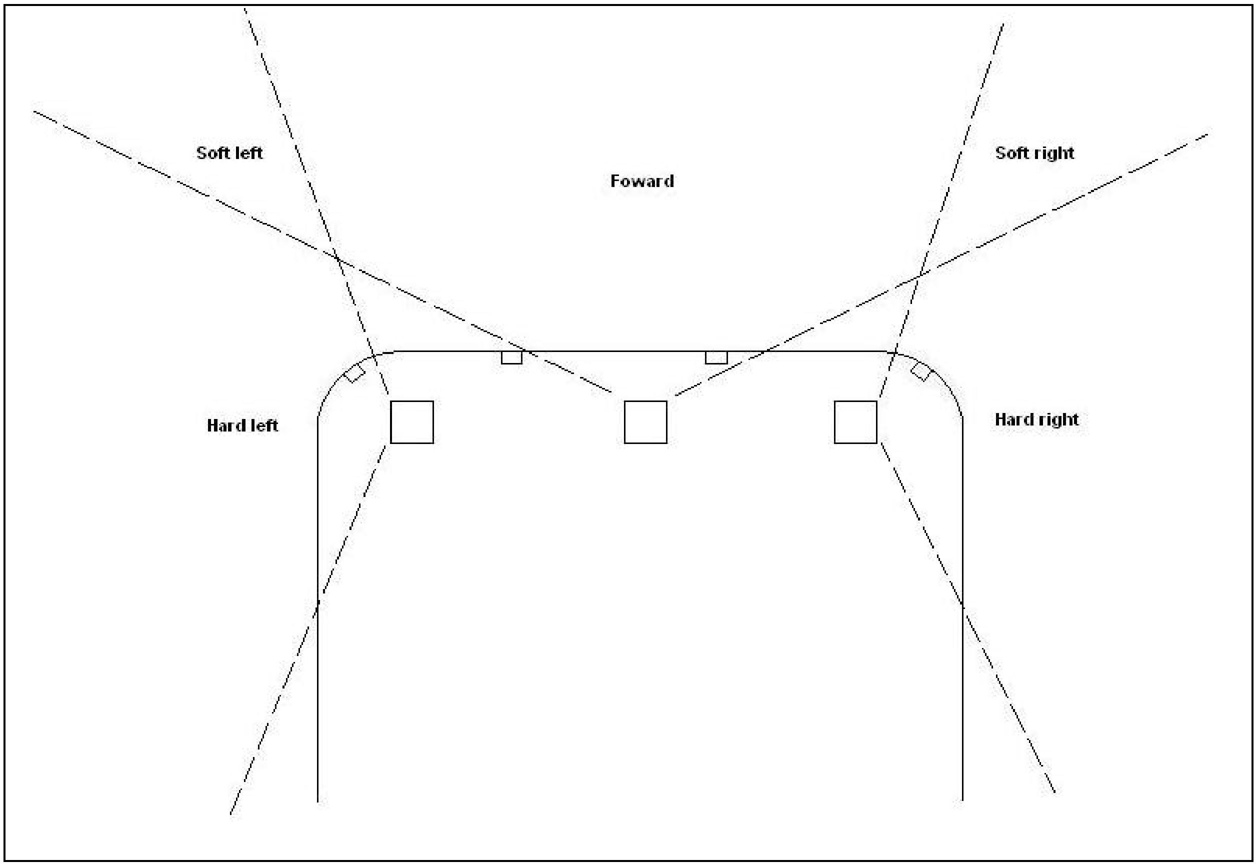

Figure 14

Conclusion

STEVE was the result of a lot of hard work. He was built from scratch. I assembled the entire electronics board. I built the memory expansion. The platform was designed, built and painted. I found a new hack for a digital IR receiver. I designed and built my unique speed trap sensor. STEVE, in his current form, almost does everything he is supposed to do. The only exception is the bumpers don't work for obstacle avoidance. The was a tremendous amount of noise on the bumper lines due to the electro-magnetic field generated by the motors. A primitive Faraday cage was built using a Barq's root beer can just to be able to use the front bumper of object following. This problem can probably be fixed with a better Faraday cage, but I felt it was better to leave STEVE as he is: WORKING.

My biggest success is my DC motors. The motors were a lot more work than servos, but they also perform much better. A lot of research went into find the motor driving chips and board. The motors also needed their own power source and to be optically isolated. But the end result was very impressive from the smooth handling to the speed STEVE is able to obtain.

If I had to start this project over, I would definitely choose a different electronics board. The board is nice, but I spent a ridiculous amount of time just trying to get the RAM memory expansion to work. I might also purchase larger DC motors for more speed. But other than that, I was very pleased with how STEVE ended up.

Documentation

Thanks to:

- Dr. Gugel and Scott Kanowitz for their help getting the memory expansion to work
- Rand Chandler for re-soldering my HC12 chip to my board
- Dr. Arroyo, Dr. Schwartz, and Aamir Qaiyumi for their help, guidance and abundance of new ideas to simplify and improve my robot
- National Semiconductor for the free motor driver chips I received
- Motor driver schematics from Magnevation (http://www.magnevation.com)
- Low battery circuit schematics from: http://www.ee.washington.edu/conselec/Sp96/projects/jeddk2/final/ee498h.htm
- My parents and girlfriend for the support and encouragement throughout the semester

Appendix A - Part Information

Component	**Vendor**	**Vendor Part#**
Laser pointers (Clearline Concepts)	Office Depot	075235520052 Model CL2005
IR Emitter/Detector Pair	Radio Shack	276-142
IR Detector Module (56.5kHz)	Jameco	176541
DC Gearhead Motors	Acroname	S5-GMOT-4
4N25 optoisolators	Jameco	40985
32kx8 SRAM	Jameco	75037
Motor driver board (Magnevation)	Acroname	R105-DC-MOTOR-D
Piezo buzzer	Radio Shack	273-065
Batteries (Ni-Cd AA's)	Radio Shack	230-449
Roll pins	Lowes	138836
Nylon spacers	Lowes	136921
7-segment latch/driver chips (DM9368)	Digi-Key	DM9368N-ND
Motor driver chips (LMD18200T)	National Semiconductor	LMD18200T

Appendix B – Source Code

```
* Filename      : SCI.ASM
* Programmer    : Michael Hattermann
* Date          : February 4, 2002
* Version       : 1.0
* Description   : This file contains SCI communication
*                  functions for input and output of
*                  data.  The following functions are
*                  available:
*
*                  WAIT_TC - wait for transmit complete
*                  SET_BAUD - change the baud rate
*                  TX_ON - turn transmitter on
*                  TX_OFF - turn transmitter off
*                  RX_ON - turn receiver on
*                  RX_OFF - turn receiver off
*                  RX_INT_ON - turn receiver interrupts on
*                  RX_INT_OFF - turn receiver interrupts off
*                  OUTCHAR - prints character to screen
*                  OUTSTR - prints string to screen
*                  INCHARWAIT - waits for character input
*                  INCHAR - get character input if any
*                  OUTNUM - prints number to screen
*                  NIBTOCHAR - prints nibble to screen
*                  OUTADDR - prints 16-bit num to screen
*                  INITSCI - turns on SCI for 9600 baud
*
*
*#define __DEBUGSCI_    1

#include "hc12.asm"

*
************************************************************
* SCI Equates
************************************************************
*
****Baud rate equates****
BAUD19200   EQU     0
BAUD14400   EQU     2
BAUD9600    EQU     4
BAUD4800    EQU     6
BAUD2400    EQU     8
BAUD1200    EQU     10
BAUD600     EQU     12
BAUD300     EQU     14

****ASCII character equates****
EOS         EQU     $04             ; User-defined End Of String (EOS) character
CR          EQU     $0D             ; Carriage Return Character
LF          EQU     $0A             ; Line Feed Character
ESC         EQU     $1B             ; ESC character

*
************************************************************
* SCI Test Program
************************************************************
*
#ifdef __DEBUGSCI_
            ORG     $0900

HELLO       DC.B    'Hello world!!!'
NEWLINE     DC.B    CR,LF           ; Newline string
            DC.B    EOS

PRESSKEY    DC.B    'Press any key to continue'
            DC.B    CR,LF,EOS
```

```
CHANGETO    DC.B    'Change baud rate to: '
            DC.B    EOS

PROMPT      DC.B    'Start typing...'
            DC.B    CR,LF,EOS

TEST        LDAA    #$00            ; turn off COP watchdog timer
            STAA    COPCTL

            LDS     #$0bff          ; init the stack pointer

            JSR     INITSCI         ; init SCI system
            LDX     #HELLO          ; print "hello world"
            JSR     OUTSTR          ;
            LDX     #CHANGETO       ; print change baud message
            JSR     OUTSTR          ;
            LDX     #$1200          ; print new baud rate
            JSR     OUTADDR         ;
            LDX     #NEWLINE        ; print new line
            JSR     OUTSTR          ;

            LDX     #PRESSKEY       ; print "press any key"
            JSR     OUTSTR          ;
            JSR     INCHARWAIT      ; wait for character

            LDAA    #BAUD1200       ; change baud rate
            JSR     SET_BAUD        ;
            LDX     #HELLO          ; print "hello world"
            JSR     OUTSTR          ;
            LDX     #$1234          ; test print address
            JSR     OUTADDR         ;
            LDX     #PROMPT         ; print prompt
            JSR     OUTSTR          ;
HERE        JSR     INCHARWAIT      ; get character
            JSR     OUTCHAR         ; echo to screen
            BRA     HERE            ; end of program

#endif
*
************************************************************
* Constant Definitions
************************************************************
*
BAUDTBL     DC.W    13              ; (0) BAUD rate = 19200
            DC.W    17              ; (1) BAUD rate = 14400
            DC.W    26              ; (2) BAUD rate = 9600
            DC.W    52              ; (3) BAUD rate = 4800
            DC.W    104             ; (4) BAUD rate = 2400
            DC.W    208             ; (5) BAUD rate = 1200
            DC.W    417             ; (6) BAUD rate = 600
            DC.W    833             ; (7) BAUD rate = 300
*
*******************************************************************************
*                       SUBROUTINE -  WAIT_TC
* Description: Waits for the current transmit operation to complete (polls the
*                TC flag in SCI status register 1)
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
WAIT_TC
            BRCLR   SC0SR1,BIT6,WAIT_TC     ; wait until done sending
            RTS                             ; Return to caller
*
*
*******************************************************************************
*                       SUBROUTINE -  SET_BAUD
* Description: Sets the baud rate to the rate specified in register A.  Reg A
*               can only take on these predefined values:
```

```
*       BAUD19200     = BAUD rate 19200
*       BAUD14400     = BAUD rate 14400
*       BAUD9600      = BAUD rate 9600
*       BAUD4800      = BAUD rate 4800
*       BAUD2400      = BAUD rate 2400
*       BAUD1200      = BAUD rate 1200
*       BAUD600       = BAUD rate 600
*       BAUD300       = BAUD rate 300
*
* Input         : New baud rate in reg A
* Output        : None.
* Destroys      : SC0BDH, SC0BDL.
* Calls         : None.
*******************************************************************************
*
SET_BAUD
            PSHX                    ; Preserve reg X

            LDX     #BAUDTBL        ; Load address of baud table
            LDX     A,X             ; Load baud rate from table
            STX     SC0BD           ; Set baud rate in register

            PULX                    ; Restore reg X
            RTS                     ; Return to caller
*
*******************************************************************************
*                       SUBROUTINE -  TX_ON, TX_OFF
* Description: Enables transmitter, disables transmitter
* Input         : None.
* Output        : None.
* Destroys      : SC0CR2.
* Calls         : None.
*******************************************************************************
*
TX_ON
            BSET    SC0CR2,BIT3     ; turn on the transmitter
            RTS                     ; return to caller
TX_OFF
            BCLR    SC0CR2,BIT3     ; turn off the transmitter
            RTS                     ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  RX_ON, RX_OFF,RX_INT_ON,RX_INT_OFF
* Description: Enables receiver, disables receiver, enables receive interrupts,
*               disables receive interrupts
* Input         : None.
