EDGAR

The Empty Drinking Glass Autonomous Retriever

By: Matt Cousins
EEL 5666 – Intelligent Machines Design Lab
Spring 2002

Instructors: Dr. Arroyo & Dr. Schwartz
TA's: Aamir Qaiyumi & Uriel Rodriguez

Table of Contents

Abstract

The purpose of EDGAR is to create an autonomous mobile agent that retrieves empty drinking glasses or bottles from people. The robot roams around avoiding obstacles and searching for humans. Once a human is found, the robot approaches the human and stops at its feet. It then waits for the human to place the bottle inside the robot. Once this happens, the robot returns the bottle to the desired location and waits there until the bottle is removed. If the human does not place a bottle inside the robot, it turns and searches for other humans.

Executive Summary

EDGAR was designed to be an aid in most types of social gatherings where drinks are served. It was aimed at helping people dispose of their empty bottles without having to worry about where to put the bottle when they are finished with it.

EDGAR locates humans and approaches them, pausing at their feet. When an bottle is place inside the robot, it takes the bottle back to an IR beacon located somewhere in the room. Once it has arrived at the beacon, it waits for the bottle to be removed and searches for more humans. If no bottle is placed inside the robot, it turns and searches for other humans.

The optimal room size for EDGAR is approximately 30 feet by 30 feet. This constraint is due to the range of the IR beacon that the bottle is returned to. It is not as efficient at finding the beacon outside of this range because it randomly roams around looking for a beacon signal. Once a signal is acquired, it stops roaming and goes towards the beacon.

EDGAR is powered by 8 NiCd AA batteries and motion is obtained with two hacked servos and two 1.5 inch carriage bolts for balance. It is controlled with the Motorola M68HC11 microprocessor mounted on Mekatronix's MTJPRO board.

EDGAR uses 5 different types of sensors to complete its task. Two Sharp GP2D12 IR emitter/detector pairs and three bump switches are used for obstacle avoidance and human encounter detection. Two analog-hacked Sharp IR detectors are used for IR beacon location. A CdS cell is used in conjunction with two ultra-bright red LED's to detect when a bottle is placed inside the robot. A pryo-electric IR motion sensor is used to detect the presence of humans.

Introduction

The worst part of hosting a social gathering is cleaning up afterwards. When the party is over, very few people want to stay and help pick up all the empty bottles and cups left behind by all the guests. The host of the party usually completes this task. This is where EDGAR can be used. While the party is taking place, EDGAR will retrieve people's empty bottles and take them back to a desired location. When the party is over, all of the empty bottles will already be taken care of, making the clean-up process much easier and less time consuming.

This paper will discuss EDGAR's integrated system, mobile platform, actuation, sensors, and its experimental layout and results. It will describe the complete organization of the system as well as functional and data descriptions.

Integrated System

The M68HC11 microprocessor from Motorola is used to control EDGAR. It is mounted on the MTJPRO board purchased from Mekatronix (www.mekatronix.com). The A/D system is used to gather all of the sensor readings on EDGAR. PORTE is used to control the motors and the servo. Other input and output ports are used to control the feedback LED's, mode of operation (download vs. run), and read bump switch data.

The object avoiding IR sensors are placed at the top of the robot as to prevent the robot from running into an overhang, while the bump switches integrated with bumper is on the bottom platform to detect objects lower than the IR sensor's field of view. The pyro-electric human detecting sensor is placed on the front of the robot. The Sharp IR detectors are placed just above the lower platform to detect the IR beacon, which is also close to the ground. The IR beacon is controlled by Axiom's CME11-E9 EVBU board,

which is also controlled by the HC11. The beacon consists of three high-output infrared LED's connected in series modulated at 29kHz.

The code for EDGAR is written in C and is compiled using ICC11 for windows.

Mobile Platform

EDGAR's platform was designed in AutoCad and cut out on the T-Tech machine. It is built out of ¼ inch aircraft plywood, supported by four 7 ½inch carriage bolts. It is approximately 8 inches high with a diameter of 9 inches, including the wheels. It consists of four flat circular levels. Each level is 1 ½inches apart, leaving room for the switches, sensors, LED's, and sensor servo. The bottom ring has a diameter of 8 inches, the second ring a diameter of 7 inches, the third ring a diameter of 6 inches, and the fourth ring a diameter of 5 inches. Each ring has a hole in the center with a diameter of 3 inches to serve as a bottle holder. The controller board is mounted to the bottom of the lower ring. The batteries are placed directly above the bottom ring. The bottom of the cup holder is place on top of the third ring, leaving the top two rings to support the bottle. Figure 1 shows the AutoCad layout of EDGAR. The center holes may look like different sizes due to an optical illusion, but they are all the same size (3 inches in diameter).

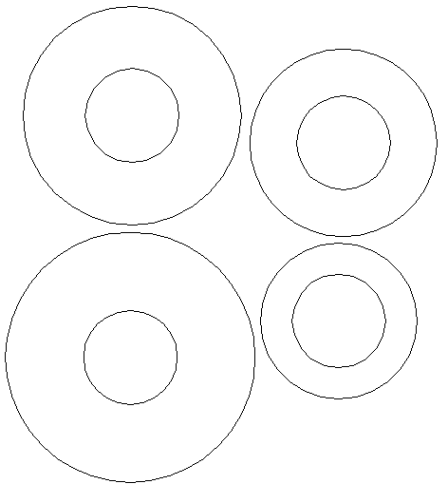

Figure 1: AutoCad layout of EDGAR platforms.

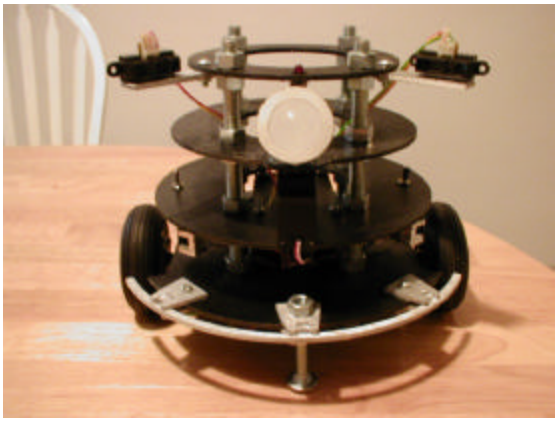

Figure 2: final platform design.

