University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machine Design Laboratory

Final Report:
Room Positioning System
Tom and Jerry

Craig Ruppel
Spring 1999

Table of Contents

Executive Summary

This project was developed because of the need for some sort of accurate positioning system. Once developed, this system can be used for numerous behaviors such as mapping and robot communication.

The positioning of the robot on the plane of the floor is achieved by using three beacon transmitters and one robot receiver. The three beacons are spread in a triangle in a room. These beacons transmit sonar and RF pulses in sequence so that a distance is obtained from each beacon. As the robot moves around in the room it receives new coordinates from these beacons.

For this project two Motorola 68HC11s were used, one for the beacon controller and one for the actual robot. I used the Talrik body as my platform base. Once the positioning system was created, the idea was to use these coordinates to have one robot chase another. One robot would transmit its coordinates and the other robot would receive these coordinates and move to that position.

While transmitting coordinates to another robot was not a success, the actual positioning system that was created will be useful to other students in the future.

Introduction

In the world of robots, one's position may seem unimportant information. After all, a

robot does not need to know where it is in a room if it needs to perform a simple task.

However, once more difficult ideas are presented a positioning system of some kind is necessary. For example, positioning could ease the difficulty of mapping a room and could be used to replace random behavior algorithms.

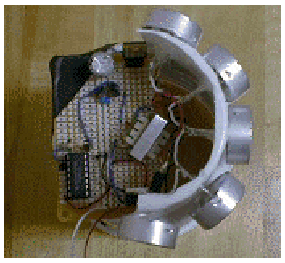
The need for an accurate positioning system led us to design the RPS system. In this paper I will talk about how it was created and also what should be done in the future to improve this important design.

Integrated System

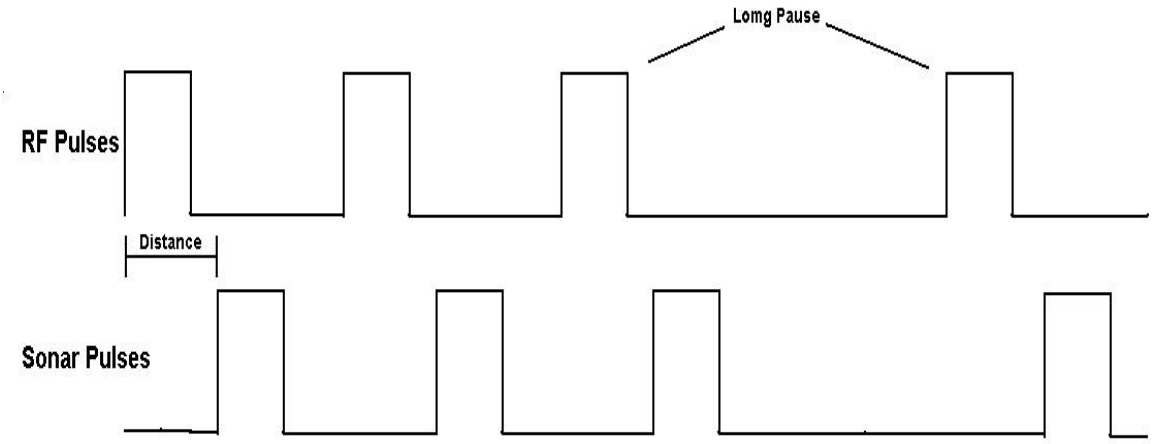
There are two main parts to the integrated system of this project: the transmitter and the receiver

Transmitter-

The RPS transmitter consists of three sonar beacons and one MHC6811. Each beacon has five sonar transducers on the front in order to give a wide area of transmittance.



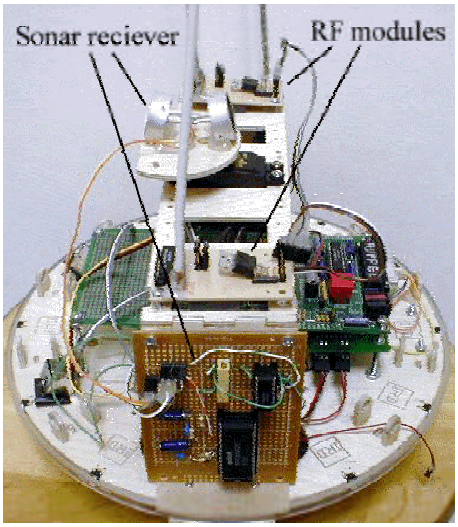
The functional layout for the controller is relatively simple. The three beacons emit a single pulse in the same sequence followed by a long pulse after the third beacon. This pause helps determine which beacon is the first beacon. At the same time each beacon pulses, a RF pulse is also released.