* Output        : None.
* Destroys      : SC0CR2.
* Calls         : None.
*******************************************************************************
*
RX_ON
            BSET    SC0CR2,BIT2     ; turn on the receiver
            RTS                     ; return to caller
RX_OFF
            BCLR    SC0CR2,BIT2     ; turn off the receiver
            RTS                     ; return to caller
RX_INT_ON
            BSET    SC0CR2,BIT5     ; enable receiver interrupts
            RTS                     ; return to caller
RX_INT_OFF
            BCLR    SC0CR2,BIT5     ; disable receiver interrupts
            RTS                     ; return to caller
*
************************************************************************
*                       SUBROUTINE -  OUTCHAR
* Description: Outputs the character in register A to the screen
* Input         : Data to be transmitted in register A.
* Output        : Transmits the data.
* Destroys      : None.
* Calls         : WAIT_TC
```

```
************************************************************************
*
OUTCHAR
            JSR     WAIT_TC         ; wait until transmitter is idle
            STAA    SC0DRL          ; output character
            RTS                     ; Return from subtoutine
*
************************************************************************
*                       SUBROUTINE -  OUTSTR
* Description: Outputs the string pointed to by X.  String must be
*               terminated by EOS character.
* Input         : String to be output in reg X
* Output        : Transmits the string.
* Destroys      : None.
* Calls         : OUTCHAR
************************************************************************
*
OUTSTR
            PSHA                    ; preserve reg A
            PSHX                    ; preserve reg X
OUTSTR1
            LDAA    1,X+            ; Get a character (put in reg A)
            CMPA    #EOS            ; Check if it's EOS
            BEQ     OUTSTR2         ; Branch to Done if it's EOS
            JSR     OUTCHAR         ; Print the character
            BRA     OUTSTR1
OUTSTR2
            PULX                    ; restore reg X
            PULA                    ; restore reg A
            RTS                     ; Return from subtoutine
*
************************************************************************
*                SUBROUTINE  -  INCHARWAIT
* Description: Waits for a character to be pressed and reads it into
*               reg A
* Input         : None
* Output        : Character pressed in reg. A
* Destroys      : A.
* Calls         : None
************************************************************************
*
INCHARWAIT
            BRCLR   SC0SR1,BIT5,INCHARWAIT  ; wait until buffer full
            LDAA    SC0DRL                  ; input character
            RTS                             ; Return from subroutine
*
************************************************************************
*                SUBROUTINE  -  INCHAR
* Description: Checks to see if character recevied - if so returns the
*               character, if not returns 0
* Input         : None
* Output        : Character pressed in reg A; 0 if none
* Destroys      : A.
* Calls         : None
************************************************************************
*
INCHAR
            BRCLR   SC0SR1,BIT5,INCHAR1     ; if there is no data, get out
            LDAA    SC0DRL                  ; yes, read data
INCHAR1
            RTS                             ; return to caller
*
************************************************************************
*                       SUBROUTINE -  OUTNUM
* Description: Outputs the number in register A to the screen
* Input         : Data to be transmitted in register A.
* Output        : Transmits the data.
* Destroys      : None.
* Calls         : NIBTOCHAR
************************************************************************
*
```

```
OUTNUM
            PSHA                    ; preserve reg A
            PSHA                    ; preserve reg A
            ANDA    #%11110000      ; get upper nibble
            LSRA                    ; shift it right to get the nibble
            LSRA
            LSRA
            LSRA
            JSR     NIBTOCHAR       ; change A and print it
            PULA                    ; restore reg A
            ANDA    #%00001111      ; get lower nibble
            JSR     NIBTOCHAR       ; change A and print it
            PULA                    ; restore reg A
            RTS                     ; return to caller
*
************************************************************************
*                       SUBROUTINE -  NIBTOCHAR
* Description: Converts lower nibble of A to ASCII and prints it
* Input         : Data to convert in A.
* Output        : Transmits the data.
* Destroys      : None.
* Calls         : OUTCHAR
************************************************************************
*
NIBTOCHAR
            CMPA    #9             ; is it greater than 9?
            BGT     NIBTOCHAR1     ; if so, print a character
            ADDA    #48            ; if not, print a number starting at 48 ASCII
            BRA     NIBTOCHAR2     ;
NIBTOCHAR1
            ADDA    #55            ; if so, print a letter starting at 55 = 65-10
NIBTOCHAR2
            JSR     OUTCHAR        ; print it
            RTS
*
************************************************************************
*                       SUBROUTINE -  OUTADDR
* Description: Outputs the number in reg X to the screen
* Input         : Data to print in X.
* Output        : Transmits the data.
* Destroys      : None.
* Calls         : OUTNUM
************************************************************************
*
OUTADDR
            PSHD                    ; save reg D
            TFR     X,D             ; load X into D
            JSR     OUTNUM          ; prints whats in A -- MSB
            TBA                     ; B -> A
            JSR     OUTNUM          ; prints whats in B -- LSB
            PULD                    ; restore D
            RTS                     ; return to caller
*
************************************************************************
*                       SUBROUTINE - INITSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and receiver
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : SET_BAUD,TX_ON,RX_ON
************************************************************************
*
INITSCI     PSHA                    ; save reg A
            LDAA    #BAUD9600       ; set the baud rate to 9600
            JSR     SET_BAUD        ;
            JSR     TX_ON           ; turn on the transmitter
            JSR     RX_ON           ; turn on the receiver
            PULA                    ; restore reg A
            RTS                     ; Return from subtoutine
```

```
*
************************************************************************

* Filename      : ATD.ASM
* Programmer    : Michael Hattermann
* Date          : February 22, 2002
* Version       : 1.0
* Description   : This file contains the Analog to
*                  Digital (A/D) conversion functions
*                  for the input of analog signals. The
*                  following functions are available:
*
*                   INITATD - Initializes the ATD system
*                   KILLATD - Shuts down the ATD system
*                   ANALOG - Returns ATD value for port specified
*
*#define __DEBUGATD_     1


#include "hc12.asm"
*
************************************************************
* A/D Equates
************************************************************
*
CHANNEL0    EQU     $00
CHANNEL1    EQU     $01
CHANNEL2    EQU     $02
CHANNEL3    EQU     $03
CHANNEL4    EQU     $04
CHANNEL5    EQU     $05
CHANNEL6    EQU     $06
CHANNEL7    EQU     $07
*
************************************************************
* A/D Channel Assignments
************************************************************
*
LEFTIR          EQU     CHANNEL0
RIGHTIR         EQU     CHANNEL1
CENTERIR        EQU     CHANNEL2
CENTERFOLLOW    EQU     CHANNEL4
LEFTFOLLOW      EQU     CHANNEL5
RIGHTFOLLOW     EQU     CHANNEL6
BUMPER          EQU     CHANNEL7
*
************************************************************
* A/D Debug Code
************************************************************
*
#ifdef __DEBUGATD_
            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $B000

TEST        LDAA    #$00            ; turn off COP watchdog timer
            STAA    COPCTL

            LDS     #$0bff          ; init the stack pointer

            JSR     INITATD         ; init A/D system
            JSR     INITSCI         ; init SCI system
            MOVB    #$80,TSCR       ; enable the timer

HERE
*            LDX     #LEFT           ; print left IR message
*            JSR     OUTSTR          ;
            LDAA    #CHANNEL7       ; get left IR value
            JSR     ANALOG          ;
            JSR     OUTNUM          ; print value
            LDX     #NEWLINE        ; print newline
            JSR     OUTSTR          ;

*            LDX     #RIGHT          ; print right IR message
*            JSR     OUTSTR          ;
*            LDAA    #CHANNEL1       ; get right IR value
*            JSR     ANALOG          ;
*            JSR     OUTNUM          ; print value
*            LDX     #NEWLINE        ; print newline
*            JSR     OUTSTR          ;

            LDX     #500            ; wait 1/2 second
            JSR     WAIT            ;

            BRA     HERE            ; end of program

LEFT        DC.B    'Left IR value = '
            DC.B    EOS
RIGHT       DC.B    'Right IR value = '
            DC.B    EOS
NEWLINE     DC.B    CR,LF           ; Newline string
            DC.B    EOS

#include "sci.asm"
#include "wait.asm"

#endif
*
********************************************************************************
*                       SUBROUTINE -  INITATD
* Description: Initializes the analog to digital converter
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
INITATD     PSHA                        ; save reg A
            MOVB    #$80,ATDCTL2        ; turn on ATD system
            MOVB    #$00,ATDCTL3        ; enable conversions in bgnd mode
            MOVB    #$01,ATDCTL4        ; setup conversion rate = 2MHz

            LDAA    #195                ; load loop counter
INITATD1    NOP                         ; wait for ATD to power up
            DBNE    A,INITATD1          ; if we still need to wait, wait

            PULA                        ; restore reg A
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  KILLATD
* Description: Shuts down the analog to digital converter
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
KILLATD     MOVB    #$00,ATDCTL0        ; stop current conversion (if there is one)
            MOVB    #$00,ATDCTL2        ; turn off ATD system
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  ANALOG
* Description: Converts the analog channel specified by reg A and returns the
*               converted value in reg A.  Valid values for channel are (the
*               equates an be found above):
*
*                   CHANNEL0    - A/D Channel #0
*                   CHANNEL1    - A/D Channel #1
*                   CHANNEL2    - A/D Channel #2
*                   CHANNEL3    - A/D Channel #3
*                   CHANNEL4    - A/D Channel #4
*                   CHANNEL5    - A/D Channel #5
*                   CHANNEL6    - A/D Channel #6
*                   CHANNEL7    - A/D Channel #7
*
* Input         : Channel to convert in reg A.