Actuation

The body of EDGAR is moved witch two 3 inch wheels mounted on hacked Futaba servos purchased from www.servocity.com. The sevos are hacked to allow a full 360 degrees of rotation. These have a rated torque of 43 oz/in and are mounted directly on the center axis of the platform to allow for exact turning in place action. Two 1 ½ inch carriage bolts are mounted on the front and back of the lower platform to act as casters and keep the robot balanced.

The pyro sensor is mounted on the same Futaba servo (unhacked) on the front of the robot. This servo will sweep the sensor a total of 90 degrees in five increments. EDGAR will know how far to turn towards a human based on the position of this servo when a human is detected.

The servos are controlled by motor and servo control methods in the header files written by Dr. Doty, the owner of Mekatronix. See Appendix B for these header files. The servos are controlled sending them a PWM signal. Different PWM's will cause the servo to go to certain positions.

Sensors

Sharp GP2D12 IR Proximity Sensors

Purchased at: Mekatronix
316 NW 17th St., Suite A
Gainesville, FL 32603
Phone: 352-376-7373
www.mekatronix.com

These sensors are used for more than one application. They are used for object avoidance and to know when to stop the robot upon a human encounter. They are placed in a cross-eyed configuration. This configuration is better than a straight-forward configuration because it will detect objects to the side of the robot. This prevents the robot from veering into walls.

These sensors are modulated at 40kHz. They can detect object up to around 2 ½ feet. At this distance, the analog port reading of the output voltage of the sensor reads around 20 (hex). The closest that these sensors can detect objects is around 4 inches. This gives an analog port reading of around 140. The sensors fail at any closer range. The threshold for my object avoiding code is set to 65. This is about a foot away from an object.

The GP2D12 sensors tend to act up under intense florescent lights. They seem to detect objects when there is nothing present under these conditions. The solution: ignore flickers in the output voltage.

These sensors draw too much current to get its voltage directly from the analog ports. I rerouted the power lines of the sensors to obtain its power from the servo power line. This is a regulated 5 volts and will provide as much current as the sensors need to function properly.

Sharp IR Cans

Purchased at: Leftover from previous class. They are no longer manufactured.

Two Sharp IR detecting sensors are used to detect my IR beacon. They are hacked to provide an analog reading for the A/D system on my board (see figure 3).

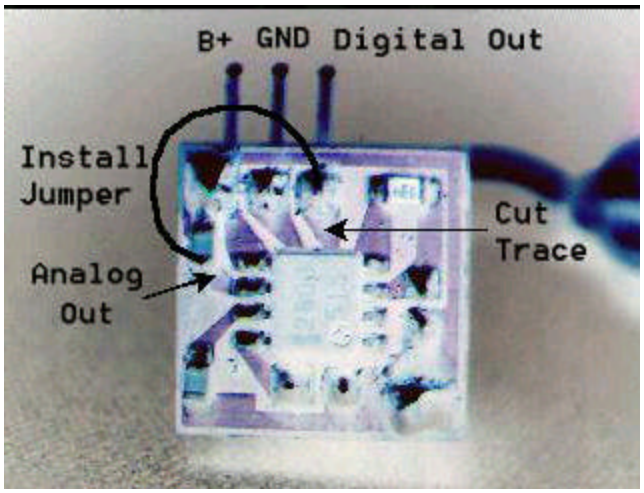

Figure 3: how to hack Sharp cans.

They can detect an IR signal modulated at ~29kHz to ~60kHz. I modulated my IR beacon at 29kHz. This provided me a large range of detection without interfering with the IR proximity sensors. The Sharp cans have an output voltage that floats around 88-90 hex. When the IR beacon is placed within a 6 inches of the sensors, the output voltage of the sensors reads around 112. I set my threshold value to 100 to stop the robot within a foot of the beacon. The sensors are placed at a slight outward angle to better realize the location of the beacon. If they were directed directly forward, a beacon slightly off of straight in front of the robot would read as if it were directly in front of the robot.

The IR beacon consists of three high-output infrared LED's purchased from Radio Shack. I use three to give a wider range of emission. One LED has a 45 degree

angle of emission to 50% intensity, shown in figure 3. Using three LED's in the pattern shown in figure 4 widens the range of emission.

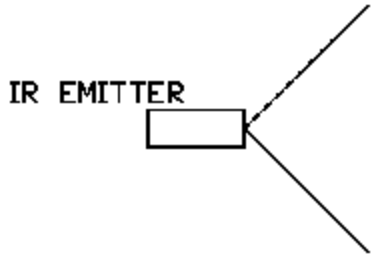

Figure 4: 45 degree angle of IR emission to 50% intensity.

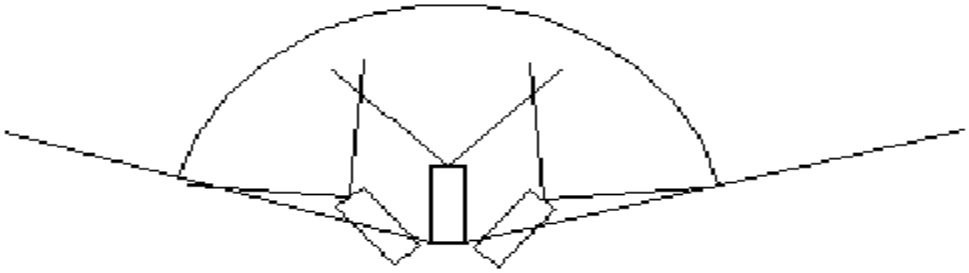

Figure 5: wide angle of IR emission using 3 LED's.

Pyro Sensor

Purchased at: Acroname, Inc.
4894 Sterling Dr.
Boulder, CO 80302
Phone: 720-564-0373
www.acroname.com
part # R3-PYRO1

Humans have a skin temperature of about 93 degrees F. The emissivity of human skin is around .98 times the emissivity of a perfect black body. The infrared radiation from a person is independent of race but can be affected by the inhibited circulation often caused by smoking. The wavelength of maximum energy radiated by humans is about 10 micrometers. The total energy emitted by a typical human is about 800 watts. Clothing

tends to mask some of the emitted energy so the job of detecting humans boils down to detecting subtle changes of energy in the 8 to 14 micrometer range while rejecting changes in other wavelengths that may be caused by light, motors, etc.