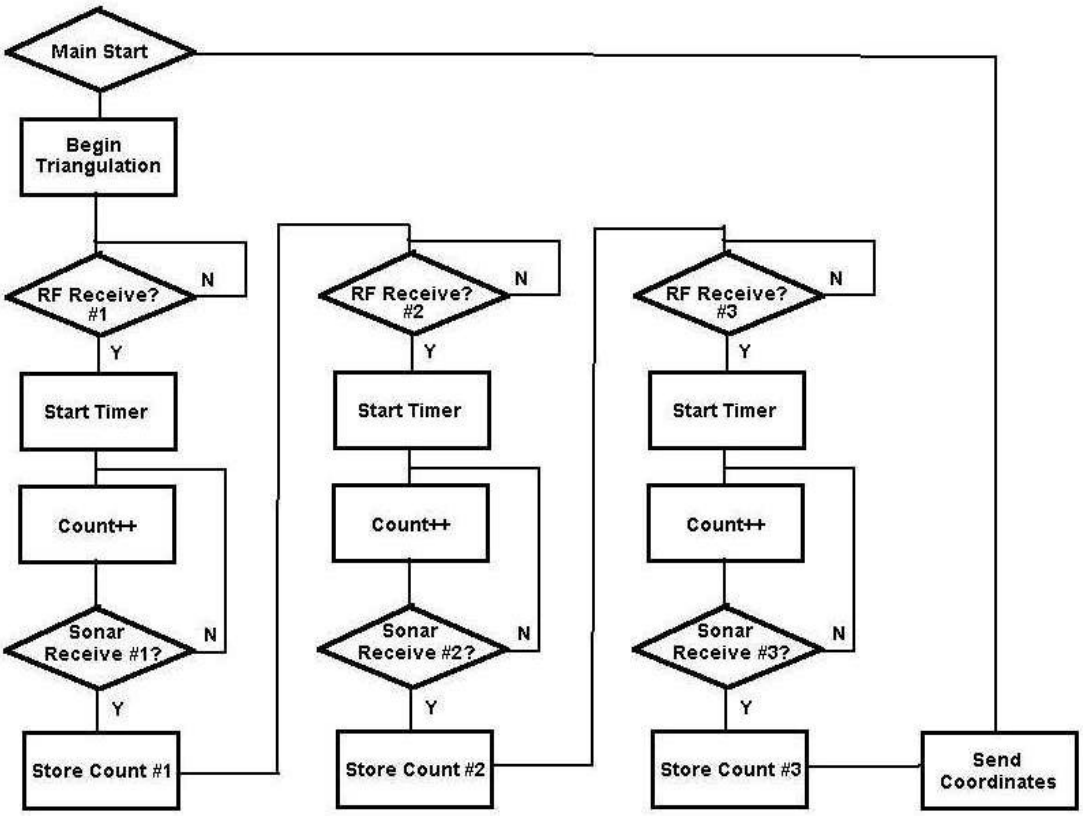
By calculating the time it takes between receiving the RF pulse and receiving the sonar pulse one can determine the distance away from that beacon. The actual design that was used for the controller circuit will be discussed in the sensor section.

Receiver-

The RPS receiver consists of two RF boards and sonar receiving circuitry wired to a string of connected transducers. These three components are relatively small and can be attached easily to most platforms.



The receiving end will first look for the three-pulse sequence in order to identify the first pulse. Next it will start timing loops in order to determine the distance from the beacons.



This software works correctly as shown and even transmits the coordinates on HyperTerminal. The coordinates are sent through a different frequency RF to a computer connected to the receiving RF board.

