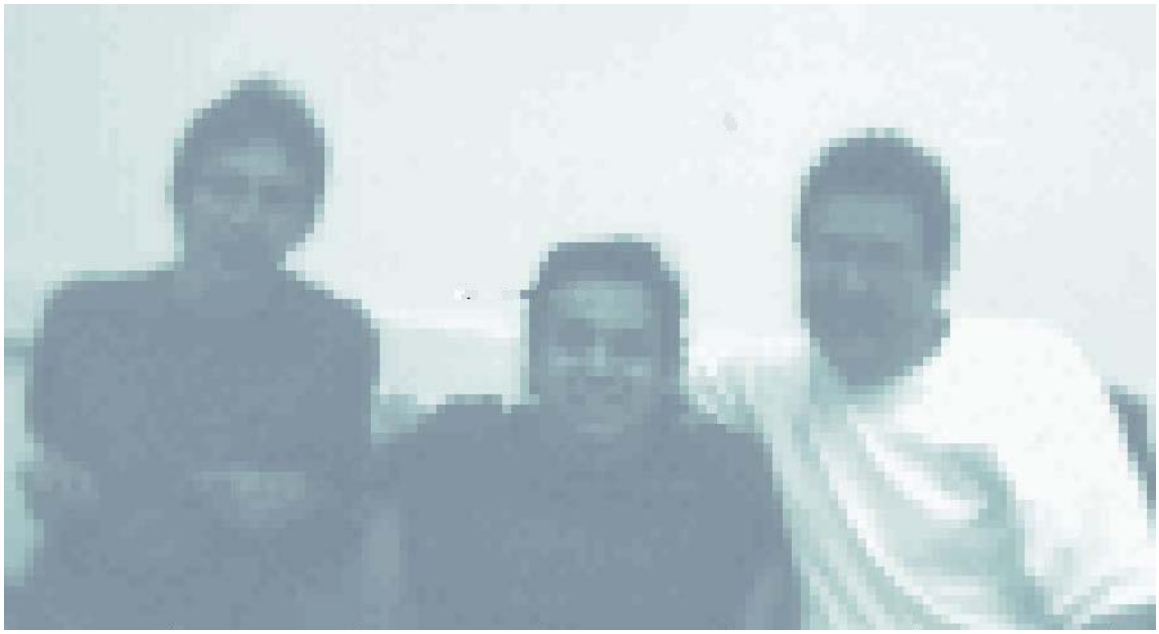
Ball and Hole Detection System
Robot B



May 6, 2002

Presented To:

New Mexico Institute of Mining and Technology
Electrical Engineering Department
Dr. William Rison
Dr. Kevin Wedeward

Presented By:

Jonathan Jackson
Aaron Prager
Keith Sanchez

Abstract

As members of the Junior design class of 2002, we were tasked with creating a robot that would be able to locate a golf ball and deliver the golf ball to a hole. As members of the Ball and Hole Detection group for Team B, we had the specific responsibility of developing a system to locate a golf ball and a hole on the green. We have designed a camera system that, under the correct conditions, can locate a ball or hole and determine the distance and angle to the object.

This report will consist of an overview of the major sections of our subsystem and details on their implementation

Keywords: robot, ball and hole, golf, image processing

Table of Contents

List of Figures

List of Tables

Introduction

2002 marked a new approach to Junior Design for the undergraduates of the New Mexico Tech Electrical Engineering program. For the first time, the project was to create a robot that is able to locate a golf ball on a green, pick it up, and deliver it to a hole. The task in front of the teams has been, in many ways, more difficult than projects of the past. Aside from designing the basic functionality of the individual subsystems, the various groups had to integrate their subsystems to create a single functional robot. This paper will cover the development of one piece of the robot, the Ball and Hole Detection subsystem.

The Ball and Hole subsystem must be able to find a standard white golf ball and white hole on a golf course green if the ball or hole is within 5 meters of the robot. It must be able to determine a distance and angle to the ball or hole, and it must be able to relay this information to the navigation subsystem of the robot. The Ball and Hole Detection system for robot B has been designed to meet these requirements and will function successfully under specific conditions.

System Design

We will examine the design of the subsystem in four main sections: Imaging, Communication, Hardware, and Calibration.

Imaging

Imaging is the most complex portion of the ball and hole detection system. We will go through the imaging process one step at a time.

The first step in getting the camera to work is setting up a system clock and resetting the camera. That is the function of the first two interrupts.

```
@interrupt void tci0(void)    //uses bit 0 to toggle output to make system clk 20KHz
{
TC0+=tempclk;  //720
if (chair==6)              //returns load line low
{
PORTEA = (PORTEA & ~0x02);
}
if (chair==16)             //checks to see if load line needs to go high
{
PORTEA= (PORTEA | 0x02);
chair=5;
cow++;
}
TFLG1=0x01;
}
```

Interrupt tci0 is the system clock throughout the whole picture-taking process. The camera can support a 500KHz system clock, but this interrupt, using a output compare to toggle PORTEA bit 1, creates a system clock that varies from ~11KHz to ~53KHz. The 53KHz clock is used for exposure time, and the 11KHz clock is used for everything else (loading registers, resetting, and downloading the image). This is done for the following reason. If the 11KHz clock was used as the exposure clock, the minimum exposure stepping interval would be ~728µS vs. ~152µS for the 53KHz and 16µS for the 500KHz

suggested system clock. The smaller stepping interval gives us more control and resolution, but the main advantage comes from the limitations of the 68HC12 itself. This program can run only as fast as its slowest interrupt, which in this case is about 90µS, and that becomes our resolution. However, the interrupt that controls exposure time can be executed much more rapidly, about 19µS, so this allows us to control the exposure time much more precisely.

```
@interrupt void xreset(void)          //uses bit 2 to reset system
{
if (a==0)
{
PORTEA=(PORTEA & ~0x04);            //lowers bit 2 of EA for reset
a++;                                //what kind of code this is
}
else
{
PORTEA=(PORTEA | 0x04);             //raises the reset line
TMSK1=((TMSK1 & ~0x04) | 0x02);     //turns off interrupt 2 and on interrupt 1
}
TFLG1=0x04;
}
```

The xreset interrupt is used to lower and raise bit 2 of PORTEA to reset the system. After this interrupt is complete, it turns itself off and turns on the interrupt to start shifting out the register data.