The active element in the pyro sensor is made with a substrate of lithium tantalate, which is very sensitive in the 8-14 micrometer range. A slice of the substrate is doped with an electrode on both sides. When infrared energy hits the substrate, heat is generated which displaced electrons, effectively generating a charge between the electrodes. This small charge is then amplified with an op-amp, and the result is a very usable signal that reflects changes in infrared energy. My pyro-sensor design uses two detectors placed side-by-side. The voltage difference between the two detectors can be amplified and measured to a get a much more sensitive reading.

The output of the sensor typically floats around a steady-state value of 2.5V. When a source of IR energy in the 8-14 um wavelength moves relative to the field-of-view (FOV) of the detector, the differential created in the dual detector elements creates a change in this output voltage. The voltage will fall when the motion is in one direction and rise with motion in the opposite direction. The detector stabilizes fairly quickly so if the motion stops, the detector will quickly gravitate back to the steady-state value of 2.5V. Since the sensor detects moving IR, a person standing still in the FOV will not cause a change in the output voltage.

Operating Characteristics:

Parameter	Unit	Rating
Responsivity	V/M	3.70E+05
Common Mode Rejection Min	-	5/1
Common Mode Rejection Type	-	15/1
Noise	mV/Hz 1/2	0.36
Thermal Breakpoint	Hz	0.15
Electrical Breakpoint	Hz	5
Incident Power (Max)	Watts	0.02
Power Supply Voltage	VDC	5-15
Power Supply Current	mA	2

Output Characteristics

Parameter	Unit	Rating
Voltage (Max)	V	+
Current (Rec.)	mA	0.02
Output load (Min)	Kohm	125

Ambient Operating Characteristics:

Parameter	Unit	Rating
Storage Temperature	degrees C	-55 to 155
Operating Temperature	degrees C	-40 to 70
Sensitivity to Temperature	%/degrees C	+0.3

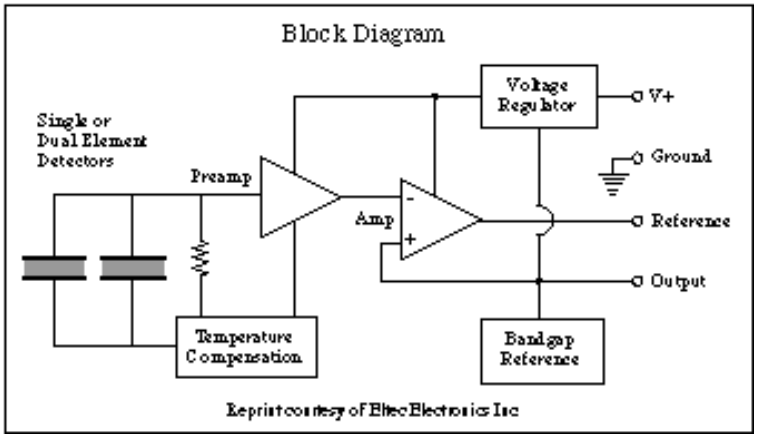

Figure 6: block diagram of dual element design.

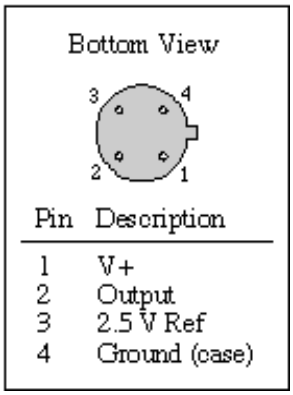

Figure 7: pinout.

Since the pyro sensor detects *moving* infrared, the sensor is mounted on a servo and swept across the room. This way, a human does not have to be moving to be detected. The servo is swept five increments of 18 degrees for a total of 90 degrees (see figure 7). EDGAR knows how far to turn towards a human based on the position of the controlling servo at the time of detection.

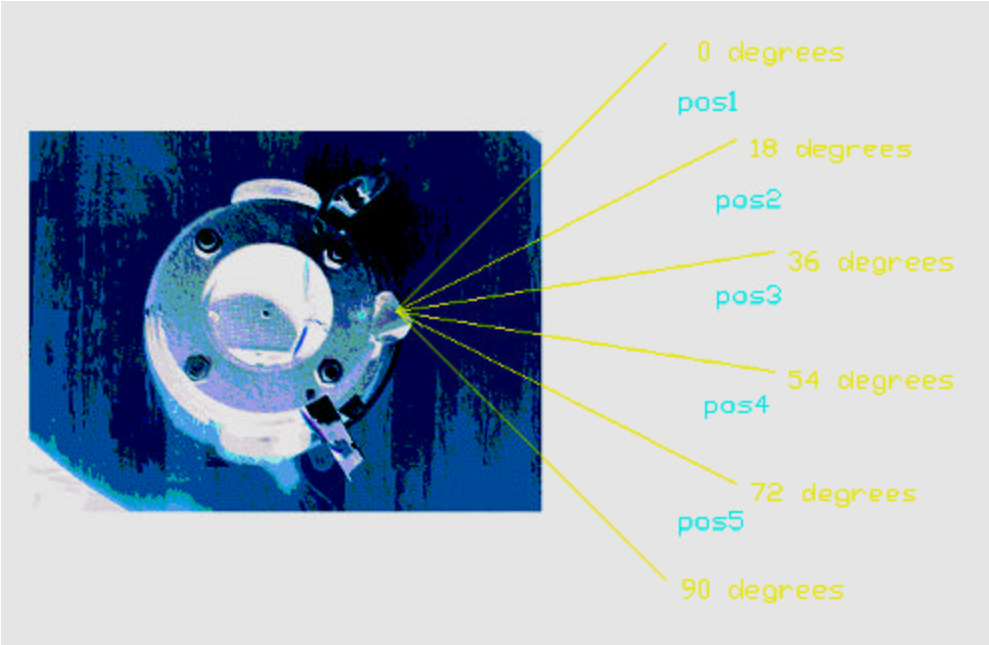

Figure 8: pyro sensor sweeping positions.