Mobile Platform

Since this project is based mainly on the triangulation system the mobile platform is not as important. I decided to use a Talrik platform so that I could get it put together quick and start working on the triangulation system. The Talrik platform is about 10 inches in diameter and about 10 inches tall. There are two hacked motors and one head servo that is not entirely necessary. The head is where I mounted the sonar receivers for the RPS system. Since I used three placed at 120-degree angles, the head should not need to turn. The bridge of the body is also where both RF boards were placed. The wood body that I used made it easy to experiment with the location of the RPS modules. In the end, it is better to have the sonar receiver mounted at the top of your robot with no obstructions.

Actuation

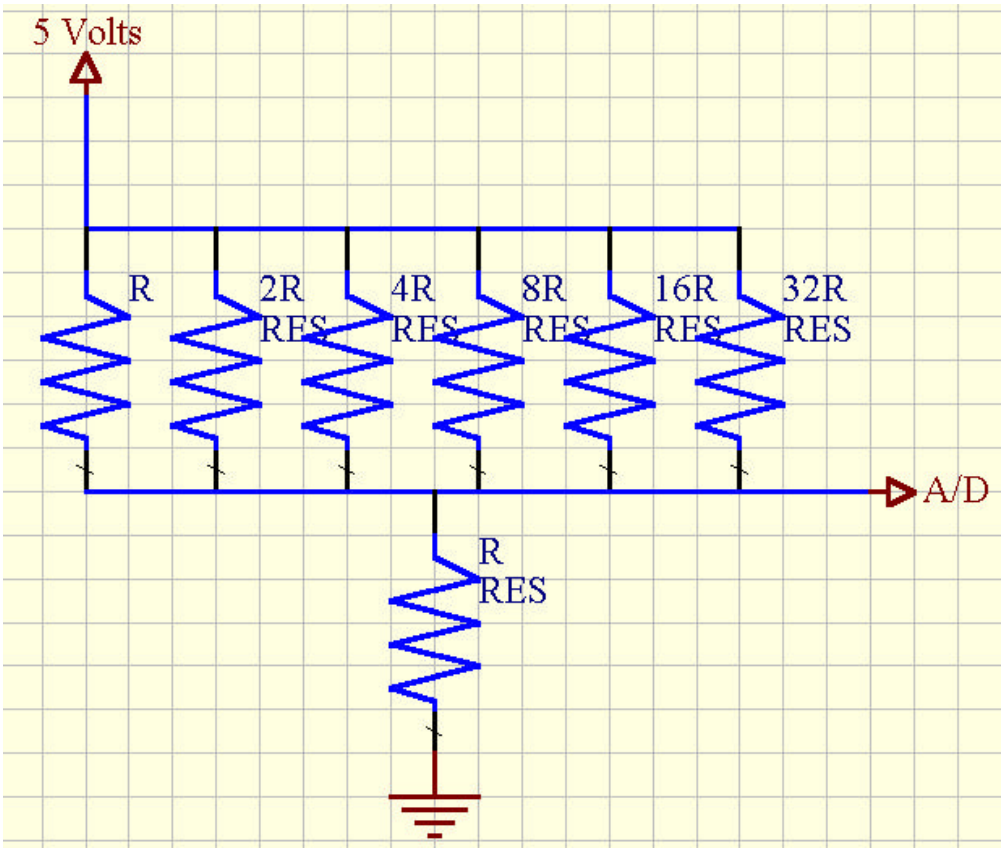
Once again, the major concern for this project was the RPS system. The Talrik platform that I created used two hacked servos for motors. After hacking the servos (this is described in the Talrik assembly manual) they become DC gearhead motors. These 42 oz/inch motors are enough to move the robot around. The head servo only has 180 degrees of rotation and can be used to position the sonar transducers if they are receiving a bad signal.

Sensors

Ah, finally the bread and butter of this project. In this section I will discuss the set up of all of the sensors necessary to recreate the RPS system. First, I will talk about the IR and bump switches that are used for obstacle avoidance and collision detection. Then I will talk about the sonar and RF that are used for the triangulation.