We will now move on to the **setregisters()** interrupt. This is the code to shift out the bits that need to be loaded into the registers of the cameras. 'a' keeps count of the number of bits shifted out, and TIOS1 raises and lowers the load line after 11 bits have been shifted. Once all 8 registers have been loaded, this interrupt turns itself off and turns on the start interrupt.

```
@interrupt void setregisters(void)    //uses bit 1 to put out data on falling edge
{
 if (cow<8)                           //number of bytes shifted out
{
            a = (((registers[cow] << chair) & 0x8000)==0x8000);
            chair++;
```

```
            if (a==1)
            {PORTEA |=0x01;}                //writes a 1 to PORTEA
            else
            {PORTEA &= ~0x01;}              //writes a 0 to PORTEA
}
else
{
            TMSK1=((TMSK1 & ~0x02) | 0x08);    //turns off interrupt 1 and on interrupt 3
            chair=6;
            a=0;                        //imPORTEAnt for later interrupt
}
TFLG1=0X02;
}
```

```
const unsigned int registers[] = {0x00a0,0x0105,0x0200, 0x0370,0x0401,0x0500,0x0601,0x0726};
```

This line of code is what will be loaded into the registers. The first two bytes is the register address, followed by the next two bytes which are the register data.

Address 0x00 receives a 0xA0. This value was suggested by the data sheet (11.6.10. Typical register settings) and worked so it was used.

Address 0x01 receives a 0x05. Bits 7,6, and 5 were suggested values from the data sheet (11.6.10. Typical register settings). However, bits 4-0 are the gain and are changed with different lighting conditions. Gain will be discussed further in the "Calibration" section.

Addresses 0x02 and 0x03 receive varying inputs. This register controls the exposure time and is changed with different lighting conditions. Gain will be discussed further in the "Calibration" section.

Address 0x04 receives a 0x01. This value was suggested by the data sheet (11.6.10. Typical register settings) so it was used.

Address 0x05 receives a 0x00. This value was suggested by the data sheet (11.6.10. Typical register settings) so it was used.

Address 0x06 receives a 0x01. This value was suggested by the data sheet (11.6.10. Typical register settings) so it was used.

Address 0x07 receives a 0x26. This value was suggested by the data sheet (11.6.10. Typical register settings) so it was used.

Please see the data sheet for specifics on setup and hold times for register clocking, loading, resetting, and the system clock.

The start interrupt is the next interrupt to get set. It raises PORTEA bit 2 long enough to get clocked (see section 10. AC Timing Requirements for exact specs). The start line is triggered after the registers are loaded, and causes the camera to take an image based on the register values. When this interrupt is finished, it turns itself off and turns on an interrupt that triggers when the ready line is asserted from the camera after the set amount of exposure time. This interrupt also speeds up the system clock.

Once the ready line is detected going high, on PTS4, the system clock slows down again and starts clocking out the data through the A/D converter (PAD7-5, depending on which camera it is downloading from). The pixels are saved as 8 bit hex values starting at 0x2000. Once all 16384 pixels are downloaded, the ready line drops low and the 'readydone' interrupt turns off all interrupts and will activate the second part of the image software, image processing.

Image processing

Image processing is started once the picture has been taken, and all the pixels have been downloaded. This code starts by looking at the first memory location data was stored, 0x2000, and compares the value to 'WHITE', which is a number we set as defining what value would be considered white. If the value is considered white, it adds the x quadrant value, y quadrant value, and increments a white pixel counter. Once all the pixels have been analyzed, the program determines if enough white pixels have been detected to make a ball or hole. If there were enough pixels, it divides the running sum of x and y values by the number of white pixels detected. This gives the centroid of white pixels. The image is then broken into 64 quadrants, or an 8x8 grid. The centroid is then placed in the appropriate quadrant. If the centroid is in front of the robot (x quadrant 3 or

4), a distance is determined from the y quadrant value. Once this is found, it is transmitted to navigation. If the x quadrant value is not 3 or 4, then the robot is turned left or right a pre-determined amount depending on which quadrant it is in. A picture is taken again, reanalyzed, and repositioned.

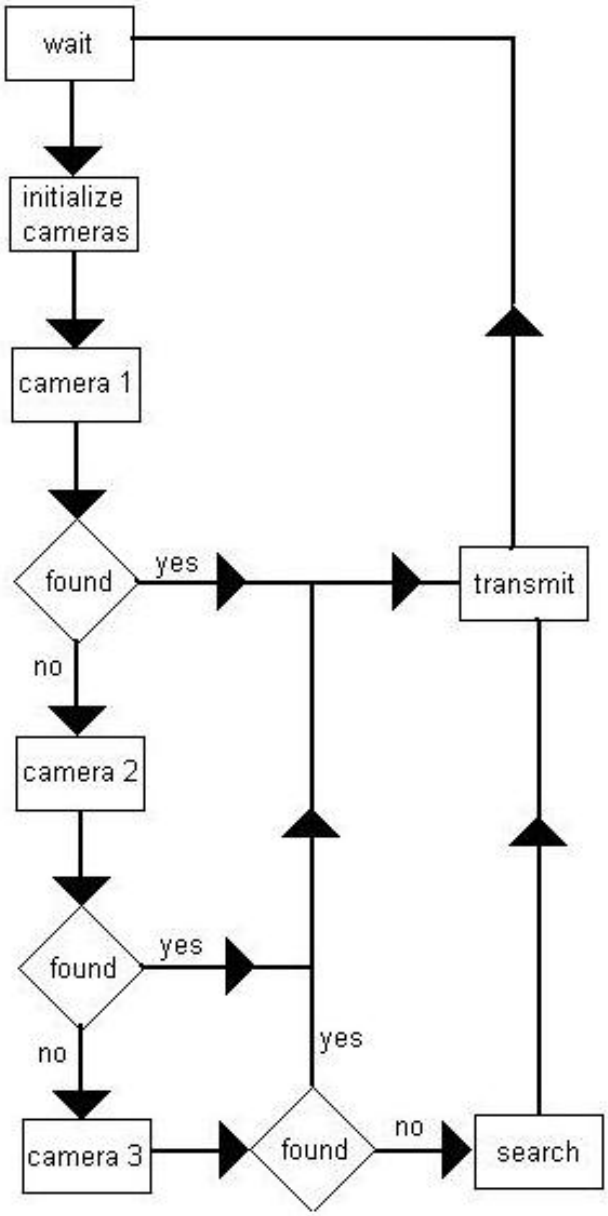


Figure 1 – Imaging flowchart

If a ball/hole is not found in the present camera, the code moves to the next camera and reexecutes the code. If no ball/hole is found in any of the cameras, a search pattern is executed. The search pattern provides movement instructions to the robot that will take it in a circle through the 5m radius circle that the ball/hole should be located in. At each stop on the search pattern the ball/hole system will run its imaging algorithm to attempt to locate the ball or hole. The search pattern will run indefinitely until the ball or hole is found.