One problem encountered with this sweeping method came about when the robot was subjected to a crowd of people. Since the sensor starts sweeping from the left to the right, it would keep turning left until it got to he leftmost person in a crowd. To fix this, I did the following: Once a human is detected, EDGAR will keep checking the middle position to see if the human is still in front of it. Once the human is no longer in front, it will then proceed with its normal sweeping pattern.

Sometimes after the sensor detects a human, the output voltage gets stuck at a

high value around 250 hex. To get rid of this "stuck" voltage, the sensor is swept back and forth over a small increment each time a human is detected. This settles the output voltage down to its normal level of 128.

CdS Cell

Purchased at: obtained in lab.

One CdS cell is used to detect when a bottle is placed inside the robot. Two ultra-bright red LED's shine directly on the cell to provide a consistent reading in most lighting conditions. The resistance of the cell changes when a bottle covers the cell, blocking the light from the LED's. This change in resistance changes the voltage across the cell, which is read by the analog port.

The cell's resistance ranges from ~70kOhm to ~30kOhm. A 50kOhm resistor is placed in series with the cell to limit the current through the cell and to ensure that the voltage across the cell is between 0 and 5 volts.

Behaviors

Upon reset, EDGAR immediately scans for humans. If no human is detected, it roams around avoiding obstacles and looks for humans until one is found. When the robot is roaming around, all of the feedback LED's are on. Once a human is detected, two LED's turn on in the direction of the human, showing where the human was detected. EDGAR then turns towards the human and drives forward. The pyro sensor continues scanning for humans, recalibrating itself along the way, until a human is encountered. EDGAR then backs up and stops, waiting for bottle placement. The rear LED's light up when the robot goes in reverse (and also when there is no human detected). If a bottle is placed inside, the robot goes into beacon searching mode. In this mode, the feedback LED's display a distinct pattern. If no bottle is placed inside, the

robot turns and searches for other humans. In beacon searching mode, once EDGAR comes within a foot of the beacon, it stops and waits for the bottle to be removed. While waiting for bottle removal, the LED's display a different distinct pattern. Once the bottle is removed, EDGAR turns and searches for more humans.

Experimental Layout and Results

EDGAR was tested in a crowd of people under florescent lights. It located people and waited for bottle placement. When no bottle was placed inside, it turned a looked for more people. I stood close by EDGAR so it would detect me first. I walked away from it. It proceeded to follow me until I stopped moving. Once it encountered me, it backed up and waited for bottle placement. I put an empty Coke can inside and it returned it to the beacon and stayed there until I removed the can. It then turned and found more people.

The proximity sensors acted up a little bit under the florescent lights. It acted like there was an object in front of it for few brief moments. This did not interfere with EDGAR completing its tasks; it just slowed down the process by a couple of seconds.

Conclusion

EDGAR completes all of its tasks with precision and accuracy. It is an attractive robot with a practical application. I am very happy with the final product of this project. The major thing that I am not happy with is the battery life. EDGAR will run for about 5 minutes on a full charge before the batteries get low enough for the robot to malfunction.

Future Work

The ultimate goal of this project is to create a robot that will serve drinks and retrieve empty drinks. I would like to add more actuation to load and unload the robot with drinks. I would need a larger battery source for this new design because the robot would be much larger. I would also need stronger motors to push the heavier weight.

Documentation

Thanks to:

Instruction from Dr. Arroyo, Dr. Schwartz, Aamir Quaiyumi, Uriel Rodriguez, Tae Choi.

Keith L. Doty. TJPRO assembly manual; Mekatronix 1999.

Special thanks to Keith L. Doty for providing header files for the MTJPRO board.