* Output        : Digital value of channel in reg A.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
ANALOG      STAA    ATDCTL5                     ; start conversion on channel specified
ANALOG1     BRCLR   ATDSTATH,BIT7,ANALOG1       ; wait for conversion to complete
            LDAA    ADR2H                       ; load conversion result
            RTS                                 ; return to caller
*
*************************************************************************

* Filename      : PWM.ASM
* Programmer    : Michael Hattermann
* Date          : February 22, 2002
* Version       : 1.0
* Description   : This file contains the pulse width
*                   modulation functions for generating
*                   an output waveform.  The following
*                   functions are available
*
*                       INITPWM - inits PWM system
*                       KILLPWM - shut down PWM system
*                       LEFTMOTOR - sets spd,dir for left motor
*                       RIGHTMOTOR - sets spd,dir for right motor
*                       CHNGSPEED - sets new speed for motors
*                       STEER - sets motors to perform known manuevers
*                       PULLOUT - move to begin chase
*
*
*#define __DEBUGPWM_     1


#include "hc12.asm"
*
*************************************************************
* PWM Equates
*************************************************************
*
PWM0        EQU     0           ; left motor
PWM1        EQU     1           ; right motor
PWM2        EQU     2
PWM3        EQU     3
PWM4        EQU     4           ; left direction
PWM5        EQU     5           ; left brake
PWM6        EQU     6           ; right direction
PWM7        EQU     7           ; right brake

ACCELCONST  EQU     6           ; acceleration constant

FULLSPEED   EQU     200
_7_8_SPEED  EQU     175
_3_4_SPEED  EQU     150
_5_8_SPEED  EQU     125
HALFSPEED   EQU     100
_3_8_SPEED  EQU     75
_1_4_SPEED  EQU     50
_1_8_SPEED  EQU     25
STOPSPEED   EQU     0

LEFTDIR     EQU     BIT6
LEFTBRK     EQU     BIT7
RIGHTDIR    EQU     BIT4
RIGHTBRK    EQU     BIT5
```

```
GOFOWARD    EQU     0
GOBACK      EQU     1
HARDLEFT    EQU     2
SOFTLEFT    EQU     3
HARDRIGHT   EQU     4
SOFTRIGHT   EQU     5
BACKLEFT    EQU     6
BACKRIGHT   EQU     7
STOP        EQU     8

*
*************************************************************
* PWM Debug Code
*************************************************************
*
#ifdef __DEBUGPWM_
            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $B000
TEST
            JSR     INITPWM         ; initialize the PWM system
            LDD     #HALFSPEED      ; load speed
            JSR     CHNGSPEED       ; set max speed of motors
TEST1
            LDX     #GOFOWARD       ; load manuever
            MOVB    #GOFOWARD,SEG7PORT      ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 1 sec
            JSR     WAIT            ;

            LDX     #GOBACK         ; load manuever
            MOVB    #GOBACK,SEG7PORT        ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 1 sec
            JSR     WAIT            ;

            LDX     #HARDLEFT       ; load manuever
            MOVB    #HARDLEFT,SEG7PORT      ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 10 sec
            JSR     WAIT            ;

            LDX     #SOFTLEFT       ; load manuever
            MOVB    #SOFTLEFT,SEG7PORT      ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 1 sec
            JSR     WAIT            ;

            LDX     #HARDRIGHT      ; load manuever
            MOVB    #HARDRIGHT,SEG7PORT     ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 10 sec
            JSR     WAIT            ;

            LDX     #SOFTRIGHT      ; load manuever
            MOVB    #SOFTRIGHT,SEG7PORT     ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 1 sec
            JSR     WAIT            ;

            LDX     #STOP           ; load manuever
            MOVB    #STOP,SEG7PORT          ; write direction to port
            JSR     STEER           ; move the robot
            LDX     #5000           ; wait 1 sec
            JSR     WAIT            ;

            BRA     TEST1
            SWI
```

```
OLEFTSPD        DC.W    $0000           ; current speed of left motor
NLEFTSPD        DC.W    $0000           ; next speed of left motor
ORIGHTSPD       DC.W    $0000           ; current speed of right motor
NRIGHTSPD       DC.W    $0000           ; next speed of right motor

MAXFOWARDSPD    DC.W    $0000           ; maximum foward speed for motors
MAXBACKSPD      DC.W    $0000           ; maximum reverse speed for motors
CURRMAN         DC.W    $0000           ; current manuever being performed

#include "wait.asm"
#endif
*
*******************************************************************************
*                       SUBROUTINE -  INITPWM
* Description: Initializes the pulse width modulation system
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
INITPWM     MOVB    #$08,PWCLK          ; set prescaler bits, concatenate PWM channels
            MOVB    #$33,PWPOL          ; set clock source (S0,S1), polarity (high first)
            MOVB    #99,PWSCAL0         ; set S0 clock prescaler
            MOVB    #$00,PWSCAL1        ; init S1 clock prescaler
            MOVB    #200,PWPER0         ; set period of PWM0 to about 20ms
            MOVB    #200,PWPER1         ; set period of PWM1 to about 20ms
            MOVB    #$FF,PWPER2         ; init period of PWM2
            MOVB    #$FF,PWPER3         ; init period of PWM3
            MOVB    #$00,PWDTY0         ; init duty cycle to 0%
            MOVB    #$00,PWDTY1         ; init duty cycle to 0%
            MOVB    #$00,PWDTY2         ; init duty cycle to 0%
            MOVB    #$00,PWDTY3         ; init duty cycle to 0%
            MOVB    #$00,PWCTL          ; init PWM to run normally, left aligned
            MOVB    #$50,DDRP           ; configure unused bits to be outputs (motor controls)
            MOVB    #$50,PORTP          ; set direction to foward, brake to off
            MOVB    #$03,PWEN           ; enable the PWM channels
            RTS                         ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  KILLPWM
* Description: Shuts down the pulse width modulation system
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
KILLPWM     MOVB    #$00,PWEN           ; turn off PWM channels
            RTS                         ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  LEFTMOTOR
* Description: Sets the left motor to the speed specified.  Valid
*               speed values range from -100 to 100, where 0 to -100 is back-
*               wards speed and 0 to 100 is foward speed.
*
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
LEFTMOTOR   PSHX                        ; save reg X
            PSHY                        ; save reg Y
            PSHD                        ; save reg D

            LDD     OLEFTSPD            ; load the old speed
            LDY     #ACCELCONST         ; get the acceleration constant
            EMULS                       ; multiply old speed by acceleration constant
            ADDD    NLEFTSPD            ; add new speed to result
            LDX     #(ACCELCONST+1)     ; load divisor
            IDIVS                       ; calc new speed
            XGDX                        ; put new speed in reg D
            STD     OLEFTSPD            ; save new speed as old speed
            BLT     LEFTMOTOR1          ; if new speed is negative, branch
            BSET    PORTP,LEFTDIR       ; set direction to foward
            BRA     LEFTMOTOR2          ; continue
LEFTMOTOR1  BCLR    PORTP,LEFTDIR       ; set direction to reverse
            NEGB                        ; take abs value of speed
LEFTMOTOR2  STAB    PWDTY1              ; set the new speed for the left motor

            PULD                        ; restore reg D
            PULY                        ; restore reg Y
            PULX                        ; restore reg X
LEFTMOTORX  RTS                         ; return to caller
*
*******************************************************************************
*                        SUBROUTINE -  RIGHTMOTOR
* Description: Sets the right motor to the speed specified.  Valid
*               speed values range from -100 to 100, where 0 to -100 is back-
*               wards speed and 0 to 100 is foward speed.
*
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
RIGHTMOTOR  PSHX                        ; save reg X
            PSHY                        ; save reg Y
            PSHD                        ; save reg D

            LDD     ORIGHTSPD           ; load the old speed
            LDY     #ACCELCONST         ; get the acceleration constant
            EMULS                       ; multiply old speed by acceleration constant
            ADDD    NRIGHTSPD           ; add new speed to result
            LDX     #(ACCELCONST+1)     ; load divisor
            IDIVS                       ; calc new speed
            XGDX                        ; put new speed in reg D
            STD     ORIGHTSPD           ; save new speed as old speed
            BLT     RIGHTMOTOR1         ; if new speed is negative, branch
            BSET    PORTP,RIGHTDIR      ; set direction to foward
            BRA     RIGHTMOTOR2         ; continue
RIGHTMOTOR1 BCLR    PORTP,RIGHTDIR      ; set direction to reverse
            NEGB                        ; take abs value of speed
RIGHTMOTOR2 STAB    PWDTY0              ; set the new speed for the right motor

            PULD                        ; restore reg D
            PULY                        ; restore reg Y
            PULX                        ; restore reg X
RIGHTMOTORX RTS                         ; return to caller
*
*******************************************************************************
*                        SUBROUTINE -  CHNGSPEED
* Description: Changes the maximum speed for either motor to the speed passed
*               in register D.
* Input         : Speed(reg D).
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
CHNGSPEED   STD     MAXFOWARDSPD        ; save max speed
            STD     MAXBACKSPD          ; save max speed to backward
            COM     MAXBACKSPD          ; convert to negative speed
            NEG     MAXBACKSPD+1        ;
            RTS                         ; return to caller
*
*******************************************************************************
*                        SUBROUTINE -  STEER
* Description: Sets up the left and right motors to perform the specified
*               manuever (reg X). The value of manuever must be one of the
*               following:
*
*                   GOFOWARD
*                   GOBACK
*                   HARDLEFT
*                   SOFTLEFT
*                   HARDRIGHT
*                   SOFTRIGHT
*                   BACKLEFT
*                   BACKRIGHT
*                   STOP
*
* Input         : Manuever(reg X).
* Output        : None.
* Destroys      : None.
* Calls         : LEFTMOTOR,RIGHTMOTOR.
*******************************************************************************
*
STEER       PSHD                            ; save register D
            PSHX                            ; save register X

            STX     CURRMAN                 ; save manuever
            TBNE    X,STEER1                ; if not FOWARD, continue
            LDD     MAXFOWARDSPD            ; load maximum foward speed
            STD     NRIGHTSPD               ; right motor full foward
            STD     NLEFTSPD                ; left motor foward full
            BRA     STEERX                  ; get out

STEER1      DBNE    X,STEER2                ; if not BACKWARD, continue
            LDD     MAXBACKSPD              ; load maximum back speed
            STD     NRIGHTSPD               ; right motor back full
            STD     NLEFTSPD                ; left motor back full
            BRA     STEERX                  ; get out

STEER2      DBNE    X,STEER3                ; if not HARD LEFT, continue
            LDD     MAXFOWARDSPD            ; load maximum foward speed
            STD     NRIGHTSPD               ; right motor foward full
            LDD     MAXBACKSPD              ; load maximum back speed
            STD     NLEFTSPD                ; left motor backward full
            BRA     STEERX                  ; get out

STEER3      DBNE    X,STEER4                ; if not SOFT LEFT, continue
            LDD     MAXFOWARDSPD            ; load maximum foward speed
            STD     NRIGHTSPD               ; right motor foward full
            LSRD                            ; set left motor speed
            STD     NLEFTSPD                ; left motor foward half
            BRA     STEERX                  ; get out

STEER4      DBNE    X,STEER5                ; if not HARD RIGHT, continue
            LDD     MAXFOWARDSPD            ; load maximum foward speed
            STD     NLEFTSPD                ; left motor foward full
            LDD     MAXBACKSPD              ; load maximum back speed
            STD     NRIGHTSPD               ; right motor back full
            BRA     STEERX                  ; get out

STEER5      DBNE    X,STEER6                ; if not SOFT RIGHT, continue
            LDD     MAXFOWARDSPD            ; load maximum foward speed
            STD     NLEFTSPD                ; left motor foward full
            LSRD                            ; set right motor speed
            STD     NRIGHTSPD               ; right motor foward half
            BRA     STEERX                  ; get out


STEER6      DBNE    X,STEER7                ; if not BACK LEFT, continue
            LDD     MAXBACKSPD              ; load maximum foward speed
            STD     NRIGHTSPD               ; right motor back full
            LSRD                            ; set left motor speed
            STD     NLEFTSPD                ; left motor foward half
            BRA     STEERX                  ; get out
```

```
STEER7      DBNE    X,STEER8                ; if not BACK RIGHT, continue
            LDD     MAXBACKSPD              ; load maximum foward speed
            STD     NLEFTSPD                ; left motor back full
            LSRD                            ; set left motor speed
            STD     NRIGHTSPD               ; right motor foward half
            BRA     STEERX                  ; get out

STEER8      DBNE    X,STEERX                ; if not STOP, get out
            MOVW    #$0000,NLEFTSPD         ; stop left motor
            MOVW    #$0000,NRIGHTSPD        ; stop right motor

STEERX      PULX                            ; restore register X
            PULD                            ; restore register D
            RTS                             ; return to caller
*
*******************************************************************************
*                        SUBROUTINE -  PULLOUT
* Description: Performs pre-programmed manuever to pull out onto the road and
*               turn toward moving objext to begin chasing
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
PULLOUT     PSHD                        ; save reg D
            PSHX                        ; save reg X

            LDD     #FULLSPEED          ; set motor speed
            JSR     CHNGSPEED           ;

            LDX     #GOFOWARD           ; pull foward
            JSR     STEER               ;
            LDX     #60                 ; for a little bit
            JSR     WAIT                ;

            LDAA    SPDCAUGHT           ; get direction of speeder
            CMPA    #LSPDFLAG           ; was the speeder going left?