Communications

One of the main subsystems of the ball and hole detection system is the communications system. The ball and hole detection system must be able to communicate with the navigation subsystem. It must be able to send angle and distance measurements and must be able to receive signals from the navigation group.

The communication algorithm is integrated into the entire code and can be summarized as follows:

1. The ball/hole subsystem is in a wait state until it receives a signal from navigation.
2. After receiving the signal, the ball/hole system goes through the image analysis, either finding the ball or selecting a search pattern coordinate.
3. The coordinate is transferred to the navigation group.
4. The ball/hole system returns to the wait state until signaled again. The process is then repeated.

The majority of the communication is done using the Serial Peripheral Interface (SPI) on the 68HC12. The SPI subsystem allows serial communication between a master and a slave to be implemented with relative simplicity. In this case, the ball/hole subsystem serves as the slave to Navigation's master. This means that the Navigation 68HC12 provides the clock and slave select lines to our 68HC12. Therefore, Navigation initiates all data transfer between the two subsystems.

The ball/hole subsystem begins in a wait state. During this time it simply waits for a signal from navigation to begin searching for the ball. It also activates the wait state LED on the status panel. The system remains in this state until it detects an activation signal from navigation in the form of the lowering of the slave select line. Lowering our slave select indicates that the robot has reached a stop, and the ball or hole needs to be found. When the low line is detected by our system, the ball/hole system goes to work activating the imaging functions. As previously discussed, the imaging function sets two variables, the distance and angle to be transmitted.

After the images are taken, analyzed, and movement data has been prepared, the imaging function exits and returns to the main function. The transmit function is then called. Transmission takes place in three phases. In the first phase, the distance to be moved, in centimeters, is sent. Next, the first half of the angle is sent, and finally, the second half of the angle is sent.

The data transfer is done in a straightforward manner. First, the data is loaded into the SP0DR register for data transfer. The ball/hole system then raises its ready line, signaling navigation that a packet is ready for transfer. We then wait for the slave select line to drop. Once the slave select falls, the navigation group activates their SPI, and the

data is transferred from ball/hole to navigation. When the transfer is complete, the completion flag will be raised in the ball/hole SPI status register. When this flag is raised, the ready line is dropped. This cycle repeats three times, once for each block of data sent to the navigation subsystem.

Once the third piece of data is sent, the ball/hole subsystem returns to the initial wait state to await another signal from navigation to begin the cycle over again.

Hardware

This section gives a detailed listing of the hardware required for this project and a thorough description of how this hardware can be used to duplicate our progress. Hardware played a vital, but less important, role in our piece of the golfing robot project. The majority of our project relied heavily upon software to complete communication with our cameras and in turn to analyze the images we retrieved. Our hardware consisted three main components. These components are the 68HC12, the Game Boy cameras, and their corresponding casings.

Materials required:

-68HC12 Motorola microcontroller with evaluation board

-Memory expansion kit for the 68HC12.

-Three Game Boy Cameras

-Casing material, including but not limited to:

-One old power supply case

-Three golf balls

-Poster board

-Spare sheet metal

-Hot glue gun and hot glue

-Mounting brackets

-Miscellaneous screws

-Ribbon cable (at least 12 feet)

-Serial connectors

-Ethernet RJ-45 cable (3 feet)

-miscellaneous electronics including but not limited to:

-miscellaneous wire

-Reset and power switches

-Black spray paint

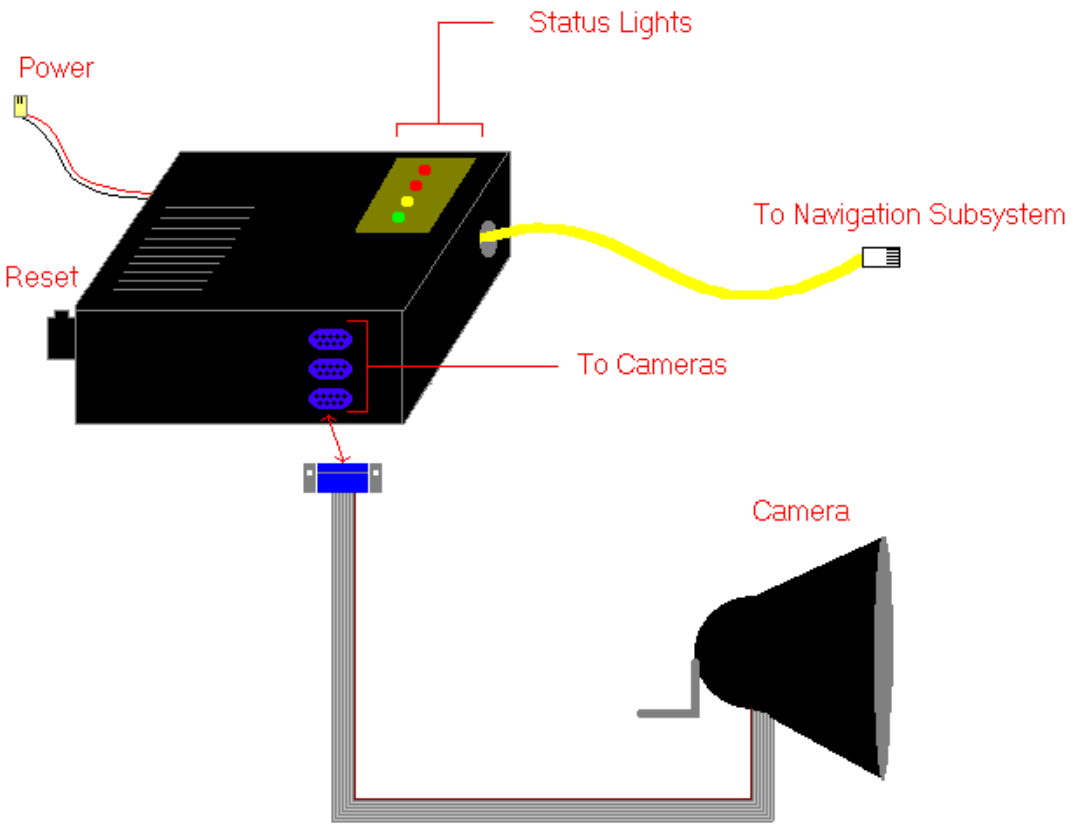


Figure 2 – Hardware Overview

The Motorola 68HC12 Microcontroller with memory expansion pack

The Motorola 68HC12 Microcontroller acted as the hub of our entire project. It allowed us to remain flexible while providing the computational power we needed to analyze our data. We picked the 68HC12 as our microcontroller because of the amount of support available to us, and because we already had three 68HC12s available to us.