Appendices

APPENDIX A
EDGAR CODE

```
//****************************************
//Matt Cousins
//edgar.c
//****************************************

#include <analog.h>
#include <motortjp.h>
#include <clocktjp.h>
#include <isrdecl.h>
#include <stdio.h>
#include <hc11.h>


//-----PROTOTYPES-----\\

void turnleft(int time);
void turnright(int time);
void turnl(void);
void turnr(void);
void forward(void);
void reverse(void);
int objectdetect(void);
int scanturn(void);
void stopmotors(void);
void randomturn(void);
void findbeacon(void);
void clearpyro(void);
int midscan(void);



//-----DEFINES-----\\

#define BOTTLE_IN (analog(1) > 170)
#define RIGHT_IR   (analog(2))
#define LEFT_IR    (analog(3))
#define SHARPL (analog(5))
#define SHARPR (analog(6))
#define UPDATE_LED *(unsigned char *)(0x7000) = LEDMASK
#define IRTHRESH 65
#define BUMPER ((analog(0) > 10) && (analog(0) < 120))
#define BEACTHRESH 4
#define SWEEPTIME 100
#define scan_threshold_h 141
#define scan_threshold_l 114
#define forwardtime 1000
```

```
//*************************************************\\
//MAIN
//*************************************************\\

void main(void)
{
      int time;
      int position = 0;
      int turncnt = 0;
      unsigned int ledcnt1 = 0;
      unsigned int ledcnt2 = 1;
      int LEDMASK;

//-----INITIALIZERS-----\\

      init_analog();
      init_serial();
      init_servotjp();
      init_clocktjp();
      init_motortjp();

      LEDMASK = 0x80;


while(1){


      if(BOTTLE_IN){

            LEDMASK = 0xFF ;  //turn all lights on
            UPDATE_LED;             //when bottle in

                  servo(0, 2000);               //put servo back to
pos1
                  findbeacon();                 //find ir beacon
                  if(objectdetect() == 1){      //avoid obstacles
                        reverse();
                        wait(500);
                        randomturn();
                  }

      }


      else if(scanturn() == 1){

            forward();
            clearpyro();
            if(objectdetect() == 1){      //if object is detected
                  reverse();                    //it is a human, so
                  wait(700);                    //stop and wait for
                  stopmotors();                 //bottle placement
                  wait(4000);
                  turnleft(1500);         //no bottle, turn and search
            }
```

```
      }

      else {
            LEDMASK = 0x03;         //rear lights on
            UPDATE_LED;
            reverse();
            wait(300);
            randomturn();
            forward();
            if(objectdetect() == 1){
                  reverse();
                  wait(500);
                  randomturn();
            }
      }



}

}

//************************************************\\
//FUNCTION: turnleft                                          \\
//DESCRIPTION: -turns robot left                        \\
//RETURNS: nothing                                            \\
//INPUTS: turn time in milliseconds               \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: motorp()                        \\
//************************************************\\

void turnleft(int time){

      motorp(0, -100);
      motorp(1, -100);
      wait(time);
      stopmotors();
}

//************************************************\\
//FUNCTION: turnright                                         \\
//DESCRIPTION: -turns robot right                       \\
//RETURNS: nothing                                            \\
//INPUTS: turn time in milliseconds               \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: motorp()                        \\
//************************************************\\

void turnright(int time)
{

      motorp(0, 100);
      motorp(1, 100);
      wait(time);
      stopmotors();
}
```

```
//************************************************\\
//FUNCTION: stopmotors                                          \\
//DESCRIPTION: -stops moving motors                        \\
//RETURNS: nothing                                              \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: motorp()                            \\
//************************************************\\

void stopmotors(void){
      motorp(0, 0);
      motorp(1, 0);
}

//************************************************\\
//FUNCTION: scanturn                                            \\
//DESCRIPTION:  -sweeps pyro sensor                     \\
//                      -turns towards person if detected   \\
//RETURNS: 1 if detect, 0 if no detect                  \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: turnleft(), turnright(),            \\
//                            wait(), servo(),                  \\
//************************************************\\

int scanturn(void)
{
      int position = 0;


//----TURN TIMES FOR HUMAN DETECTION----\\

      int pos1 = 300;
      int pos2 = 150;
      int pos3 = 0;
      int pos4 = 150;
      int pos5 = 300;
      int LEDMASK;



            servo(0, 2000);
            wait(SWEEPTIME);

            if(objectdetect() == 1){      //if object is detected
                  reverse();                    //turn and avoid
                  wait(600);
                  randomturn();
            }
            servo(0, 2400);

            printf("Value: %d\n",analog(4));
            if((analog(4)) > scan_threshold_h){
                  LEDMASK = 0xC0;
                  UPDATE_LED;
                  turnright(pos1);
                  goto FOUND;
```

```
}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0xC0;
      UPDATE_LED;
      turnright(pos1);
      goto FOUND;

}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 2800);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x60;
      UPDATE_LED;
      turnright(pos2);
      goto FOUND;

}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x60;
      UPDATE_LED;
      turnright(pos2);
      goto FOUND;

}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 3200);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x30;
      UPDATE_LED;
      turnright(pos3);
      goto FOUND;

}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x30;
      UPDATE_LED;
      turnright(pos3);
      goto FOUND;
}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
```

```
      wait(600);
      randomturn();
}
servo(0, 3600);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x18;
      UPDATE_LED;
      turnleft(pos4);
      goto FOUND;
}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x18;
      UPDATE_LED;
      turnleft(pos4);
      goto FOUND;
}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                    //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 4000);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x0C;
      UPDATE_LED;
      turnleft(pos5);
      goto FOUND;
}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x0C;
      UPDATE_LED;
      turnleft(pos5);
      goto FOUND;

}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                    //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 3600);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x0C;
      UPDATE_LED;
      turnleft(pos5);
      goto FOUND;
}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x0C;
```

```
      UPDATE_LED;
      turnleft(pos5);
      goto FOUND;
}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 3200);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x18;
      UPDATE_LED;
      turnleft(pos4);
      goto FOUND;

}
else if((analog(4)) <  scan_threshold_l){
      LEDMASK = 0x18;
      UPDATE_LED;
      turnleft(pos4);
      goto FOUND;
}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 2800);

printf("Value: %d\n",analog(4));
if((analog(4)) > scan_threshold_h){
      LEDMASK = 0x30;
      UPDATE_LED;
      turnright(pos3);
      goto FOUND;

}
else if((analog(4)) < scan_threshold_l){
      LEDMASK = 0x30;
      UPDATE_LED;
      turnright(pos3);
      goto FOUND;

}
wait(SWEEPTIME);
if(objectdetect() == 1){      //if object is detected
      reverse();                  //turn and avoid
      wait(600);
      randomturn();
}
servo(0, 2400);
```

```
            printf("Value: %d\n",analog(4));
            if((analog(4)) > scan_threshold_h){
                  LEDMASK = 0x60;
                  UPDATE_LED;
                  turnright(pos2);
                  goto FOUND;