            BEQ     PULLOUTL            ; yes, so go pull out left
PULLOUTR
            LDX     #SOFTRIGHT          ; no, then pull out right
            JSR     STEER               ;
            LDX     #625                ; wait for manuever
            JSR     WAIT                ;
            BRA     PULLOUTX            ; get out
PULLOUTL
            LDX     #SOFTLEFT           ; turn left
            JSR     STEER               ;
            LDX     #600                ; wait for manuever
            JSR     WAIT                ;

PULLOUTX    PULX                        ; restore reg X
            PULD                        ; restore reg D
            RTS                         ; return to caller
*
*******************************************************************************

* Filename      : TIME.ASM
* Programmer    : Michael Hattermann
* Date          : February 21, 2002
* Version       : 1.0
* Description   : This file contains time functions.  The
*                  following functions are available:
*
*                   INITTIME - initialize timer system
*                   KILLTIME - shut down timer system
*                   WAITSPEED - waits for a speeder to go by
*                   FLASHON - turns on flashing lights
*                   FLASHOFF - turns off flashing lights
*                   SIRENON - turns on the siren
*                   SIRENOFF - turns off the siren
*                   LEFTSTS - handles left speed trap sensor
*                   RIGHTSTS - handles right speed trap sensor
*                   TIMEEXT - handles extended timer,flashing lights,siren
*                   UPDMOTORS - updates speed on motors
*                   SIREN - handles siren output
*
*#define __DEBUGTIME_        1
*#define __PRINTTIME_        1

#include "hc12.asm"

*
************************************************************
* Time Equates
************************************************************
*
LSPDFLAG        EQU         BIT0    ; Flags to indicate if a
RSPDFLAG        EQU         BIT1    ;  beam was broken
BSPDFLAG        EQU         3       ;

SPDLIMIT        EQU         $1F     ; speed limit
FLASHRATE       EQU         15      ; flashing light rate (1/4 second)
FLASHPAT1       EQU         $0F     ; 1st light pattern
FLASHPAT2       EQU         $F0     ; 2nd light pattern

SIRENRATE1      EQU         -10     ; siren frequency change rate
SIRENRATE2      EQU         10      ; siren frequency change rate
SIRENFREQ1      EQU         2000    ; first siren frequency
SIRENFREQ2      EQU         1200    ; second siren frequency

*
************************************************************
* Time Debug Program
************************************************************
*
#ifdef  __DEBUGTIME_
            ORG     T0_PVECT
            JMP     LEFTSTS
            ORG     T1_PVECT
            JMP     RIGHTSTS
            ORG     TMR_OVER_PVECT
            JMP     TIMEEXT
            ORG     RTI_PVECT
            JMP     UPDMOTORS
            ORG     T6_PVECT
            JMP     SIREN
*
************************************************************
* Time Global Vairables
************************************************************
*
                ORG     $0900
UPPERTIMER      DC.W    $0000           ; 16-bit extension of TCNT
LEFTSPDTIME     DC.W    $0000           ; Time left break beam was broken
RIGHTSPDTIME    DC.W    $0000           ; Time right break beam was broken
SPDTIMEFLG      DC.B    $00             ; Flags to indicate which beams were broken
SPDCAUGHT       DC.B    $00             ; Flags to indicate speeder caught
FLASHTIMER      DC.B    $00             ; Timer for flashing lights
FLASHPREV       DC.B    $00             ; Previous status of lights
SIRENFREQ       DC.W    $0000           ; Frequency of the siren
SIRENTIMER      DC.W    $0000           ; Timer for siren


            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $B000
TEST        LDAA    #$00            ; turn off COP watchdog timer
            STAA    COPCTL
            LDS     #$0bff          ; init the stack pointer
            JSR     INITTIME        ; initialize the time system

*            LDX     #0              ; wait 0 ms
*            JSR     WAIT            ;
*            LDX     #1              ; wait 1 ms
*            JSR     WAIT            ;
*            LDX     #10             ; wait 10 ms
*            JSR     WAIT            ;

            CLI                     ; turn on interrupts
            JSR     FLASHON         ;
*            JSR     SIRENON

HERE        BRA     HERE            ; end of program

LEFTMOTOR       RTS
RIGHTMOTOR      RTS

LEFTSPSTR   DC.B    'L'
            DC.B    CR,LF,EOS
RIGHTSPSTR  DC.B    'R'
            DC.B    CR,LF,EOS
NEWLINE     DC.B    CR,LF,EOS

#include "sci.asm"
*#include "wait.asm"

#endif
*
*******************************************************************************
*                       SUBROUTINE -  INITTIME
* Description: Initializes the timer system
*               TC0     - left speed trap IR receiver
*               TC1     - right speed trap IR receiver
*               TC6     - siren output
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
INITTIME    MOVB    #$40,TIOS           ; setup input capture/output compare lines
            MOVB    #$00,CFORC          ; setup timer compare force register
            MOVB    #$00,OC7M           ; setup OC7 mask register
            MOVB    #$00,OC7D           ; setup OC7 data register
            MOVB    #$00,TCTL1          ; setup output compare pin action reg 1
            MOVB    #$00,TCTL2          ; setup output compare pin action reg 2
            MOVB    #$00,TCTL3          ; setup input capture edge detection reg 1
            MOVB    #$05,TCTL4          ; setup input capture edge detection reg 2
            MOVB    #$03,TMSK1          ; setup interrupts on timer lines
            MOVB    #$A0,TMSK2          ; setup misc timer pin options(pullups on IC pins)
            MOVW    #$0000,TC0          ; clear TC0 register
            MOVW    #$0000,TC1          ; clear TC1 register
            MOVW    #$0000,TC2          ; clear TC2 register
            MOVW    #$0000,TC3          ; clear TC3 register
            MOVW    #$0000,TC4          ; clear TC4 register
            MOVW    #$0000,TC5          ; clear TC5 register
            MOVW    #$0000,TC6          ; clear TC6 register
            MOVW    #$0000,TC7          ; clear TC7 register
            MOVB    #$FF,TFLG1          ; clear all interrupt flags
            MOVB    #$80,TFLG2          ; clear interrupt flag
            MOVW    #$0000,UPPERTIMER   ; clear timer extension timer
            MOVW    #$0000,LEFTSPDTIME  ; clear left speed time
            MOVW    #$0000,RIGHTSPDTIME ; clear left speed time
            MOVB    #$00,FLASHTIMER     ; clear flashing lights timer
            MOVB    #$00,SPDTIMEFLG     ; clear broken beam flags
            MOVB    #$00,SPDCAUGHT      ; clear speeder caught flag
            MOVB    #$00,FLASHPREV      ; turn off flashing lights
            MOVW    #$0000,SIRENFREQ    ; clear siren frequency
            MOVW    #$0000,SIRENTIMER   ; clear siren timer
            MOVB    #$00,BUMPVALUE      ; clear bumper value
            MOVB    #$86,RTICTL         ; set up, enable RTI
            MOVB    #$FF,RTIFLG         ; clear all RTI flags
            MOVB    #$80,TSCR           ; enable the timer
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  KILLTIME
* Description: Shuts down the timer system
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
KILLTIME    MOVB    #$00,TCTL1          ; disconnect all output compare pins
            MOVB    #$00,TCTL2          ;
            MOVB    #$00,TCTL3          ; disable all input capture pins
            MOVB    #$00,TCTL4          ;
            MOVB    #$00,TMSK1          ; turn off interrupts
            MOVB    #$00,TMSK2          ; turn off timer overflow interrupts
            MOVB    #$00,PACTL          ; turn off pulse accumulator
            MOVB    #$00,TSCR           ; turn off timer
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  WAITSPEED
* Description: Waits for a speeder to be caught and returns the direction the
*               object was travelling (Reg A).  Valid values are:
*                    LSPDFLAG - object was going left
*                    RSPDFLAG - object was going right
*
* Input         : None.
* Output        : Direction in reg A.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
WAITSPEED
WAITSPEED1  LDAA    SPDCAUGHT       ; get flags for speeder caught
            TBEQ    A,WAITSPEED1    ; wait for a caught speeder
            BCLR    TMSK1,#BIT0     ; turn off left interrupts
            BCLR    TMSK1,#BIT0     ; turn off right interrupts
            RTS                     ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  FLASHON
* Description: Turns the flashing lights on
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
FLASHON     MOVB    #FLASHPAT1,FLASHPREV    ; turn on the lights
            RTS                             ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  FLASHOFF
* Description: Turns the flashing lights off
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
FLASHOFF    MOVB    #$00,FLASHPREV          ; turn off the lights
            RTS                             ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  SIRENON
* Description: Turns the siren on
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
SIRENON     MOVB    #$10,TCTL1              ; turn on pin action for siren
            BSET    TMSK1,#BIT6              ; turn on interrupts for siren
            MOVW    #SIRENFREQ1,SIRENFREQ   ; initialize the siren frequency

            MOVW    #SIRENRATE1,SIRENTIMER  ; start going up in frequency first

            RTS                             ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  SIRENOFF
* Description: Turns the siren off
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
SIRENOFF    BCLR    TMSK1,#BIT6              ; turn off interrupts for siren
            MOVB    #$00,TCTL1              ; turn off pin action for siren
            MOVW    #$0000,SIRENFREQ        ; clear siren frequency
            RTS                             ; return to caller
*
*******************************************************************************
*                 INTERRUPT SERVICE ROUTINE - LEFTSTS
* Description: Handles the processing for a signal received from the left
*               speed trap sensor.
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
LEFTSTS     BRCLR   TFLG1,BIT0,LEFTSTSX     ; make sure we should be here
            MOVB    #BIT0,TFLG1             ; clear the flag

#ifdef __PRINTTIME_
            PSHX                            ; save reg X
            LDX     #LEFTSPSTR              ; print beam broken
            JSR     OUTSTR                  ;
            PULX                            ; restore X
#endif

            MOVW    UPPERTIMER,LEFTSPDTIME  ; save time beam was broken
            LDAA    SPDTIMEFLG              ; load flags for broken beams
            ORAA    #LSPDFLAG               ; set the left flag
            CMPA    #BSPDFLAG               ; have both beams been broken?
            BEQ     LEFTSTS1                ; yes, then go handle
            STAA    SPDTIMEFLG              ; no, save flags for borken beams
            BRA     LEFTSTSX                ; get out

LEFTSTS1    MOVB    #$00,SPDTIMEFLG         ; clear flags for broken beams
            LDD     LEFTSPDTIME             ; get the left beam broken time
            CPD     RIGHTSPDTIME            ; did the timer roll over
            BLO     LEFTSTS2                ; yes, so calc speed differently
            SUBD    RIGHTSPDTIME            ; calculate time difference
            BRA     LEFTSTS3

LEFTSTS2    LDD     RIGHTSPDTIME            ; load time right beam broken
            SUBD    LEFTSPDTIME             ; calculate time difference

LEFTSTS3    CPD     #SPDLIMIT               ; was the object speeding?
            BHS     LEFTSTS4                ; no, so get out

            MOVB    #LSPDFLAG,SPDCAUGHT     ; yes, set flag for speeder caught
```

```
LEFTSTS4    COMB                            ;
            STAB    SEG7PORT                ; write speed to port
LEFTSTSX    RTI                             ; return from interrupt
*
********************************************************************************
*                 INTERRUPT SERVICE ROUTINE - RIGHTSTS
* Description: Handles the processing for a signal received from the right
*               speed trap sensor.
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
RIGHTSTS    BRCLR   TFLG1,BIT1,RIGHTSTSX    ; make sure we should be here
            MOVB    #BIT1,TFLG1             ; clear the flag

#ifdef __PRINTTIME_
            PSHX                            ; save reg X
            LDX     #RIGHTSPSTR             ; print beam broken
            JSR     OUTSTR                  ;
            PULX                            ; restore reg X
#endif

            MOVW    UPPERTIMER,RIGHTSPDTIME ; save time beam was broken
            LDAA    SPDTIMEFLG              ; load flags for broken beams
            ORAA    #RSPDFLAG               ; set the right flag
            CMPA    #BSPDFLAG               ; have both beams been broken?
            BEQ     RIGHTSTS1               ; yes, then go handle
            STAA    SPDTIMEFLG              ; no, save flags for borken beams
            BRA     RIGHTSTSX               ; get out

RIGHTSTS1   MOVB    #$00,SPDTIMEFLG         ; clear flags for broken beams
            LDD     RIGHTSPDTIME            ; get the right beam broken time
            CPD     LEFTSPDTIME             ; did the timer roll over
            BLO     RIGHTSTS2               ; yes, so calc speed differently
            SUBD    LEFTSPDTIME             ; calculate time difference
            BRA     RIGHTSTS3

RIGHTSTS2   LDD     LEFTSPDTIME             ; load time right beam broken
            SUBD    RIGHTSPDTIME            ; calculate time difference

RIGHTSTS3   CPD     #SPDLIMIT               ; was the object speeding?