The 68HC12 has two main connections: ribbon cabling to the Game Boy Cameras, and an RJ-45 connection to the navigation subsystem, as can be seen in both Figure 2 and Figure 3. The 68HC12 is powered by a +5 volt line.

The Game Boy Camera

In order to detect the golf ball or the hole, we decided to use a series of three fixed cameras positioned at a height of four feet above the green. These cameras would take pictures of their corresponding areas and relay this information to the 68HC12. If the 68HC12 detected the ball or the hole in the range of any of the cameras, then we would be able to calculate the distance and angle to the ball or hole.

In actuality, we did not use the entire Game Boy Camera. Rather, we disassembled the Game Boy Camera Cartridge and used the Mitsubishi Integrated Circuit M64282FP Image Sensor (Artificial Retina LSI), which is included with any Game Boy Camera. The M64282FP (camera chip) is a 128 x 128 pixel CMOS image sensor with built-in image processing and analog image output functions. The camera chips we used are the basis of our detection system. Without them, both our hardware and software designs would be significantly different. We picked this particular camera for a few basic reasons. Each individual camera is relatively cheap, they are black and white cameras, and the resolution is only 128 x 128. By purchasing a cheap camera, this allowed us to purchase more than one camera for the project. This allowed us to increase the range of our detection system without depending on one camera to operate correctly. We picked a black and white camera because it simplifies our searching algorithm. If we were to deal with color in our system, there would be many more variables that we would have to deal with. The fact that we have a 128 x 128 resolution for our camera decreases the amount of actual data we have to work with. While a better resolution could possibly increase our range, this smaller design allowed us to stay within our limits as far as

memory allocation is concerned. Overall, this camera appeared to be the best possible solution for our problem.

In order to communicate with the cameras, there are nine lines leading from the camera to the 68HC12. These lines are xclk, aout, read, start, load, xrst, sin. Gnd, and Vcc (+5 volts). These can be seen in detail in Figure 3, the wiring diagram for the 68HC12 to the M64282FP. As for the actual manipulation of these lines, a more detailed description can be found in the software section on page 6.

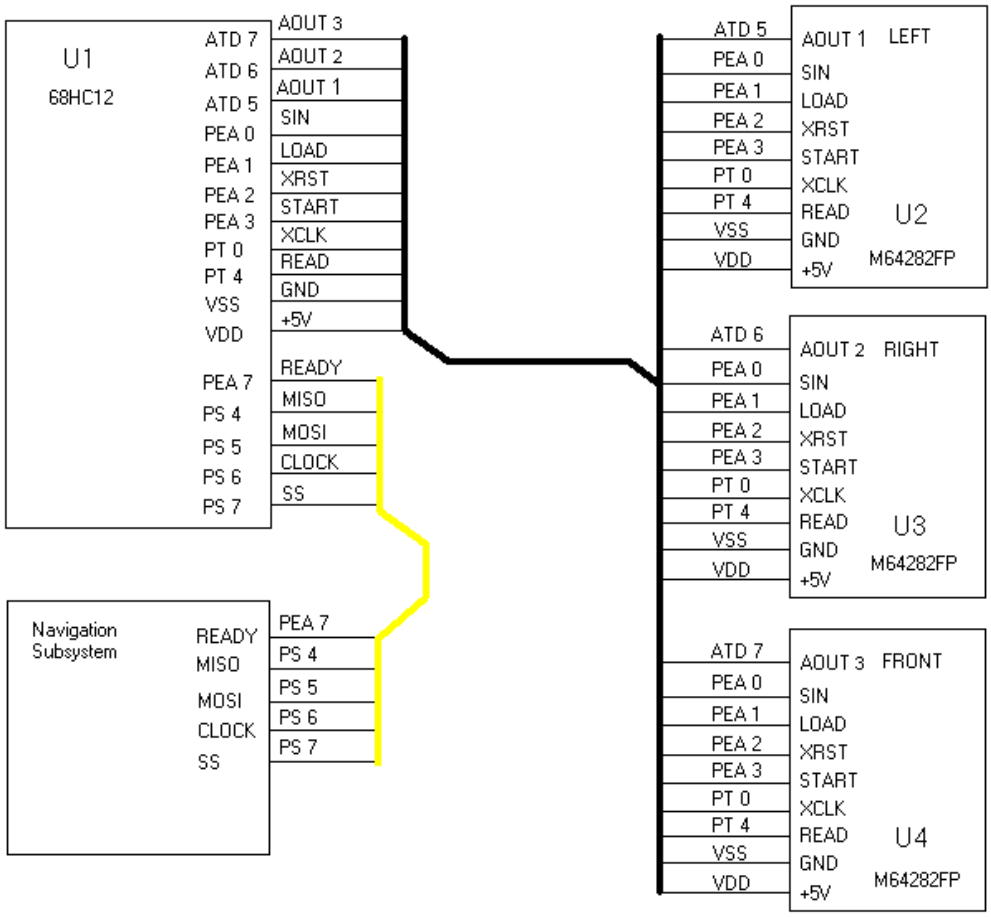


Figure 3 – Wiring Diagram

The Casings

The casings, while sounding fairly simple, actually took quite a lot of work. We needed a system which would hold our microcontroller system, allow for wiring to the outside world, and protect our system from various elements. We also needed a casing which would encase our camera system as well, but one that would also allow us to move our cameras up to four feet away from the casing of the microcontroller, and still be able to read images from the cameras.

The Microcontroller Casing

In order to satisfy our needs for a microcontroller housing, we disassembled an old power supply casing and modified it for our needs. In order to do this, we drilled mounting holes for the HC12, and added serial port holes for camera connectivity, other various features such as a status LED board, and a reset switch. The modifications for the casing of the 68HC12 accomplished two things: increasing the ease of use of our system and the mobility of our system. We increased the ease of use of our design by mounting external serial and power connectors on the outside of the case. These connectors allowed us to work fairly quickly without being concerned about the specifics of wiring. Our power and camera connectivity is thus made simple for the average user.

The casing also allowed us to increase the mobility of our system. By providing a simple set of power and interface connections, and consolidating the system into a single box, the system is made highly mobile. The system can be used anywhere a 5V power supply is available, which in our case means that the system can be easily installed on the completed robot.

The Game Boy Camera Casing

The camera's casing is a fairly simple design. The only specification for the casing design is that it protects the camera from excess light. While indoors, this is not really a concern. However, in order to operate outdoors in the bright sun, we needed a plan of blocking out excess light. Unfortunately, intense light as that from the sun is actually powerful enough that the camera pixels can be affected by the amount of light seeping in through the back of the chip, not only through the lens in the front. The casing is made out of a hollowed out golf ball shell, which has been painted black in order to block out as much sun as possible. The shell was sealed with hot glue and painted black one more time to ensure complete darkness on the inside of the case. While the camera peaked out through one tiny hole, we still ran into the problem of wide-angle light beams coming in and blinding our camera. In an attempt to decrease this wide-angle light, we added a cone of poster board to each camera. These cones surrounded the lens of the camera and extended outwardly at a 20 degree angle to the line of vision.