Bump sensor suite-

The bump sensor suite on my robot consist of ten bum switches and a voltage divider circuit. The five sensors on the rear of the platform are wired together while the front five are all wired separately. The voltage divider circuit I used as a suggestion from Scott Jantz returns a different analog value depending on which switch is depressed.(R = 10k)



IR Sensor Suite-

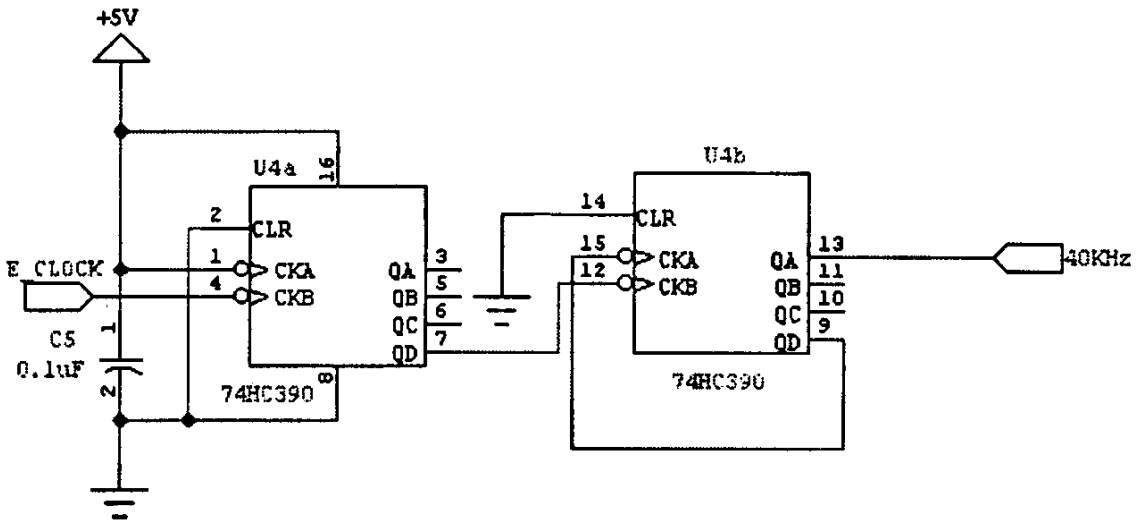
I used four IR emitters and Two IR detectors on the front of my platform. The closer an object gets to the robot the high the analog value will become. The two Detectors were positioned like eyes on the front of the platform. I found that two detectors would be enough to avoid most objects coming from most angles (you have to test those bump sensors sometime!).

Sonar Sensor Suite-

Sonar was a very important part of this project. We used 15 sonar transducers on the transmit beacons and three on the receiver.

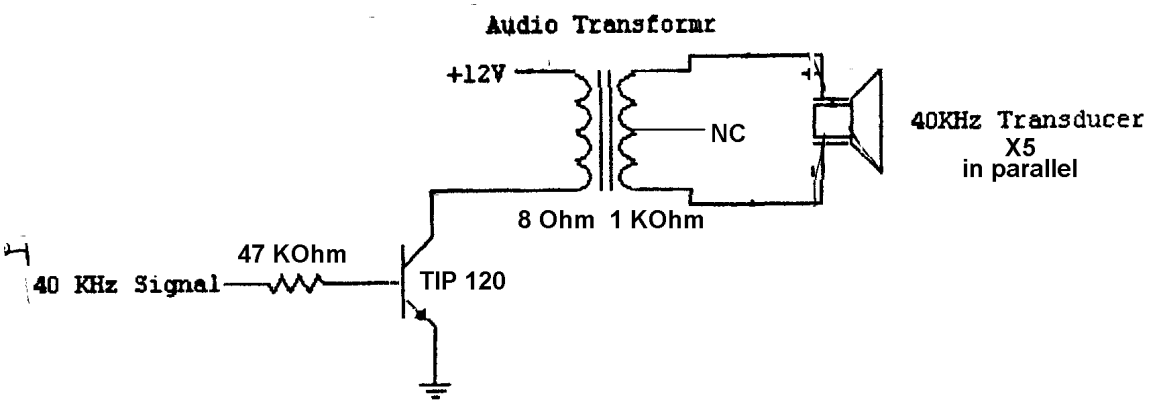
Transmitter-

The transmitter is controlled by a MHC6811 that does not have the ME11 expansion. As a result of this we had to generate the 40 kHz signal some other way. We decided to wire a decade counter up just like it is in the ME11. We needed three of these so we could send entirely separate signals to each of the beacons.