            BHS     RIGHTSTS4               ; no, so get out

            MOVB    #RSPDFLAG,SPDCAUGHT     ; yes, set flag for speeder caught (going right)

RIGHTSTS4   COMB                            ;
            STAB    SEG7PORT                ; write speed to port
RIGHTSTSX   RTI                             ; return from interrupt
*
********************************************************************************
*                 INTERRUPT SERVICE ROUTINE - TIMEEXT
* Description: Increments the extended timer when a timer overflow occurs in
*               TCNT (it is incremented every 8ms).
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
TIMEEXT     BRCLR   TFLG2,BIT7,TIMEEXTX     ; make sure we should be here
            MOVB    #BIT7,TFLG2             ; clear the flag
            LDX     UPPERTIMER              ; get the timer extension
            INX                             ; increment the timer extension
            STX     UPPERTIMER              ; save the timer extension

            LDD     SIRENFREQ               ; get previous siren frequency
            BEQ     TIMEEXT2                ; if siren off, keep it off

            ADDD    SIRENTIMER              ; update siren frequency
            STD     SIRENFREQ               ; save siren frequency

            CPD     #SIRENFREQ1             ; are we at low end of range
            BLO     TIMEEXT1                ; no, continue
            MOVW    #SIRENRATE1,SIRENTIMER  ; yes, so start going back up
            BRA     TIMEEXT2                ; continue

TIMEEXT1    CPD     #SIRENFREQ2             ; are we at high end of range
            BHI     TIMEEXT2                ; no, continue
            MOVW    #SIRENRATE2,SIRENTIMER   ; yes, so start going down

TIMEEXT2    LDAB    FLASHPREV               ; get previous status of lights
            BEQ     TIMEEXTX                ; if lights off, keep them off

            LDAA    FLASHTIMER              ; get the flashing lights timer
            INCA                            ; increment the timer
            STAA    FLASHTIMER              ; save flashing lights timer
            CMPA    #FLASHRATE              ; do we need to change light status
            BNE     TIMEEXTX                ; no, so get out

            COMB                            ; switch light pattern
            STAB    FLASHPREV               ; save new light pattern
            STAB    LED1PORT                ; turn on lights with new pattern

            MOVB    #$00,FLASHTIMER         ; reset flashing lights timer

TIMEEXTX    RTI                             ; return from interrupt
*
*******************************************************************************
*                 INTERRUPT SERVICE ROUTINE - UPDMOTORS
* Description: Updates the speed of the motors on every RTI interrupt
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
UPDMOTORS   BRCLR   RTIFLG,BIT7,UPDMOTORSX      ; make sure we should be here
            MOVB    #BIT7,RTIFLG                ; clear the flag

            JSR     LEFTMOTOR                   ; update speed on left motor
            JSR     RIGHTMOTOR                  ; update speed on right motor

UPDMOTORSX  RTI                                 ; return from interrupt
*
*******************************************************************************
*                 INTERRUPT SERVICE ROUTINE - SIREN
* Description: Handles generating the frequency of the siren
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
SIREN       BRCLR   TFLG1,BIT6,SIRENX           ; make sure we should be here
            MOVB    #BIT6,TFLG1                 ; clear the flag

            LDD     TC6                         ; get previous interrupt time
            ADDD    SIRENFREQ                    ; calc next time (set frequency of siren)
            STD     TC6                         ; set next interrupt time

SIRENX      RTI                                 ; return from interrupt
*
*******************************************************************************

* Filename      : WAIT.ASM
* Programmer    : Michael Hattermann
* Date          : March 29, 2002
* Version       : 1.0
* Description   : This file contains the wait and bumper
*                 functions. They must be together because
*                 the bumpers are checked while waiting. The
*                 following functions are available:
*
*                   WAIT - waits for specified # ms
*                   BUMPED - Determines if a bump has occured
*
*#define __DEBUGBUMP_    1
#define __PRINTBUMP_    1

#include "hc12.asm"
*
************************************************************
* Bump/Wait Equates
************************************************************
*
NOBUMPMAX       EQU     $09             ; max value for no bumper pressed
FRONTREAR       EQU     $47             ; division between front and back

*
************************************************************
* Bump/Wait Debug Code
************************************************************
*
#ifdef __DEBUGBUMP_
            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $0900
BUMPVALUE   DC.B    $00             ; bumper value

            ORG     $B000

TEST        LDAA    #$00            ; turn off COP watchdog timer
            STAA    COPCTL

            LDS     #$0bff          ; init the stack pointer

            LDX     #1000           ;
            JSR     WAIT            ;

            JSR     INITATD         ; init A/D system
            JSR     INITSCI         ; init SCI system
            MOVB    #$80,TSCR       ; enable the timer

TEST1       LDX     #500            ; wait 1/2 sec
            JSR     WAIT            ;
            LDAA    BUMPVALUE       ; get bumper value
            TBEQ    A,TEST1         ; if bumper not pressed, keep checking

            BRA     TEST1           ;

TEST2       LDX     #1              ; wait 1ms to time wait routine
            LDY     TCNT            ; get start value of timer
            JSR     WAIT            ; wait
            LDX     TCNT            ; get end value of timer
            JSR     OUTADDR         ; print start value
            LDAA    #$20            ; print blank space
            JSR     OUTCHAR         ;
            TFR     Y,X             ; get end value
            JSR     OUTADDR         ; print end value
            LDX     #NEWLINE        ; print end of line
            JSR     OUTSTR          ;

            BRA     TEST1           ;

NEWLINE     DC.B    CR,LF,EOS

#include "sci.asm"
#include "atd.asm"

#endif
```

```
#ifdef __PRINTBUMP_
BUMPSTR     DC.B    'BUMPER VALUE = '
            DC.B    EOS
#endif

*
********************************************************************************
*                       SUBROUTINE -  WAIT
* Description: Waits for the designated amount of time (in ms).  If a bumper is
*               pressed while waiting, it will quit waiting and returns to the
*               function that called it. Function returns 0 if waited full time,
*               returns bumper value otherwise
* Input         : # of ms to wait in reg X.
* Output        : Bumper value in BUMPVALUE.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
WAIT        TBEQ    X,WAITX         ; if no time to wait, get out
            PSHX                    ; save reg X
            PSHD                    ; save reg D
            MOVB    #$00,BUMPVALUE  ; clear old bumper value

WAIT1       LDAB    #11             ; load loop counter

WAIT2       LDAA    #BUMPER         ; check to see if we have
            JSR     ANALOG          ;   been bumped
            CMPA    #NOBUMPMAX      ; were we bumped?
            BHI     WAITBX          ; yes, get out
            NOP                     ; do nothing
            NOP                     ;
            NOP                     ;
            NOP                     ;
            NOP                     ;
            NOP                     ;
            NOP                     ;
            NOP                     ;
            NOP                     ;
            DBNE    B,WAIT2         ; repeat until counter=0

            DBNE    X,WAIT1         ; if we need to wait more, go wait

WAITX       PULD                    ; restore reg D
            PULX                    ; restore reg X
            RTS                     ; return to caller
WAITBX      STAA    BUMPVALUE       ; save bumper value

#ifdef __PRINTBUMP_
            PSHX                    ; save reg X
            LDX     #BUMPSTR        ; print bump string
            JSR     OUTSTR          ;
            JSR     OUTNUM          ; print analog value
            LDX     #NEWLINE        ; print end of line
            JSR     OUTSTR          ;
            PULX                    ; restore reg X
#endif

            BRA     WAITX           ; get out

********************************************************************************
*                       SUBROUTINE -  BUMPED
* Description: Determines if a bump sensor has been pressed.  If it has, the
*               function will return the value read from the bumper A/D port.
*               If no bumper is pressed, a $00 will be returned.
* Input         : None.
* Output        : Bumper value in reg A.
* Destroys      : Reg A.
* Calls         : None.
********************************************************************************
*
```

```
BUMPED      LDAA    BUMPVALUE           ; get the last bump value
            RTS                         ; return to caller
*
********************************************************************************
*                        SUBROUTINE -  WAIT
* Description: Waits for the designated amount of time (in ms).
* Input         : # of ms to wait in reg X.
* Output        : None.
* Destroys      : None.
* Calls         : None.