Calibration

In order to function properly, the ball/hole system must be properly calibrated for its conditions. Gain and exposure must be set to the appropriate levels so that the golf ball or hole is the brightest object seen by the camera. The following steps can be followed to calibrate the camera system:

1. point the camera at the type of surface the image is to be taken over, i.e. Astroturf, carpet, dark wood floor, and place the golf ball on it.
2. Set the gain to 5 and the exposure time to 0x00 in register 2 and 0x60 in register 3 (for more information on registers refer to the imaging section of this paper or the data sheet section 11.6, Appendix 2).
3. Compile and run the code. If there are no white pixels found, raise the value in register 3 by 0x10. Take another picture. If no white is found continue raising the value in register 3. If the value becomes 0xF0 raise register 2 by 0x01 amount, and start register 3 at 0x00. If the exposure time becomes 0x0FFF, raise the gain by 0x01 and start the time registers over at 0x0050.

Once white pixels are found, continue to raise the exposure time until about 18±3 white pixels are detected. At this point, calibration is complete. If there are over 25 white pixels found, then either the gain is too high, or other bright objects are in the field of view. We found that the lighting conditions in the digital lab required a gain of 5 and an exposure time of about 0x0060.

System Strengths

The ball/hole system has many strengths. First, under controlled circumstances, i.e. inside a building, under fluorescent lighting, the system can detect the ball/hole and find a distance and heading to it. The system can locate the ball/hole within about a one meter radius.

Second, the system is modular. Connections are simple for customers to make since the cameras use standard 9-pin serial connectors. Communication connections are also made simple through the RJ-45 cable. It is also expandable. Additional cameras can be added easily and require only minor modification to the code.

Third, the system is totally encased. All cameras are housed in containers, and the 68HC12 is contained in an aluminum box. This cuts down on static interference, as well as protecting the 68HC12 from the elements.

System Limitations

The ball/hole system, while functional in most respects, does suffer from some limitations. The most glaring limitation is its inability to function in sunlight. The system cannot handle the intensity and spectrum of sunlight. The cameras also cannot adjust to varying brightness, making the outdoors an impossible environment to function in. Attempts to filter light using IR and light filters have been unsuccessful. The system does

function inside, however, under fluorescent lighting. The controlled environment allows for successful use of the ball/hole system.

Production Costs

The following table is a breakdown of our production costs for the project.

Purchases	Quantity	Price per each	Total	Donated quantity	Total donated
Gameboy cameras	4	$25.00	$100.00	4	$100.00
Infrared filter	1	$38.77	$38.77		
Golf ball housing for cameras	3	$2.00	$6.00	3	$6.00
Poster board	1	$0.75	$0.75	1	$0.75
L angle mounting brackets	3	$0.12	$0.36		
Male serial port connectors	3	$2.00	$6.00		
Female serial port connectors	3	$2.00	$6.00		
9 pin ribbon cable	12	$0.25	$3.00		
Screws	22	$0.05	$1.10	22	$1.10
Washers	3	$0.05	$0.15	3	$0.15
Nuts	5	$0.05	$0.25	5	$0.25
Power connection	1	$0.60	$0.60	1	$0.60
Female pinheaders	18	$0.17	$3.06	18	$3.06
LED's	4	$1.00	$4.00		
CAT5 connection	1	$2.87	$2.87	1	$2.87
Testing Materials					
Wood dowels	2	$0.67	$1.34	2	$1.34
Tape	1	$1.73	$1.73	1	$1.73
Case	1	$5.00	$5.00		
Misc.					
Ultraviolet filters	2	$7.00	$14.00	2	$14.00
Sona Switch	1	$75.48	$75.48		
Subtotal			$270.46		
Total donated			$131.85		
Total cost to department			$138.61		
Reproduction cost			$393.64*		

*The reproduction costs includes approx. $200 for a new 68HC12 w/ memory expansion

Table 1 – Production Costs

Conclusion

We feel that we were faced with a difficult task. Image processing and detection is a complex field with numerous challenges. However, despite the difficulty of the task, we believe that we have created a good basic ball and hole detection system. It can find a white ball on a green surface, compute the distance and angle to the ball, and relay this information to the navigation subsystem. Unfortunately, it only functions under ideal conditions, but we believe that with further time, we could overcome many of the obstacles that we face in designing a functional system to work outdoors.

References

Web Sites

"http://www.ee.nmt.edu/~wedeward/EE382/SP02/ee382.html" - EE382 class website.

"http://homepages.paradise.net.nz/~vkemp/gbcam.htm" - Gameboy Camera Parallel Port interface

"http://www.motorola.com" - Motorola, Inc.

"http://www.cliffshade.com/dpfwiw/ir.htm#testing" - Infrared Photography with Digital Cameras