            }
            else if((analog(4)) <  scan_threshold_l){
                  LEDMASK = 0x60;
                  UPDATE_LED;
                  turnright(pos2);
                  goto FOUND;

            }
            wait(SWEEPTIME);
            if(objectdetect() == 1){      //if object is detected
                  reverse();                    //turn and avoid
                  wait(600);
                  randomturn();
            }
            servo(0, 2000);

            printf("Value: %d\n",analog(4));
            if((analog(4)) > scan_threshold_h){
                  LEDMASK = 0xC0;
                  UPDATE_LED;
                  turnright(pos1);
                  goto FOUND;

            }
            else if((analog(4)) <  scan_threshold_l){
                  LEDMASK = 0xC0;
                  UPDATE_LED;
                  turnright(pos1);
                  goto FOUND;

            }
            wait(SWEEPTIME);

            return 0;   //no human found

      FOUND:

            return 1;   //human found



}

//************************************************\\
//FUNCTION: randomturn                                          \\
//DESCRIPTION:  -turns a random direction for a       \\
//                      random amount of time                   \\
//RETURNS: nothing                                              \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
```

```
//FUNCTIONS CALLED: motorp(), wait()                        \\
//************************************************\\

void randomturn(void){

 int i;
 unsigned rand;

 rand = TCNT;

 if (rand & 0x0001) {
   motorp(0, -100);  //turn left
   motorp(1, -100);
 }
 else {
   motorp(0, 100);  //turn right
   motorp(1, 100);
 }

 i=(rand % 1024);
 if(i>250)
      wait(i);
 else
      wait(250);
 stopmotors();
}


//************************************************\\
//FUNCTION: turnl                                             \\
//DESCRIPTION: -turns robot left                              \\
//RETURNS: nothing                                            \\
//INPUTS: none                                                \\
//OUTPUTS: none                                          \\
//FUNCTIONS CALLED: motorp()                             \\
//************************************************\\

void turnl(void)
{

      motorp(0, -100);
      motorp(1, -100);

}

//************************************************\\
//FUNCTION: turnr                                             \\
//DESCRIPTION: -turns robot right                             \\
//RETURNS: nothing                                            \\
//INPUTS: none                                                \\
//OUTPUTS: none                                          \\
//FUNCTIONS CALLED: motorp()                             \\
//************************************************\\

void turnr(void)
{
```

```
      motorp(0, 100);
      motorp(1, 100);

}


//*************************************************\\
//FUNCTION: objectdetect                                        \\
//DESCRIPTION:  -uses ranging sensors to see if       \\
//                          there is and object nearby          \\
//RETURNS: 1 if object, 0 if no object                \\
//INPUTS: none                                        \\
//OUTPUTS: none                                       \\
//FUNCTIONS CALLED:
      \\
//*************************************************\\

int objectdetect(void)
{
      if((RIGHT_IR > IRTHRESH) || (LEFT_IR > IRTHRESH))
            return 1;
      if (BUMPER)
            return 1;
      else
            return 0;
}

//*************************************************\\
//FUNCTION: forward
      \\
//DESCRIPTION: -drive forward                                   \\
//RETURNS: nothing                                                    \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: motorp()                            \\
//*************************************************\\

void forward(void){
      motorp(0, 100);
      motorp(1, -100);

}

//*************************************************\\
//FUNCTION: reverse
      \\
//DESCRIPTION: -drive in reverse                                \\
//RETURNS: nothing                                                    \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: motorp()                            \\
//*************************************************\\

void reverse(void){
      motorp(0, -100);
      motorp(1, 100);
```

```
}



//************************************************\\
//FUNCTION: findbeacon                                                  \\
//DESCRIPTION: -finds IR beacon                                         \\
//                      -stops and waits for bottle removal \\
//                            upon arrival
      \\
//RETURNS: nothing                                                      \\
//INPUTS: none                                                    \\
//OUTPUTS: none                                             \\
//FUNCTIONS CALLED: randomturn(), motorp(), wait()    \\
//                            turnl(), turnr(), forward()         \\
//                            objectdetect(), stopmotors()  \\
//************************************************\\

void findbeacon(void){

      int LEDMASK, wtime = 300 ;

            if((SHARPR > 110) || (SHARPL > 110)){     //close to beacon
                  stopmotors();

      //----LED PATTERN GENERATOR----\\

                  LEDMASK = 0x01;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x03;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x07;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x0F;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x1F;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x3F;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x7F;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xFF;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xFE;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xFC;
                  UPDATE_LED;
                  wait(wtime);
```

```
                  LEDMASK = 0xF8;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xF0;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xE0;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0xC0;
                  UPDATE_LED;
                  wait(wtime);
                  LEDMASK = 0x80;
                  UPDATE_LED;
                  wait(wtime);

            }

            else if((SHARPR < 91) && (SHARPL < 91)) //no ir detection
                  randomturn();

            else if((SHARPL - SHARPR) > BEACTHRESH)
                  turnl();

            else if((SHARPR - SHARPL) > BEACTHRESH)
                  turnr();

            else if((SHARPR - SHARPL) < BEACTHRESH)
                  forward();

            else if((SHARPL - SHARPR) < BEACTHRESH)
                  forward();




}

//************************************************\\
//FUNCTION: clearpyro                                          \\
//DESCRIPTION: -sweeps pyro to clear stuck Vo         \\
//RETURNS: nothing                                             \\
//INPUTS: none                                                 \\
//OUTPUTS: none                                                \\
//FUNCTIONS CALLED: servo()                                        \\
//************************************************\\

void clearpyro(void) {
            servo(0, 2000);               //clear pyro Vo
            wait(75);
            servo(0, 2300);
            wait(75);
            servo(0, 2000);
            wait(75);
            servo(0, 2300);
            wait(75);
```

```
}

//************************************************\\
//FUNCTION: midscan
      \\
//DESCRIPTION: -sweeps pyro to detect person         \\
//                     directly in front                        \\
//RETURNS: 1 if true, 0 if not                              \\
//INPUTS: none                                          \\
//OUTPUTS: none                                         \\
//FUNCTIONS CALLED: servo()   wait()                    \\
//************************************************\\

int midscan(void){

      servo(0, 2400);
      wait (400);
      servo(0, 3600);
      wait(400);
      servo(0, 2400);
      if((analog(4)) > scan_threshold_h)
            return 1;
      else if((analog(4)) <  scan_threshold_l)
            return 1;
      else
            return 0;
}
```