********************************************************************************
*
*WAIT         TBEQ    X,WAITX         ; if no time to wait, get out
*             PSHX                    ; save reg X
*             PSHD                    ; save reg D
*
*WAIT1        LDD    #1323            ; load loop counter
*
*WAIT2
*             DBNE    D,WAIT2         ; repeat until counter=0
*
*             DBNE    X,WAIT1         ; if we need to wait more, go wait
*
*WAITX        PULD                    ; restore reg D
*             PULX                    ; restore reg X
*             RTS                     ; return to caller
*
********************************************************************************

* Filename      : OBJAVOID.ASM
* Programmer    : Michael Hattermann
* Date          : February 4, 2002
* Version       : 1.0
* Description   : This file contains the code for
*                   object avoidance. The following
*                   functions are available:
*
*                     OBJAVOID - reads IR and avoids obstacles
*                     GETVALUES - reads IR values from analog port
*                     CONVREACT - converts IR value to a reaction
*                     LEFTRIGHT - decided to turn left,right,or random
*                     BACKUP - backs robot up and turns it
*                     HARD - turns robot hard in a direction
*                     SOFT - turns robot soft in a direction
*
*
*#define __DEBUGOBJAVOID2_ 1
*#define __PRINTOBJAVOID2_ 1
#include "hc12.asm"


*
************************************************************
* Object Avoidance Equates
************************************************************
*
LREQUAL         EQU         $10         ; left,right IR values equality threshold
BACKUPTIME      EQU         150         ; # ms to backup

*
************************************************************
* OBJAVOID2 Debug Code
************************************************************
*
#ifdef __DEBUGOBJAVOID2_
            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $B000
TEST        JSR     INITATD         ; init A/D system
TEST1       LDAA    #HALFSPEED      ; go at half speed
```

```
            JSR     OBJAVOID        ; go avoid objects
            BRA     TEST1
            SWI

LVALUE      DC.B        $00         ; value of left IR
LREACT      DC.B        $00         ; reaction to left IR value
RVALUE      DC.B        $00         ; value of right IR
RREACT      DC.B        $00         ; reaction to right IR value
CVALUE      DC.B        $00         ; value of center IR
CREACT      DC.B        $00         ; reaction to center IR value
PREVRAND    DC.B        $00         ; previous turn direction
PREVREACT   DC.B        $00         ; previous reaction value

#include "atd.asm"
#include "time.asm"
#include "pwm.asm"
#include "sci.asm"
#include "wait.asm"

#endif

#ifdef __PRINTOBJAVOID2_
LEFT        DC.B    'Left IR='
            DC.B    EOS
CENTER      DC.B    ', Center IR='
            DC.B    EOS
RIGHT       DC.B    ', Right IR='
            DC.B    EOS
LEFTR       DC.B    'Left Reaction='
            DC.B    EOS
CENTERR     DC.B    ', Center Reaction='
            DC.B    EOS
RIGHTR      DC.B    ', Right Reaction='
            DC.B    EOS
BACKSTR     DC.B    'BACKUP'
            DC.B    CR,LF,CR,LF,EOS
LHARDSTR    DC.B    'HARD LEFT'
            DC.B    CR,LF,CR,LF,EOS
LSOFTSTR    DC.B    'SOFT LEFT'
            DC.B    CR,LF,CR,LF,EOS
RHARDSTR    DC.B    'HARD RIGHT'
            DC.B    CR,LF,CR,LF,EOS
RSOFTSTR    DC.B    'SOFT RIGHT'
            DC.B    CR,LF,CR,LF,EOS
FOWARDSTR   DC.B    'FOWARD'
            DC.B    CR,LF,CR,LF,EOS
#endif

* Reaction table for values for center IR
************************************************************
CNTRTBL     DC.B        NOACTION        ; $00-$07
            DC.B        NOACTION        ; $08-$0F
            DC.B        NOACTION        ; $10-$17
            DC.B        NOACTION        ; $18-$1F
            DC.B        REACTSOFT       ; $20-$27
            DC.B        REACTSOFT       ; $28-$2F
            DC.B        REACTSOFT       ; $30-$37
            DC.B        REACTSOFT       ; $38-$3F
            DC.B        REACTSOFT       ; $40-$47
            DC.B        REACTHARD       ; $48-$4F
            DC.B        REACTHARD       ; $50-$57
            DC.B        REACTHARD       ; $58-$5F
            DC.B        REACTHARD       ; $60-$67
            DC.B        REACTHARD       ; $68-$6F
            DC.B        REACTBACK       ; $70-$77
            DC.B        REACTBACK       ; $78-$7F
            DC.B        REACTBACK       ; $80-$87
            DC.B        REACTBACK       ; $88-$8F
            DC.B        REACTBACK       ; $90-$97
            DC.B        REACTBACK       ; $98-$9F
```

```
* Reaction table for values for left,right IR
************************************************************
LEFTTBL     DC.B        NOACTION        ; $00-$07
            DC.B        NOACTION        ; $08-$0F
            DC.B        NOACTION        ; $10-$17
            DC.B        NOACTION        ; $18-$1F
            DC.B        REACTSOFT       ; $20-$27
            DC.B        REACTSOFT       ; $28-$2F
            DC.B        REACTSOFT       ; $30-$37
            DC.B        REACTSOFT       ; $38-$3F
            DC.B        REACTHARD       ; $40-$47
            DC.B        REACTHARD       ; $48-$4F
            DC.B        REACTHARD       ; $50-$57
            DC.B        REACTHARD       ; $58-$5F
            DC.B        REACTHARD       ; $60-$67
            DC.B        REACTHARD       ; $68-$6F
            DC.B        REACTBACK       ; $70-$77
            DC.B        REACTBACK       ; $78-$7F
            DC.B        REACTBACK       ; $80-$87
            DC.B        REACTBACK       ; $88-$8F
            DC.B        REACTBACK       ; $90-$97
            DC.B        REACTBACK       ; $98-$9F

* Reaction table for values for left,right IR
************************************************************
RIGHTTBL    DC.B        NOACTION        ; $00-$07
            DC.B        NOACTION        ; $08-$0F
            DC.B        NOACTION        ; $10-$17
            DC.B        NOACTION        ; $18-$1F
            DC.B        REACTSOFT       ; $20-$27
            DC.B        REACTSOFT       ; $28-$2F
            DC.B        REACTSOFT       ; $30-$37
            DC.B        REACTSOFT       ; $38-$3F
            DC.B        REACTHARD       ; $40-$47
            DC.B        REACTHARD       ; $48-$4F
            DC.B        REACTHARD       ; $50-$57
            DC.B        REACTHARD       ; $58-$5F
            DC.B        REACTHARD       ; $60-$67
            DC.B        REACTHARD       ; $68-$6F
            DC.B        REACTBACK       ; $70-$77
            DC.B        REACTBACK       ; $78-$7F
            DC.B        REACTBACK       ; $80-$87
            DC.B        REACTBACK       ; $88-$8F
            DC.B        REACTBACK       ; $90-$97
            DC.B        REACTBACK       ; $98-$9F

*
*******************************************************************************
*                       SUBROUTINE -  OBJAVOID
* Description: Performs an obstacle avoidance behavior by reading the values
*               from the IR and bump sensors and moving the robot accordingly.
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : GETVALUES,CONVREACT.
*******************************************************************************
*
OBJAVOID    PSHD                    ; save reg D
            PSHX                    ; save reg X

            JSR     BUMPED          ; check to see if we bumped something
            CMPA    #FRONTREAR      ; did we bump something in the front?
            BLO     OBJAVOIDB       ; no bump, check the IR
            JMP     BACKUP          ; we bumped something, backup

OBJAVOIDB   JSR     GETVALUES       ; get the IR readings
            JSR     CONVREACT       ; convert readings to reactions

            LDAB    #REACTBACK      ; get code for back up reaction
            CMPB    CREACT          ; does center say backup
            BNE     OBJAVOID1       ; no, continue checking
            JMP     BACKUP          ; yes, backup

OBJAVOID1   CMPB    LREACT          ; does left say backup
            BNE     OBJAVOID2       ; no, continue checking
            JMP     BACKUP          ; yes, backup

OBJAVOID2   CMPB    RREACT          ; does right say backup
            BNE     OBJAVOID3       ; no, continue checking
            JMP     BACKUP          ; yes, backup

OBJAVOID3   LDAB    #REACTHARD      ; get code for hard turn reaction
            CMPB    CREACT          ; does center say backup
            BNE     OBJAVOID4       ; no, continue checking
            JMP     HARD            ; yes, turn hard

OBJAVOID4   CMPB    LREACT          ; does left say turn hard
            BNE     OBJAVOID5       ; no, continue checking
            JMP     HARD            ; yes, turn hard

OBJAVOID5   CMPB    RREACT          ; does right say turn hard
            BNE     OBJAVOID6       ; no, continue checking
            JMP     HARD            ; yes, turn hard

OBJAVOID6   LDAB    #REACTSOFT      ; get code for hard turn reaction
            CMPB    CREACT          ; does center say backup
            BNE     OBJAVOID7       ; no, continue checking
            JMP     SOFT            ; yes, turn soft

OBJAVOID7   CMPB    LREACT          ; does left say turn hard
            BNE     OBJAVOID8       ; no, continue checking
            JMP     SOFT            ; yes, turn soft

OBJAVOID8   CMPB    RREACT          ; does right say turn hard
            BNE     OBJAVOID9       ; no, continue checking
            JMP     SOFT            ; yes, turn hard

OBJAVOID9
#ifdef __PRINTOBJAVOID2_
            PSHX                    ; save reg X
            LDX     #FOWARDSTR      ; print foward string
            JSR     OUTSTR          ;
            PULX                    ; restore reg X
#endif

OBJAVOID10  LDX     #GOFOWARD               ; no obstacles, go foward at set speed
            JSR     STEER                   ;
            CLR     PREVRAND                ; clear previous random direction
            MOVB    #NOACTION,PREVREACT     ; save this reaction

OBJAVOIDX   PULX                    ; restore reg X
            PULD                    ; restore reg D
            RTS                     ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  GETVALUES
* Description: Gets the IR values for the left,center,and right channels
* Input        : None.
* Output       : RVALUE,LVALUE,CVALUE.
* Destroys     : RVALUE,LVALUE,CVALUE.
* Calls        : ANALOG.
********************************************************************************
*
GETVALUES   PSHA                    ; save reg A
            LDAA    #RIGHTIR        ; read right IR value
            JSR     ANALOG          ;
            STAA    RVALUE          ; save right IR value
            LDAA    #LEFTIR         ; read left IR value
            JSR     ANALOG          ;
            STAA    LVALUE          ; save left IR value
            LDAA    #CENTERIR       ; read center IR value
            JSR     ANALOG          ;
            STAA    CVALUE          ; save center IR value

#ifdef __PRINTOBJAVOID2_
            PSHX                    ; save reg X
            LDX     #LEFT           ; print left IR header
            JSR     OUTSTR          ;
            LDAA    LVALUE          ; print left IR value
            JSR     OUTNUM          ;
            LDX     #CENTER         ; print center IR header
            JSR     OUTSTR          ;
            LDAA    CVALUE          ; print center IR value
            JSR     OUTNUM          ;
            LDX     #RIGHT          ; print right IR header
            JSR     OUTSTR          ;
            LDAA    RVALUE          ; print right IR value
            JSR     OUTNUM          ;
            LDX     #NEWLINE        ; print new line
            JSR     OUTSTR          ;
            PULX                    ; restore reg X
#endif

            PULA                    ; restore reg A
            RTS                     ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  CONVREACT
* Description: Converts IR readings to reaction values using lookup tables
* Input         : RVALUE,LVALUE,CVALUE.
* Output        : CREACT,LREACT,RREACT.
* Destroys      : CREACT,LREACT,RREACT.
* Calls         : None.
********************************************************************************
*
CONVREACT   PSHX                            ; save register X
            LDX     #CNTRTBL                ; load address of lookup table
            LDAB    CVALUE                  ; get center value
            LSRB                            ; convert center value
            LSRB                            ;  to table lookup
            LSRB                            ;   value
            MOVB    B,X,CREACT              ; lookup reaction for center

            LDX     #LEFTTBL                ; load address of lookup table
            LDAB    LVALUE                  ; get left value
            LSRB                            ; convert left value
            LSRB                            ;  to table lookup
            LSRB                            ;   value
            MOVB    B,X,LREACT              ; lookup reaction for left channel

            LDX     #RIGHTTBL               ; load address of lookup table
            LDAB    RVALUE                  ; get right value
            LSRB                            ; convert right value
            LSRB                            ;  to table lookup
            LSRB                            ;   value
            MOVB    B,X,RREACT              ; lookup reaction for right channel

#ifdef __PRINTOBJAVOID2_
            PSHX                            ; save reg X
            LDX     #LEFTR                  ; print left IR header
            JSR     OUTSTR                  ;
            LDAA    LREACT                  ; print left IR reaction
            JSR     OUTNUM                  ;
            LDX     #CENTERR                ; print center IR header
            JSR     OUTSTR                  ;
            LDAA    CREACT                  ; print center IR reaction
            JSR     OUTNUM                  ;
            LDX     #RIGHTR                 ; print right IR header
            JSR     OUTSTR                  ;
            LDAA    RREACT                  ; print right IR reaction
            JSR     OUTNUM                  ;
            LDX     #NEWLINE                ; print new line
            JSR     OUTSTR                  ;
            PULX                            ; restore reg X
#endif
            PULX                            ; restore register X
            RTS                             ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  LEFTRIGHT
* Description: Decides if we should turn left or right based on IR sensor
*               readings. It will compare left and right values, and if they
*               differ by more than some threshold, this decides the turn
*               direction.  Otherwise, turn direction is random.  Return values:
*                   REACTLEFT - Turn left
*                   REACTRIGHT - Turn right
* Input         : LVALUE,RVALUE.
* Output        : Reg B has direction.
* Destroys      : Reg B, PREVRAND.
* Calls         : None.
*******************************************************************************
*
LEFTRIGHT   PSHA                        ; save reg A

            LDAA    LVALUE              ; get the left value
            LDAB    RVALUE              ; get the right value
            SBA                         ; compare the values
            CMPA    #LREQUAL            ; if left > right by the threshold
            BGE     LEFTRIGHT1          ; turn right
            CMPA    #(-LREQUAL)         ; if left < right by the threshold
            BLE     LEFTRIGHT2          ; turn left

            TST     PREVRAND            ; do we have a previous direction
            BEQ     LEFTRIGHTR          ; no, so go generate a direction
            LDAB    PREVRAND            ; yes, so use previous direction
            BRA     LEFTRIGHTX          ; get out

LEFTRIGHTR  LDAB    TCNTL               ; otherwise, get lower half of timer
            LSRB                        ; check lowest bit
            BCS     LEFTRIGHT2          ; go left if set, right if clear
LEFTRIGHT1  LDAB    #REACTRIGHT         ; turn right
            STAB    PREVRAND            ; save previous direction
            BRA     LEFTRIGHTX          ; get out
LEFTRIGHT2  LDAB    #REACTLEFT          ; turn left
            STAB    PREVRAND            ; save previous direction
LEFTRIGHTX  PULA                        ; restore reg A
            RTS                         ; return to caller
*
*******************************************************************************
*                       SUBROUTINE -  BACKUP
* Description: Backs up and turns a random direction (left or right).