"http://www.digitalphotobasics.com/article1008.html" - Filter Basics

Appendix 1 – The Code

```
//THE CODE TO RUN!!!!


#include <hc12.h>
#include <DBug12.h>
#define START 0x2000            //where the image is stored
#define WHITE 0xef              //what color will be considered white
#define MEMPROB 0x380   //For checking memory problem
#define fieldofview
#define depthofview
#define THRESHOLD 20           //number of pixels it takes to make a
ball/hole
#define xquad
#define yquad
#define D_1MS (8000/4)

void delay(unsigned int ms);

void initialize();
volatile int duh;
volatile char value;
volatile char chair;
volatile char regnum;
volatile unsigned int cameranum;
volatile unsigned int a;
volatile unsigned int tempclk;
volatile unsigned int xquad;
volatile unsigned int yquad;
volatile unsigned int xplace;
volatile unsigned int yplace;
volatile unsigned int searchplace;
volatile unsigned int badpic;
volatile unsigned int sendangle;
volatile unsigned int senddistance;
volatile unsigned char *mempos;
//a=(fieldofview/8);
//regnum=(360/a);
//unsigned char angle[] = {(regnum-3)*a,(regnum-2)*a,(regnum-1)*a,360-
(a/2),a-(a/2),a,a*2,a*3};
//unsigned char angle[] = {315,330,345,353,7,15,30,45};
//unsigned char distance[] = {16,14,12,10,8,6,4,2};
const unsigned int searchangle[] = {45,270,45,45,270,135,
45,270,75,45,270,75,45,270,75,45,270,75,45,270,75,45,270,75,45
,270,75,45,270,75,45,270,75,45,270,75,45,270,75};
const unsigned char searchdistance[] =
{0,0,200,0,0,50,0,0,100,0,0,100,0,0,100,0,0,100,0,0,100,0,0,100,
0,0,100,0,0,100,0,0,100,0,0,100,0,0,100};
const unsigned int registers[] = {0x00a0,0x0105,0x0200,
0x0360,0x0401,0x0500,0x0601,0x0726}; //- < provides 2 V p-p?
```

```
const unsigned char distance[8][8] = {0, 0, 0, 97, 97, 0, 0, 0, 0, 0, 0,
85, 85, 0, 0, 0, 0, 0, 0, 67, 67, 0, 0, 0, 0, 0, 0, 52, 52, 0, 0, 0, 0,
0, 0, 37, 37, 0, 0, 0, 0, 0, 0, 25, 25, 0, 0, 0, 0, 0, 0, 14, 14, 0, 0,
0, 0, 0, 0, 4, 4, 0, 0, 0};

const unsigned int angle[8][8] = {325, 332, 341, 0, 0, 19, 28, 35, 320,
326, 339, 0, 0, 21, 34, 40, 312, 329, 335, 0, 0, 25, 39, 48, 304, 313,
332, 0, 0, 28, 47, 56, 295, 308, 325, 0, 0, 35, 52, 65, 287, 298, 317,
0, 0, 43, 62, 73, 280, 280, 300, 0, 0, 60, 80, 80, 275, 275, 273, 0, 0,
273, 275, 275};




void main()
{
      searchplace = 0;
      DDRD = 0x01;      //CONFIRM THAT This is A tx, B RX.
      PORTEA=0x10;      //LED stuff
      SP0CR1 = 0x4c;          //01001100
      SP0CR2 = 0x00;
      SP0BR = 0x07;
      duh = 1;
      DDRS = 0x10;
      //delay(500);
      while(1){
            /*if( (PORTEB & 0x01) == 1)
                  {
                  searchplace = 0;
                  PORTEA=0x10;            //LED stuff
                  }*/
            if ((PORTS & 0x80) == 0){
                  delay(100);
                  action();
                  delay(100);
                  //transmit();
                  }

              //  action();
            PORTEA = (PORTEA | 0x10);
                //delay(500);
            }

}

void transmit(){


      char manglechar;
      char langlechar;
      char distance;
      int duh;
//    senddistance = 0xaa;
//    sendangle = 0xaa;
      distance = senddistance;
```

```
//    DBug12FNP->printf("Beginning transmission...%x\n\r ", distance);
      SP0DR = distance;             // load register - distance 1
      delay(10);
      PORTEA = (PORTEA | 0x80);
      while ((PORTS & 0x80) == 0x80);
      //delay(1);
      PORTEA = (PORTEA & ~0x80);
//    DBug12FNP->printf("Sent first D %x\n\r ", SP0DR);
      while ((PORTS & 0x80) != 0x80);
      duh = SP0SR; //while ((SP0SR & 0x80) != 0x80);
//    DBug12FNP->printf("Sent D %x\n\r ", SP0DR);

      delay(5);
      langlechar = sendangle;
      /*manglechar = (sendangle >> 8);
      manglechar = (manglechar | 0x02);
      manglechar = (manglechar & 0x03);
      manglechar = 0xaa;*/
      manglechar = sendangle/16;
      manglechar = (manglechar | 0x02);
      manglechar = (manglechar & 0x03);

      SP0DR = manglechar; //load register - angle 1
      PORTEA = (PORTEA | 0x80);
      while ((PORTS & 0x80) == 0x80);
      PORTEA = (PORTEA & ~0x80);
      while ((PORTS & 0x80) != 0x80);
      duh = SP0SR;//    while ((SP0SR & 0x80) != 0x80);
//    DBug12FNP->printf("Sent A1 %x \n\r ", SP0DR);

      delay(5);
      langlechar = 0x11;
      SP0DR = langlechar; //load register - angle 2
      PORTEA = (PORTEA | 0x80);
      while ((PORTS & 0x80) == 0x80);
      PORTEA = (PORTEA & ~0x80);
      while ((PORTS & 0x80) != 0x80);
      duh = SP0SR;//    while ((SP0SR & 0x80) != 0x80);
//    DBug12FNP->printf("Transmission complete! %x \n\r ", SP0DR);

      return;


}

void delay(unsigned int ms){
      int i;
      while (ms>0){
            i=D_1MS;
            while(i>0)
            {
                  i--;
            }
            ms--;
      }
}
```

```
void action ()
{
  DBug12FNP->printf("Action!!\n\r ");
  DDRD=0x01;
  PORTEA=0x84;

/*  TIOS=0x01;     // makes bit 0 an output compare all others input
capture
  TSCR=0x80;     // enables timer
  TCTL2=0x01;    // sets bit 0 to toggle on successful compare
  TCTL4=0xa8;    // makes bit 1 (reg), 2(xreset), and 3 (start)interrupt
on a falling edges
  TCTL3=0x09;    // makes bit 4 (img ready) rising and bit 5 falling
edges
  TMSK1=0x05;    // MAKES BIT 0 (xclk) AND 2 (xreset) ABLE TO MAKE
INTERRUPTS
  TMSK2=0x00;    // does nothing
  TFLG1=0x05;    // clears bit 0 and 2 to cause interrupts

  ATDCTL2 = 0x80;
  ATDCTL3 = 0x00;
  ATDCTL4 = 0x00; */
  ATDCTL5 = 0x27;

initialize();
//TMSK1=0x05;    // MAKES BIT 0 (xclk) AND 2 (xreset) ABLE TO MAKE
INTERRUPTS

/*  chair=5;
  regnum=0;
  a=0;
  mempos=(unsigned char *) START;
  xquad=0;
  yquad=0;
  xplace=1;
  yplace=1;*/

  cameranum=1;
  enable();

while (321!=123)
   {
      cameranum = duh;
      if (TMSK1==0x00 & (badpic == 0))
      {
            //DBug12FNP->printf("starting calc\n\r ");
            //DBug12FNP->printf("cameranum=%d\n\r",cameranum); //print
            //DBug12FNP->printf("mempos: %x\n\r",mempos);

            if (*mempos>WHITE)                          //compare
            {
                  xquad=xquad + (xplace/16);              //adding x
pixel value
                  yquad=yquad + (yplace/16);          //adding y pixel
value
```

```
                  //DBug12FNP->printf("xplace: %d  xquad: %d
",xplace/16,xquad);
                  a++;
                  //DBug12FNP->printf("a=%d\n\r",a);
            }

            xplace++;
            mempos=mempos+1;

            if (xplace==128)
            {
                  xplace=0;
                  yplace++;

            //DBug12FNP->printf("a=%d\n\r",yplace);