APPENDIX B
HEADER FILES

```
//analog.h
//mekatronix

#ifndef ANALOG_H
#define ANALOG_H

void init_analog(void);

int analog(int port);
/* Takes one reading from the analog port and returns the value read
 * Inputs : Port
 * Output : None
 * Return Value : Value read from A/D port
 */

#endif

/*************************************************************
* MEKATRONIX Copyright 1998
* Header File for TJ Pro motor drivers
* Programmer: Drew Bagnell
  Modified  Sept. 04,1998 by Keith L. Doty
*
*
*************************************************************/

#ifndef _MOTORTJP_H
#define _MOTORTJP_H

/*************** Includes ************************************/
#include <hc11.h>
#include <mil.h>

/*************** Constants ***********************************/
#define PERIOD 40000


/*******Function Prototypes******************/
void TOC5_isr(void);   /*Interrupt handler for the motors*/
void init_motortjp(void); /*Must be executed before motors will
operate*/
void motorp(int, int); /*Motor command*/

#endif

/*********************************************************************
**MEKATRONIX Copyright 1998                                          *
 * Title        clocktjp.h
*
 * Programmer   Keith L. Doty
*
```

```
 * Date         Sept 04, 1998
*
 * Version      1
*
 *
*
 * Description
 *  This header file refers to all routines and interrupt drivers   *
 *      necessary for TJ Pro clock control. The clock keeps track of
*
 *      milliseconds, seconds, minutes, hours, days and timertj.

**********************************************************************
*/


/*************************** Includes
*********************************/
#include <hc11.h>
#include <mil.h>
/**********************************************************************
**/

/*************************** Prototypes
*********************************/

#pragma interrupt_handler TOC1_isr
void TOC1_isr(void); /*Interrupt service routine to generate clock
ticks*/
void init_clocktjp(void);/*Must be executed to enable clock routines*/
void wait(int);   /* Wait for int milliseconds */

/**********************************************************************
**/


/*************************** Globals
*********************************/

unsigned int msec, seconds, minutes, hours, days, timertjp;

/*
           Range
  msec:    0-999
  seconds: 0-59
  minutes: 0-59
  hours:   0-23
  days:    0-255

  timertjp:  0 to 65535 milliseconds

*/

/**********************************************************************
**/

/**********************************************************************
```

```
 * Title            isrdecl.h                                          *
 * Programmer       Keith L. Doty                                      *
 * Date             September 16, 1998                                 *
 * Version          1.1 with servo driver                              *
 *                                                                     *
 * Description
 *       This file declares nil functions for all the undeclared       *
 *   interrupt service routines. If you define a new handler, its  *
 *   ISR name must appear in a #define below in order to bypass the*
 *   conditional compile that will force it to be a null function. *
 *
 * Version History:                                                    *
 *                                                                     *

**********************************************************************/

/*
   The following list of #defines lists the currently used interrupt
   service routines.
*/


/* TJ Pro timer/clock handler */
#define TOC1_isr

/* TJ Pro Servo handler */
#define TOC4_isr

/* TJ Pro Motor handler */
#define TOC5_isr


/*
   ISR names not listed above as a #define will be compiled to a null
   function in the conditional compilation below.
*/

#ifndef SCI_isr
#pragma interrupt_handler SCI_isr;
void SCI_isr()
{/* Nil Function Declaration */}
#endif

#ifndef SPI_isr
#pragma interrupt_handler SPI_isr;
void SPI_isr()
{/* Nil Function Declaration */}
#endif

#ifndef PAIE_isr
#pragma interrupt_handler PAIE_isr;
void PAIE_isr()
{/* Nil Function Declaration */}
#endif

#ifndef PAV_isr
#pragma interrupt_handler PAV_isr;
```

```
void PAV_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TOF_isr
#pragma interrupt_handler TOF_isr;
void TOF_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TOC5_isr
#pragma interrupt_handler TOC5_isr;
void TOC5_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TOC4_isr
#pragma interrupt_handler TOC4_isr;
void TOC4_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TOC2_isr
#pragma interrupt_handler TOC2_isr;
void TOC2_isr()
{/* Nil Function Declaration */} /* Controls PWM of wheel motors */
#endif

#ifndef TOC3_isr
#pragma interrupt_handler TOC3_isr;
void TOC3_isr()
{/* Nil Function Declaration */} /* IR detector service routine */
#endif

#ifndef TOC1_isr
#pragma interrupt_handler TOC1_isr;
void TOC1_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TIC3_isr
#pragma interrupt_handler TIC3_isr;
void TIC3_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TIC2_isr
#pragma interrupt_handler TIC2_isr;
void TIC2_isr()
{/* Nil Function Declaration */}
#endif

#ifndef TIC1_isr
#pragma interrupt_handler TIC1_isr;
void TIC1_isr()
{/* Nil Function Declaration */}
#endif
```

```
#ifndef RTI_isr
#pragma interrupt_handler RTI_isr;
void RTI_isr()
{/* Nil Function Declaration */}
#endif

#ifndef IRQ_isr
#pragma interrupt_handler IRQ_isr;
void IRQ_isr()
{/* Nil Function Declaration */}
#endif

#ifndef XIRQ_isr
#pragma interrupt_handler XIRQ_isr;
void XIRQ_isr()
{/* Nil Function Declaration */}
#endif

#ifndef SWI_isr
#pragma interrupt_handler SWI_isr;
void SWI_isr()
{/* Nil Function Declaration */}
#endif

#ifndef ILLOP_isr
#pragma interrupt_handler ILLOP_isr;
void ILLOP_isr()
{/* Nil Function Declaration */}
#endif

#ifndef COP_isr
#pragma interrupt_handler COP_isr;
void COP_isr()
{/* Nil Function Declaration */}
#endif

#ifndef CLMON_isr
#pragma interrupt_handler CLMON_isr;