* Input         : None.
* Output        : None.
* Destroys      : Reg B,Reg X,PREVRAND.
* Calls         : STEER,WAIT.
*******************************************************************************
*
BACKUP
            MOVB    #REACTBACK,PREVREACT    ; save this reaction
            CLR     PREVRAND                ; clear previous random direction

#ifdef __PRINTOBJAVOID2_
            PSHX
            LDX     #BACKSTR
            JSR     OUTSTR
            PULX
#endif

BACKUP1     JSR     LEFTRIGHT       ; do we have a preference left or right
            CMPB    #REACTLEFT      ; if we need to go left, go left
            BEQ     BACKUPL         ; go left

            LDX     #BACKRIGHT      ; go hard right
            JSR     STEER           ;
            BRA     BACKUPX         ; get out

BACKUPL     LDX     #BACKLEFT       ; go hard left
            JSR     STEER           ;

BACKUPX     LDX     #GOBACK         ; go backward
            JSR     STEER           ;
            LDX     #BACKUPTIME     ; go back for set amount of time
            JSR     WAIT            ;
            JMP     OBJAVOIDX       ; get out
*
********************************************************************************
*                       SUBROUTINE -  HARD
* Description: Turns hard in a random direction (left or right).
* Input         : None.
* Output        : None.
* Destroys      : Reg B,Reg X.
* Calls         : STEER.
********************************************************************************
*
HARD
            LDAB    PREVREACT               ; get previous reaction
            CMPB    #REACTHARD              ; was it react hard?
            BEQ     HARD1                   ; yes, so continue
            CLR     PREVRAND                ; no, clear previous turn direction
            MOVB    #REACTHARD,PREVREACT    ; save this reaction
HARD1
            JSR     LEFTRIGHT               ; do we have a preference left or right
            CMPB    #REACTLEFT              ; if we need to go left, go left
            BEQ     HARDL                   ; go left

#ifdef __PRINTOBJAVOID2_
            PSHX                            ; save reg X
            LDX     #RHARDSTR               ; print hard right string
            JSR     OUTSTR                  ;
            PULX                            ; restore reg X
#endif

HARDR       LDX     #HARDRIGHT              ; go hard right
            JSR     STEER                   ;
            JMP     OBJAVOIDX               ; get out

HARDL

#ifdef __PRINTOBJAVOID2_
            PSHX                            ; save reg X
            LDX     #LHARDSTR               ; print left hard string
            JSR     OUTSTR                  ;
            PULX                            ; restore reg X
#endif

            LDX     #HARDLEFT               ; go hard left
            JSR     STEER                   ;
            JMP     OBJAVOIDX               ; get out
*
********************************************************************************
*                       SUBROUTINE -  SOFT
* Description: Turns soft in a random direction (left or right).
* Input         : None.
* Output        : None.
* Destroys      : Reg B,Reg X.
* Calls         : STEER,WAIT.
********************************************************************************
*
SOFT
            LDAB    PREVREACT               ; get previous reaction
            CMPB    #REACTSOFT              ; was it react soft?
            BEQ     SOFT1                   ; yes, so continue
            CLR     PREVRAND                ; no, clear previous turn direction
            MOVB    #REACTSOFT,PREVREACT    ; save this reaction
SOFT1
```

```
            JSR     LEFTRIGHT       ; do we have a preference left or right
            CMPB    #REACTLEFT      ; if we need to go left, go left
            BEQ     SOFTL           ; go left

#ifdef __PRINTOBJAVOID2_
            PSHX                    ; save reg X
            LDX     #RSOFTSTR       ; print soft right string
            JSR     OUTSTR          ;
            PULX                    ; restore reg X
#endif

SOFTR       LDX     #SOFTRIGHT      ; go soft right
            JSR     STEER           ;
            JMP     OBJAVOIDX       ; get out

SOFTL
#ifdef __PRINTOBJAVOID2_
            PSHX                    ; save reg X
            LDX     #LSOFTSTR       ; print soft left string
            JSR     OUTSTR          ;
            PULX                    ; restore reg X
#endif

            LDX     #SOFTLEFT       ; go soft left
            JSR     STEER           ;
            JMP     OBJAVOIDX       ; get out
*
*******************************************************************************

* Filename      : FOLLOW.ASM
* Programmer    : Michael Hattermann
* Date          : March 31, 2002
* Version       : 1.0
* Description   : This file contains the code for
*                 object following. The following
*                 functions are available:
*
*
*
*#define __DEBUGFOLLOW_ 1
*#define __PRINTFOLLOW_  1

#include "hc12.asm"

*
************************************************************
* Object Following Equates
************************************************************
*
MINFOLLOW       EQU     $55         ; minimum reading on sensor to execute following
OBJAVOIDSPD     EQU     HALFSPEED   ; speed to perform object avoidance at
MINFOLLOWSPD    EQU     HALFSPEED   ; minimum speed to follow at
MAXFOLLOWSPD    EQU     FULLSPEED   ; maximum speed to follow at
FOLLOWTURNSPD   EQU     _5_8_SPEED  ; max speed to perform a hard turn at
NUMAVGCNT       EQU     20          ; number of sensor values to average
SPDACCEL        EQU     1           ; desired amount of acceleration
SPDDOWNACCEL    EQU     -2          ; slow down acceleration
SPDUPACCEL      EQU     2           ; speed up acceleration
*
************************************************************
* Object Following Debug Code
************************************************************
*
#ifdef __DEBUGFOLLOW_
            ORG     USERPROG_PVECT
            JMP     TEST

            ORG     $B000
TEST        LDAA    #$00            ; turn off COP watchdog timer
            STAA    COPCTL
            LDS     #$0bff          ; init the stack pointer
```

```
HERE        BRA     HERE            ; end of test program

CFVALUE     DC.B    $00             ; center follow sensor value
LFVALUE     DC.B    $00             ; left follow sensor value
RFVALUE     DC.B    $00             ; right follow sensor value
FTBLIDX     DC.B    $00             ; index into reaction table
LASTFOLLOW  DC.B    $00             ;  last follow direction

OLDCFAVG    DC.W    $0000           ; old average of center follow sensor values
NEWCFAVG    DC.W    $0000           ; new average of center follow sensor values
AVGCNT      DC.B    $0000           ; number of items in new average

#include "atd.asm"
#include "pwm.asm"
#endif

* Reaction table for following        (Center|Left|Right)
************************************************************
FOLLOWTBL   DC.W    FOBJAVOID       ; 000 - Object avoidance
            DC.W    FHARDR          ; 001 - Hard right
            DC.W    FHARDL          ; 010 - Hard left
            DC.W    FCONT           ; 011 - ERROR - do what we did last
            DC.W    FFOWARD         ; 100 - Foward
            DC.W    FSOFTR          ; 101 - Soft right
            DC.W    FSOFTL          ; 110 - Soft left
            DC.W    FCONT           ; 111 - ERROR - do what we did last

*
*******************************************************************************
*                       SUBROUTINE -  FOLLOW
* Description: Performs obstacle following behavior by reading the values
*               from the IR sensors and moving the robot accordingly.
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
FOLLOW      PSHX                        ; save register X
            PSHD                        ; save register D

*            JSR     BUMPED              ; check to see if we bumped something
*            TBEQ    A,FOLLOW1           ; no bump, continue following behavior
*            JMP     MAINOUT             ; we bumped something, quit program

FOLLOW1     JSR     FGETDATA            ; get data from following sensors

#ifdef __PRINTFOLLOW_
            PSHD                        ; save reg D
            PSHX                        ; save reg X
            LDAA    LFVALUE             ; print left value
            JSR     OUTNUM
            LDAA    #$20                ; print space
            JSR     OUTCHAR
            LDAA    CFVALUE             ; print center value
            JSR     OUTNUM
            LDAA    #$20                ; print space
            JSR     OUTCHAR
            LDAA    RFVALUE             ; print right value
            JSR     OUTNUM
            LDX     #NEWLINE            ; print new line
            JSR     OUTSTR
            PULX                        ; restore reg X
            PULD                        ; restore reg D
#endif

FOLLOW2
            LDX     #FOLLOWTBL          ; load address of reaction table
            LDAA    FTBLIDX             ; get the reaction table index
            LSLA                        ; convert to 16-bit index
            LDX     A,X                 ; get address of handling routine
            JMP     0,X                 ; jump to appropriate routine

FOLLOWX     PULD                        ; restore register D
            PULX                        ; restore register X
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINE -  FGETDATA
* Description: Reads the values from the following IR detectors.
* Input         : None.
* Output        : LFVALUE,RFVALUE.
* Destroys      : LFVALUE,RFVALUE.
* Calls         : None.
********************************************************************************
*
FGETDATA    PSHA                        ; save register A
            MOVB    #$00,FTBLIDX        ; clear reaction table index

            LDAA    #CENTERFOLLOW       ; get data from center sensor
            JSR     ANALOG              ;
            STAA    CFVALUE             ; save center sensor value
            CMPA    #MINFOLLOW          ; did center see speeding car?
            BLO     FGETDATA1           ; no, so continue
            BSET    FTBLIDX,BIT2        ; yes, so set bit in index

FGETDATA1   LDAA    #LEFTFOLLOW         ; get data from left sensor
            JSR     ANALOG              ;
            STAA    LFVALUE             ; save left sensor value
            CMPA    #MINFOLLOW          ; did left see speeding car?
            BLO     FGETDATA2           ; no, so continue
            BSET    FTBLIDX,BIT1        ; yes, so set bit in index

FGETDATA2   LDAA    #RIGHTFOLLOW        ; get data from right sensor
            JSR     ANALOG              ;
            STAA    RFVALUE             ; save right sensor value
            CMPA    #MINFOLLOW          ; did right see speeding car?
            BLO     FGETDATAX           ; no, so get out
            BSET    FTBLIDX,BIT0        ; yes, so set bit in index

FGETDATAX   PULA                        ; restore register A
            RTS                         ; return to caller
*
********************************************************************************
*                       SUBROUTINES -  FHARDR,FHARDL,FFOWARD,FSOFTR,FSOFTL,FCONT
* Description: Handle motor control for following behavior
* Input         : None.
* Output        : None.
* Destroys      : Reg X, Reg D.
* Calls         : STEER.
********************************************************************************
*
FHARDR      MOVB    FTBLIDX,LASTFOLLOW  ; save off last table index
            LDD     #FOLLOWTURNSPD      ; set speed to turning speed
            JSR     CHNGSPEED           ;
            LDX     #HARDRIGHT          ; turn hard right
            JSR     STEER               ;
            JMP     FOLLOWX             ; get out

FHARDL      MOVB    FTBLIDX,LASTFOLLOW  ; save off last table index
            LDD     #FOLLOWTURNSPD      ; set speed to turning speed
            JSR     CHNGSPEED           ;
            LDX     #HARDLEFT           ; turn hard left
            JSR     STEER               ;
            JMP     FOLLOWX             ; get out

FFOWARD     MOVB    FTBLIDX,LASTFOLLOW  ; save off last table index
            JSR     SPEEDCALC           ; go set new speed
            LDX     #GOFOWARD           ; go foward
            JSR     STEER               ;
            JMP     FOLLOWX             ; get out
```

```
FSOFTR      MOVB    FTBLIDX,LASTFOLLOW  ; save off last table index
            JSR     SPEEDCALC           ; go set new speed
            LDX     #SOFTRIGHT          ; turn soft right
            JSR     STEER               ;
            JMP     FOLLOWX             ; get out

FSOFTL      MOVB    FTBLIDX,LASTFOLLOW  ; save off last table index
            JSR     SPEEDCALC           ; go set new speed
            LDX     #SOFTLEFT           ; turn soft left
            JSR     STEER               ;
            JMP     FOLLOWX             ; get out

FCONT       LDX     #FOLLOWTBL          ; load address of reaction table
            LDAA    LASTFOLLOW          ; get the last reaction index
            STAA    FTBLIDX             ; save as current reaction
            LSLA                        ; convert to 16-bit index
            LDX     A,X                 ; get address of handling routine
            JMP     0,X                 ; jump to appropriate routine

*
*******************************************************************************
*                       SUBROUTINE -  FOBJAVOID
* Description: Handles object avoidance for following behavior
* Input         : None.