            }



if ((xplace==0) & (yplace==123))  //123
{
  xquad=(xquad/a);
  yquad=(yquad/a);

  if ((a>=THRESHOLD) & (cameranum==1))
  {
  searchplace = 0;
  DBug12FNP->printf("xquad=%d yquad=%d a=%d\n\r", xquad,yquad,a);

        if (xquad != 3 & xquad !=4)
        {
        sendangle=angle[yquad][xquad];
        senddistance=0;
        DBug12FNP->printf("sendangle=%d senddistance=%d - FOUND!! camera
%d - FORWARD\n\r",sendangle, senddistance, cameranum);
        return;
       //  transmit();
        }
        else
        {
        sendangle=angle[yquad][xquad];
        senddistance=distance[yquad][xquad];
       //  transmit();
        DBug12FNP->printf("sendangle=%d senddistance=%d - FOUND!! camera
%d\n\r - FORWARD",sendangle, senddistance, cameranum);
        return;
        }
        //die();
  }
  else if ((a>=THRESHOLD) & (cameranum==2))
  {
        searchplace = 0;
        sendangle=90;
        senddistance=0;
        DBug12FNP->printf("sendangle=%d senddistance=%d - FOUND!! camera
%d - RIGHT\n\r",sendangle, senddistance, cameranum);
        cameranum=1;
```

```
        duh = 1;
        ATDCTL5=0x27;
        PORTEA=(PORTEA & ~0xf0) | 0x20;
      //  transmit();

        return;
  }
  else if ((a>=THRESHOLD) & (cameranum==3))
  {
        searchplace = 0;
        sendangle=270;
        senddistance=0;
        DBug12FNP->printf("sendangle=%d senddistance=%d - FOUND!! camera
%d - LEFT\n\r",sendangle, senddistance, cameranum);
        cameranum=1;
        duh = 1;
        ATDCTL5=0x27;
        PORTEA=(PORTEA & ~0xf0) | 0x20;
      //  transmit();

        return;
  }



            /*if ((xplace==0) & (yplace==123))  //123
            {
                    xquad=(xquad/a);
                    yquad=(yquad/a);

                  if ((a>=THRESHOLD) & (cameranum==1))
                    {

                        //  DBug12FNP->printf("xquad=%d yquad=%d a=%d
mempos%x\n\r", xquad,yquad,a,mempos);
                        //  DBug12FNP->printf("angle=%d
distance=%d,\n\n\r",angle[xquad],distance[yquad]);

                          if (xquad != 3 & xquad !=4)
                          {
                                sendangle=angle[yquad][xquad];
                                senddistance=0;
                                  DBug12FNP->printf("angle=%d
distance=%d,\n\n\r",sendangle,senddistance);
                                return;
                          }
                          else
                          {
                                sendangle=angle[yquad][xquad];
                                senddistance=distance[yquad][xquad];
                                //if (THRESHOLD==2) sendangle+=90;
                              //  if (THRESHOLD==3) sendangle+=270;
                                DBug12FNP->printf("angle=%d
distance=%d,\n\n\r",sendangle,senddistance);
                                 return;
                          }
                          //die();
```

```
                    }
                    else if ((a>=THRESHOLD) & (cameranum==2))
                    {
                          sendangle=90;
                          senddistance=0;
                          cameranum=1;
                          return;
                    }
                    else if ((a>=THRESHOLD) & (cameranum==3))
                    {
                          sendangle=270;
                          senddistance=0;
                          cameranum=1;
                          return;
                    }*/




  else if (a<THRESHOLD)
      {
        //DBug12FNP->printf("xquad=%d yquad=%d
arr=%d\n\n\r",xquad,yquad,cameranum);
      //DBug12FNP->printf("NO WHITE FOUND!\n\r");
        cameranum++;
        if (cameranum==2)
        {
              ATDCTL5=0x26;
              PORTEA=((PORTEA & ~0xf0) | 0x40);
              //DBug12FNP->printf("cam2 cameranum %d\n\n\r", cameranum);
              //DBug12FNP->printf("cameranum = %d\n\r", cameranum);
        }
        if (cameranum==3)
        {
              ATDCTL5=0x25;
              PORTEA=((PORTEA & ~0xf0) | 0x80);
      //      DBug12FNP->printf("cam3\n\n\r");
        }
        if (cameranum==4)
        {
              sendangle = searchangle[searchplace];
              senddistance = searchdistance[searchplace];
              searchplace++;
             // transmit();
              DBug12FNP->printf("sendangle=%d senddistance=%d - NOT
FOUND\n\n\r",sendangle, senddistance);

              cameranum=1;
             duh = 1;
              PORTEA=((PORTEA & ~0xf0) | 0x10);
            return;
        }
        duh = cameranum;
        TMSK1=0x05;
        initialize();
        enable();
```

```
        //a=6342;
        //DBug12FNP->printf("end arro %d\n\n\r", cameranum);
  }




                    /*else if ((a<THRESHOLD))
                     {                     // DBug12FNP->printf("a=%d
xquad=%d yquad=%d mempos=%x cameranum = %d\n\n\r",a,xquad,yquad,mempos,
cameranum);
                            DBug12FNP->printf("NO WHITE FOUND!\n\r");
                  //          if(cameranum==1)
                  //                cameranum =2;
                  //          else if (cameranum ==2)
                  //                cameranum = 3;
                  //          else if(cameranum == 3)
                  //                cameranum = 4;
                              //  die();
                                       // duh = cameranum;
                                       // duh++;
                                       // cameranum = 23;
                            //cameranum = cameranum + 1;
                  //                    DBug12FNP->printf("cameranum =
%d\n\r", cameranum);
                            if (cameranum == 3)
                                {
                                  if (searchplace == 42){
                                      searchplace = 0;}
                                  sendangle = searchangle[searchplace];
                                  senddistance =
searchdistance[searchplace];
                                  //if (searchplace == searchplace)
                                           DBug12FNP->printf("
Searchpattern %d, a=%d sendangle=%d senddistance=%d mempos=%x cameranum
= %d\n\n\r",searchplace,a,sendangle,senddistance,mempos, cameranum);

                                  searchplace = searchplace+1;
                                  cameranum = 4;

                                  PORTEA= (PORTEA & ~0x40);         //LED
stuff
                                  //return;

                                }
                           if (cameranum==2)