void CLMON_isr()
{/* Nil Function Declaration */}
#endif
```

```
#ifndef __HC11_H
#define __HC11_H                    1

/* base address of register block, change this if you relocate the
register
 * block. This is from an A8. May need to be changed for other HC11
members
 * or if you relocate the IO base address.
 */
#define _IO_BASE                 0x1000
#define PORTA *(unsigned char volatile *)(_IO_BASE + 0x00)
```

```
#define PIOC  *(unsigned char volatile *)(_IO_BASE + 0x02)
#define PORTC *(unsigned char volatile *)(_IO_BASE + 0x03)
#define PORTB *(unsigned char volatile *)(_IO_BASE + 0x04)
#define PORTCL*(unsigned char volatile *)(_IO_BASE + 0x05)
#define DDRC  *(unsigned char volatile *)(_IO_BASE + 0x07)
#define PORTD *(unsigned char volatile *)(_IO_BASE + 0x08)
#define DDRD  *(unsigned char volatile *)(_IO_BASE + 0x09)
#define PORTE *(unsigned char volatile *)(_IO_BASE + 0x0A)
#define CFORC *(unsigned char volatile *)(_IO_BASE + 0x0B)
#define OC1M  *(unsigned char volatile *)(_IO_BASE + 0x0C)
#define OC1D  *(unsigned char volatile *)(_IO_BASE + 0x0D)
#define TCNT  *(unsigned short volatile *)(_IO_BASE + 0x0E)
#define TIC1  *(unsigned short volatile *)(_IO_BASE + 0x10)
#define TIC2  *(unsigned short volatile *)(_IO_BASE + 0x12)
#define TIC3  *(unsigned short volatile *)(_IO_BASE + 0x14)
#define TOC1  *(unsigned short volatile *)(_IO_BASE + 0x16)
#define TOC2  *(unsigned short volatile *)(_IO_BASE + 0x18)
#define TOC3  *(unsigned short volatile *)(_IO_BASE + 0x1A)
#define TOC4  *(unsigned short volatile *)(_IO_BASE + 0x1C)
#define TOC5  *(unsigned short volatile *)(_IO_BASE + 0x1E)
#define TCTL1 *(unsigned char volatile *)(_IO_BASE + 0x20)
#define TCTL2 *(unsigned char volatile *)(_IO_BASE + 0x21)
#define TMSK1 *(unsigned char volatile *)(_IO_BASE + 0x22)
#define TFLG1 *(unsigned char volatile *)(_IO_BASE + 0x23)
#define TMSK2 *(unsigned char volatile *)(_IO_BASE + 0x24)
#define TFLG2 *(unsigned char volatile *)(_IO_BASE + 0x25)
#define PACTL *(unsigned char volatile *)(_IO_BASE + 0x26)
#define PACNT *(unsigned char volatile *)(_IO_BASE + 0x27)
#define SPCR  *(unsigned char volatile *)(_IO_BASE + 0x28)
#define SPSR  *(unsigned char volatile *)(_IO_BASE + 0x29)
#define SPDR  *(unsigned char volatile *)(_IO_BASE + 0x2A)
#define BAUD  *(unsigned char volatile *)(_IO_BASE + 0x2B)
#define SCCR1 *(unsigned char volatile *)(_IO_BASE + 0x2C)
#define SCCR2 *(unsigned char volatile *)(_IO_BASE + 0x2D)
#define SCSR  *(unsigned char volatile *)(_IO_BASE + 0x2E)
#define SCDR  *(unsigned char volatile *)(_IO_BASE + 0x2F)
#define ADCTL *(unsigned char volatile *)(_IO_BASE + 0x30)
#define ADR1  *(unsigned char volatile *)(_IO_BASE + 0x31)
#define ADR2  *(unsigned char volatile *)(_IO_BASE + 0x32)
#define ADR3  *(unsigned char volatile *)(_IO_BASE + 0x33)
#define ADR4  *(unsigned char volatile *)(_IO_BASE + 0x34)
#define OPTION*(unsigned char volatile *)(_IO_BASE + 0x39)
#define COPRST*(unsigned char volatile *)(_IO_BASE + 0x3A)
#define PPROG *(unsigned char volatile *)(_IO_BASE + 0x3B)
#define HPRIO *(unsigned char volatile *)(_IO_BASE + 0x3C)
#define INIT  *(unsigned char volatile *)(_IO_BASE + 0x3D)
#define TEST1 *(unsigned char volatile *)(_IO_BASE + 0x3E)
#define CONFIG*(unsigned char volatile *)(_IO_BASE + 0x3F)

unsigned int read_sci(void);
unsigned char read_spi(void);
void write_eeprom(unsigned char *addr, unsigned char c);
void write_sci(unsigned int);
void write_spi(unsigned char);

/* These values are for a 8Mhz clock
 * 0x3? set the SCP1|SCP0 to 0x3
```

```
 */
typedef enum {
           BAUD9600 = 0x30, BAUD4800 = 0x31, BAUD2400 = 0x32,
            BAUD1200 = 0x33, BAUD600 = 0x34, BAUD300 = 0x35
                              } BaudRate;
void setbaud(BaudRate);

#ifndef INTR_ON
#define INTR_ON()                asm("                              cli")
#define INTR_OFF()               asm("                              sei")
#endif

#ifndef bit
#define bit(x)                  (1 << (x))
#endif

#ifdef _SCI
/* SCI bits */
#define RDRF                      bit(5)
#define TDRE                      bit(7)
#define T8                                                        bit(6)
#define R8                                                        bit(7)
#endif

#ifdef _SPI
/* SPI bits */
#define MSTR                      bit(4)
#define SPE                                                       bit(6)
#define SPIF                      bit(7)
#endif

#ifdef _EEPROM
/* EEPROM */
#define EEPGM                     bit(0)
#define EELAT                     bit(1)
#endif

#endif

/*********************************************************************
**
 * MEKATRONIX Copyright 1998                                         *
 * Title        mil.h
*
 * Programmer   Keith L. Doty
*
 * Date         March 10, 1998
*
 * Version      1
*
 *
*
 * Description
 *   A collection of functions that operate on MC68HC11 registers.
*
 *   Object file is in libtj.a
*
```

```
*****************************************************************************
*/

#ifndef _MIL_H
#define _MIL_H

#define CLEAR_BIT(x,y) x &= ~y;
/* Clear all bits in x corresponding to the bits set in y.
 * The other bits in x are not affected.
*/

#define SET_BIT(x,y) x |= y;
/* Set all bits in x corresponding to the bits set in y.
 * The other bits in x are not affected.
*/

#define CLEAR_FLAG(x,y) x &= y;
/* USE this routine ONLY for clearing flags. Clears flags in x
corresponding
 * to the bits set in y. Other flags in x are not affected.
 */


#endif
```

APPENDIX C
IR BEACON CODE

```
*************************************
*MATT COUSINS
*PROGRAM: IRLED
*DESCRIPTION: CREATES 29KHZ PULSE AT
*PORTB BIT0
*************************************



ORG $B600 ;START IN EEPROM

     LDAA $0 ;CLEAR A

LOOP
      INCA ;2 CLOCKS
  STAA  $1004  ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
      BRN      LOOP   ;3 CLOCKS
```

```
BRN       LOOP   ;3 CLOCKS
BRN       LOOP   ;3 CLOCKS
BRN       LOOP   ;3 CLOCKS
NOP              ;2 CLOCKS
NOP              ;2 CLOCKS
BRA       LOOP   ;3 CLOCKS
```