* Output        : None.
* Destroys      : Reg X, Reg D.
* Calls         : SIRENON,SIRENOFF,OBJAVOID,WAIT.
*******************************************************************************
*
FOBJAVOID   JSR     SIRENOFF            ; turn off siren to indicate we lost speeder
            LDD     #OBJAVOIDSPD        ; set motor speed to obj avoid speed
            JSR     CHNGSPEED           ;

FOBJAVOID1  JSR     OBJAVOID            ; avoid obstacles
            LDX     #OAPROCRATE         ; wait designated amount of time
            JSR     WAIT                ;

            JSR     FGETDATA            ; check following sensors
            TST     FTBLIDX             ; did we find the speeder?
            BEQ     FOBJAVOID1          ; no, continue object avoidance

            JSR     SIRENON             ; turn siren on to indicate active chase

            LDX     #FOLLOWTBL          ; load address of reaction table
            LDAA    FTBLIDX             ; get the reaction table index
            LSLA                        ; convert to 16-bit index
            LDX     A,X                 ; get address of handling routine
            JMP     0,X                 ; jump to appropriate routine
*
*******************************************************************************
*                       SUBROUTINE -  SPEEDCALC
* Description: Calculates and sets the next max motor speed so that the robot
*               has a constant acceleration
* Input         : None.
* Output        : None.
* Destroys      : Reg X, Reg D.
* Calls         : .
*******************************************************************************
*
SPEEDCALC   LDD     NEWCFAVG            ; load the current working average
            ADDB    CFVALUE             ; add the newest center sensor
            ADCA    #$00                ; to the current average
            LDX     AVGCNT              ; get the count of # items in working average
            INX                         ; increment the count
            STX     AVGCNT              ; save the count
            CPX     #NUMAVGCNT          ; do we have the correct sum yet?
            BLT     SPEEDCALCX          ; no, get out

            IDIVS                       ; calculate average
            XGDX                        ; get average
            STD     NEWCFAVG            ; save average
```

```
#ifdef __PRINTFOLLOW_
            PSHX                        ; save reg X
            LDX     OLDCFAVG            ; print old average
            JSR     OUTADDR             ;
            LDAA    #$20                ; print a space
            JSR     OUTCHAR             ;
            LDX     NEWCFAVG            ; print new average
            JSR     OUTADDR             ;
            LDX     #NEWLINE            ; print newline
            JSR     OUTSTR              ;
            PULX                        ; restore reg X
#endif

            SUBD    OLDCFAVG            ; calc difference between old,new average
            CPD     #SPDACCEL           ; compare to the desired acceleration
            BEQ     SPEEDCALC4          ; they are equal, do nothing
            BLT     SPEEDCALC2          ; need to speed up
* SLOW DOWN
            LDD     MAXFOWARDSPD        ; get the current max speed
            ADDD    #SPDDOWNACCEL       ; slow it down
            CPD     #MINFOLLOWSPD       ; have we slowed down too far?
            BGE     SPEEDCALC1          ; no, skip adjustment
            LDD     #MINFOLLOWSPD       ; make speed = minimum speed
SPEEDCALC1  JSR     CHNGSPEED           ; set the new speed
            BRA     SPEEDCALC4          ; get out

* SPEED UP
SPEEDCALC2  LDD     MAXFOWARDSPD        ; get the current max speed
            ADDD    #SPDUPACCEL         ; speed it up
            CPD     #MAXFOLLOWSPD       ; have we speed up too far?
            BLE     SPEEDCALC3          ; no, skip adjustment
            LDD     #MAXFOLLOWSPD       ; make speed = maximum speed
SPEEDCALC3  JSR     CHNGSPEED           ; set the new speed

SPEEDCALC4  MOVW    NEWCFAVG,OLDCFAVG   ; make new average the old average
            MOVW    #$0000, NEWCFAVG    ; clear new average to start over
            MOVW    #$0000,AVGCNT       ; clear average count for next time

SPEEDCALCX  RTS                         ; return to caller
*
********************************************************************************

* Filename      : MAIN.ASM
* Programmer    : Michael Hattermann
* Date          : February 22, 2002
* Version       : 1.0
* Description   : This file contains the main routine to
*                   control the rest of the robot.  It
*                   includes all the other control files.
*                   The followong functions are available:
*
*                   MAIN - start of program, inits/uses systems
*                   MAINOUT - end of program, kills systems
*

#include "hc12.asm"

*
************************************************************
* Main Equates
************************************************************
*
PROGSTART       EQU         $B000       ; start of the program
STACKPTR        EQU         $0A00       ; bottom of internal RAM for stack
GLBLVARS        EQU         $0900       ; top of internal RAM for global variables

OAPROCRATE      EQU         10          ; how often to execute object avoidance (in ms)
FPROCRATE       EQU         10          ; how often to execute following (in ms)
*
************************************************************
```

```
* Global Variables
************************************************************
*
                ORG     GLBLVARS
UPPERTIMER      DC.W    $0000           ; 16-bit extension of TCNT
LEFTSPDTIME     DC.W    $0000           ; Time left break beam was broken
RIGHTSPDTIME    DC.W    $0000           ; Time right break beam was broken
SPDTIMEFLG      DC.B    $00             ; Flags to indicate which beams were broken
SPDCAUGHT       DC.B    $00             ; Flags to indicate speeder caught
FLASHTIMER      DC.B    $00             ; Timer for flashing lights
FLASHPREV       DC.B    $00             ; Previous status of lights
SIRENFREQ       DC.W    $0000           ; Frequency of the siren
SIRENTIMER      DC.W    $0000           ; Timer for siren

OLEFTSPD        DC.W    $0000           ; current speed of left motor
NLEFTSPD        DC.W    $0000           ; next speed of left motor
ORIGHTSPD       DC.W    $0000           ; current speed of right motor
NRIGHTSPD       DC.W    $0000           ; next speed of right motor

MAXFOWARDSPD    DC.W    $0000           ; maximum foward speed for motors
MAXBACKSPD      DC.W    $0000           ; maximum reverse speed for motors
LVALUE          DC.B    $00             ; value of left IR
LREACT          DC.B    $00             ; reaction to left IR value
RVALUE          DC.B    $00             ; value of right IR
RREACT          DC.B    $00             ; reaction to right IR value
CVALUE          DC.B    $00             ; value of center IR
CREACT          DC.B    $00             ; reaction to center IR value
PREVRAND        DC.B    $00             ; previous random turning direction
PREVREACT       DC.B    $00             ; previous reaction value
CURRMAN         DC.W    $0000           ; current manuever being performed

LFVALUE         DC.B    $00             ; left follow sensor value
CFVALUE         DC.B    $00             ; center follow sensor value
RFVALUE         DC.B    $00             ; right follow sensor value
FTBLIDX         DC.B    $00             ; index into reaction table
LASTFOLLOW      DC.B    $00             ; last follow direction
OLDCFAVG        DC.W    $0000           ; old average of center follow sensor values
NEWCFAVG        DC.W    $0000           ; new average of center follow sensor values
AVGCNT          DC.B    $0000           ; number of items in new average

BUMPVALUE       DC.B    $00             ; A/D bumper value from wait function

*
************************************************************
* Pseudointerrupt vectors
************************************************************
*
            ORG     T0_PVECT
            JMP     LEFTSTS

            ORG     T1_PVECT
            JMP     RIGHTSTS

            ORG     TMR_OVER_PVECT
            JMP     TIMEEXT

            ORG     RTI_PVECT
            JMP     UPDMOTORS

            ORG     T6_PVECT
            JMP     SIREN

            ORG     USERPROG_PVECT
            JMP     MAIN

            ORG     PROGSTART
*
************************************************************
* Main Constants
************************************************************
*
```

```
WELCOME     DC.B    CR,LF
            DC.B    'STEVE - Speed Trap Enforcement VehiclE'
            DC.B    CR,LF
            DC.B    'Michael Hattermann'
            DC.B    CR,LF
            DC.B    'IMDL - Spring 2002'
            DC.B    CR,LF,EOS
SHUTDOWN    DC.B    CR,LF
            DC.B    'STEVE - Program ended...shutting down systems'
            DC.B    CR,LF,EOS
LEFTIRSTR   DC.B    'Left IR value = '
            DC.B    EOS
RIGHTIRSTR  DC.B    'Right IR value = '
            DC.B    EOS
NEWLINE     DC.B    CR,LF,EOS
LEFTSPSTR   DC.B    'Left beam broken'
            DC.B    CR,LF,EOS
RIGHTSPSTR  DC.B    'Right beam broken'
            DC.B    CR,LF,EOS
SPEED       DC.B    CR,LF
            DC.B    'SPEED = '
            DC.B    EOS


*
*******************************************************************************
*                       SUBROUTINE -  MAIN
* Description: Main program.  Inits the robots sub-systems and begins the robot
*               behavior code.
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
MAIN        MOVB    #$00,COPCTL         ; turn off COP watchdog timer
            LDS     #STACKPTR           ; load stack pointer

            JSR     INITSCI             ; init SCI system
            JSR     INITATD             ; init A/D system
            JSR     INITTIME            ; init timer system
            MOVB    #$00,LED1PORT       ; turn off flashing lights
            MOVB    #$00,SEG7PORT       ; reset speed capture to zero

            LDX     #WELCOME            ; print welcome message
            JSR     OUTSTR              ;

            CLI                         ; turn on interrupts

            JSR     WAITSPEED           ; wait for a speeder
            JSR     INITPWM             ; init PWM system
            JSR     FLASHON             ; turn on flashing lights
            JSR     SIRENON             ; turn on siren
            JSR     PULLOUT             ; pull out and prepare to follow

MAIN2
            JSR     FOLLOW              ; follow speeder
            LDX     #FPROCRATE          ; wait designated amount of time
            JSR     WAIT                ;
            BRA     MAIN2               ; do it again

*
*******************************************************************************
*                       SUBROUTINE -  MAINOUT
* Description: Exits the program.  Stops motors, signals end of program with
*               the lights, then shuts down the subsystems and enters a
*               never ending loop
* Input         : None.
* Output        : None.
* Destroys      : None.
* Calls         : STEER,OUTSTR,WAIT,KILLATD,KILLPWM,KILLTIME.
```

```
*******************************************************************************
*
MAINOUT     LDX     #STOP               ; stop the motors
            JSR     STEER               ;
            LDX     #SHUTDOWN           ; print shutdown message
            JSR     OUTSTR              ;

            LDAA    #8                  ; load loop counter
MAINOUT1    MOVB    #$3C,LED1PORT       ; turn off lights
            JSR     SIRENOFF            ; turn off siren
            LDX     #250                ; wait
            JSR     WAIT                ;
            MOVB    #$C3,LED1PORT       ; turn on lights
            JSR     SIRENON             ; turn on siren
            LDX     #250                ; wait
            JSR     WAIT                ;
            DBNE    A,MAINOUT1          ; continue looping until done
            MOVB    #$00,LED1PORT       ; turn on lights

            JSR     KILLATD             ; shutdown atd system
            JSR     KILLPWM             ; shutdown pwm system
            JSR     KILLTIME            ; shutdown timer system
            MOVB    #$00,LED1PORT       ; turn off flashing lights
            MOVB    #$00,SEG7PORT       ; clear 7 segment display
MAINOUTX    BRA     MAINOUTX            ; end of program
*
*******************************************************************************

#include "wait.asm"
#include "sci.asm"
#include "atd.asm"
#include "time.asm"
#include "pwm.asm"
#include "objavoid2.asm"
#include "follow.asm"
```