                                {
                                  DBug12FNP->printf("gate2");
                                  cameranum = 3;
                                  ATDCTL5=0x25;
                                  PORTEA= PORTEA & ~0x20;
                                  PORTEA = PORTEA|0x40;       //LED stuff
                                }
                             if (cameranum==1)
                                {
```

```
                                DBug12FNP->printf("gate1");
                                cameranum = 2;
                                ATDCTL5=0x26;
                                PORTEA=PORTEA | 0x20; //LED stuff

                              }
                            if(cameranum == 4){
                                cameranum = 1;
                                duh = 1;
                                return;
                                }

                            //if (cameranum==3) die();


                           //  mempos=(unsigned char *) START;
                            TMSK1=0x05;
                            initialize();
                            //DBug12FNP->printf("mempos: %x\n\r",mempos);
                            enable();
                        }*/
                  }
      }
      if (badpic == 1){
            badpic = 0;
            initialize();
            enable();
      }
}}


void initialize()
{
  badpic = 0;
  chair=5;
  regnum=0;
  a=0;
  tempclk=720;
  *(unsigned char *) (START+0x4002) = 0x00;
  mempos=(unsigned char *) START;
  xquad=0;
  yquad=0;
  xplace=0;
  yplace=0;
//DBug12FNP->printf("init");
  TIOS=0x01;     // makes bit 0 an output compare all others input
capture
  TSCR=0x80;     // enables timer
  TCTL2=0x01;    // sets bit 0 to toggle on successful compare
  TCTL4=0xa8;    // makes bit 1 (reg), 2(xreset), and 3 (start)interrupt
on a falling edges
  TCTL3=0x09;    // makes bit 4 (img ready) rising and bit 5 falling
edges
  TMSK1=0x05;    // MAKES BIT 0 (xclk) AND 2 (xreset) ABLE TO MAKE
INTERRUPTS
  TMSK2=0x00;    // does nothing
  TFLG1=0x05;    // clears bit 0 and 2 to cause interrupts
```

```
  ATDCTL2 = 0x80;
  ATDCTL3 = 0x00;
  ATDCTL4 = 0x00;
}

@interrupt void getimage(void)    //uses bit 5
{
      if (cameranum==1) *mempos=ADR3H;
      if (cameranum==2) *mempos=ADR2H;
      if (cameranum==3) *mempos=ADR1H;
      mempos=mempos+1;
      TFLG1=0x20;
}


@interrupt void readydone(void)       //uses bit 4 for image ready
{
      tempclk=720;
      if ((TCTL3 & 0x01) == 0x01)
      {
            TCTL3=((TCTL3 & ~0x01) | 0x02);      //changes to now look
for falling edges
            TMSK1=(TMSK1 | 0x20);                // turns on getimage
      }
      else
      {
            TCTL3=((TCTL3 & ~0x02) | 0x01);
            TMSK1= 0x00;                            //turns off all
interrupts
            disable();
            //mempos=((unsigned char *) START + MEMPROB);
            //value=(*mempos & 0xe0);
            //regnum=(*(mempos+MEMPROB) & 0xe0);
            //chair=*(char *) 0x6002;
            if (*(char*)(START+0x4002) !=0x00)
            {
                  mempos=((unsigned char *) START + MEMPROB+2);
                  badpic = 1;
                  cameranum = cameranum - 1;
            }
            else
            {
            mempos=((unsigned char *) START);
            }
            a=0;
      }
      TFLG1=0x10;
}


@interrupt void start(void)           //uses bit 3 to start the system
{
  if (a==0)
  {
        PORTEA=(PORTEA | 0x08);             //raises the start line
```

```
        a++;                                  //the grade we are going to
get
  }
  else
  {
       PORTEA=(PORTEA & ~0x08);             //lowers the start bit
       TMSK1=((TMSK1 & ~0x08) | 0x10);      //turns off interrupt 3 and
on interrupt 4
       tempclk=150;
 }
 TFLG1=0x08;
}


@interrupt void xreset(void)          //uses bit 2 to reset system
{
  if (a==0)
  {
        PORTEA=(PORTEA & ~0x04);             //lowers bit 2 of EA for
reset
        a++;                                 //what kind of code this is
  }
  else
  {
        PORTEA=(PORTEA | 0x04);              //raises the reset line
        TMSK1=((TMSK1 & ~0x04) | 0x02);      //turns off interrupt 2 and
on interrupt 1
  }
  TFLG1=0x04;
}


@interrupt void setregisters(void)    //uses bit 1 to put out data on
falling edge
{
  if (regnum<8)                          //number of bytes shifted out
  {
        a = (((registers[regnum] << chair) & 0x8000)==0x8000);
        chair++;
        if (a==1)
         {PORTEA |=0x01;}                  //writes a 1 to PORTEA
        else
         {PORTEA &= ~0x01;}                //writes a 0 to PORTEA
  }
  else
  {
       TMSK1=((TMSK1 & ~0x02) | 0x08);    //turns off interrupt 1 and on
interrupt 3
        chair=6;
        a=0;                                 //imPORTEAnt for later
interrupt
  }
  TFLG1=0X02;
}
```

```
@interrupt void tci0(void)    //uses bit 0 to toggle output to make
system clk 20KHz
{
  TC0+=tempclk;  //720
  if (chair==6)                //returns load line low
  {
        PORTEA = (PORTEA & ~0x02);
  }
  if (chair==16)               //checks to see if load line needs to go
high
  {
        PORTEA= (PORTEA | 0x02);
        chair=5;
        regnum++;
  }
  TFLG1=0x01;
}
```

Appendix 2 – Vector.c

```
/*      INTERRUPT VECTORS TABLE 68HC12
 */
void tci0 ();
void setregisters ();
void xreset();
void start();
void readydone();
void getimage();

void (* const _vectab[])() = {   /* 0x0B10 */
        0,              /* BDLC             */
        0,              /* ATD              */
        0,              /* reserved         */
        0,              /* SCI0             */
        0,              /* SPI              */
        0,              /* Pulse acc input */
        0,              /* Pulse acc overf */
        0,              /* Timer overf      */
        0,              /* Timer channel 7 */
        0,              /* Timer channel 6 */
        getimage,       /* Timer channel 5 */
        readydone,      /* Timer channel 4 */
        start,          /* Timer channel 3 */
        xreset,         /* Timer channel 2 */
        setregisters,   /* Timer channel 1 */
        tci0,           /* Timer channel 0 */
        0,              /* Real time        */
        0,              /* IRQ              */
        0,              /* XIRQ             */
        0,              /* SWI              */
        0,              /* illegal          */
        0,              /* cop fail         */
        0,              /* cop clock fail   */
        (void *)0xff80, /* RESET            */
    };
```