We wired the power pin of each chip to pins 23, 24, and 25 of the 68HC11 so that we could turn the beacons on individually by turning port D pins on and off. After the three 40 kHz signals left the chip, we sent the signal through line driving circuitry to reduce noise in the signal. The signal left the board through a 74'04 inverter and came to the beacon through a 74'14 Shmitt trigger. This was Scott's idea and helped to reduce the noise in the signal.

After the signal left the 68HC11 it traveled across a low gauge wire to the three beacons. The original design for the sonar transmitter had to be changed because of the large number of sonar transducers we had to use. The transducers were acting as a huge load and as a result, no voltage was getting through. To solve this problem we replaced the normal BJT with a TIP120 transistor. This sent way to much voltage so we had to tame it down by using a 47 Kohm resistor. We found that about a twenty volt signal is all that is necessary to inundate the room with sonar.

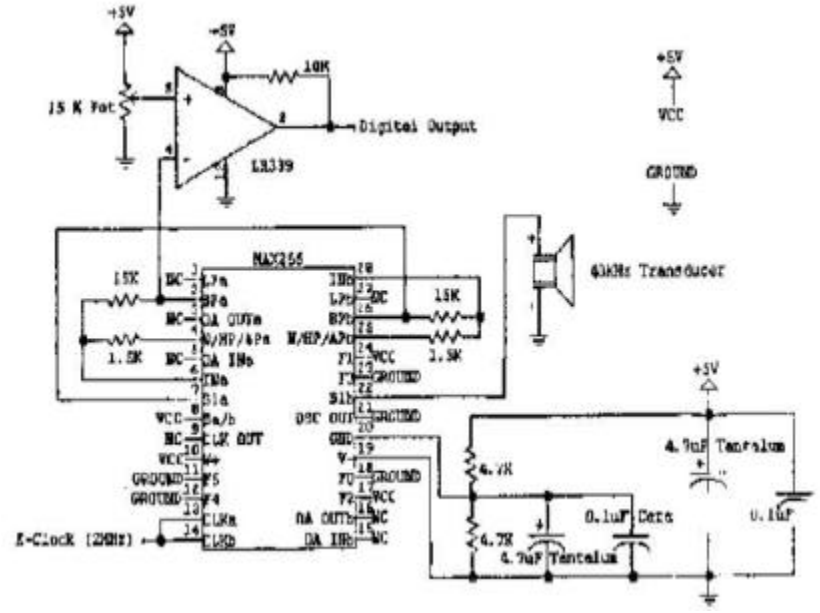


We had to turn this circuit on and off frequently because of the way it was acting. Once on the signal would start at about 15 volts peak to peak. Then the longer it was on the signal would grow continuously. The TIP120 would get very hot (hence letting more voltage through) and the transformer would burn. This is why we chose to hook the

power switch of each signal to the port D pins. This way the circuit never has to be on for too long of a period.

Receiver-

The receiver circuit we used is the same that has been used in the past except that we connected three transducers (in parallel) to the input. This was necessary to do because the receiving circuit will be in the middle of the beacons while they transmit their pulses.



In the future I would suggest that as many transducer as possible be used for the receiver. Three worked fairly well but still returned a few bad readings. I'm not sure how many transducers this circuit can drive but there is only one way to find out.

RF Sensor Suite-

The other sensor we used in order to determine positioning is the RF module. We used two separate frequencies for our project. One (315 MHz) was used for the controller and the other (418 MHz) was used to transmit the coordinates. The board we used for this was designed by Patrick O'Malley and is relatively simple. The actual chips were hard to

apply because they were surface mount chips. Scott Jantz helped us create these boards successfully.

The RF that we used for the controller was connected to pin 22 of the HC11. At the start of each sonar pulse we pulsed the RF as well. This worked very well and was a lot easier than actually sending serial data from the RF board. The digital pulse was received by the RF receiver which was connected to an expanded port from the ME11 board. The sonar was wired to the 2^3 bit and the RF was connected to the 2^7 bit. This way if the port read 128 we knew there was an RF pulse and if the port read 8 we knew there was a sonar pulse.

Behaviors

The main behaviors for this project were determining coordinates and sending those coordinates. The IR and bump switches are used for the obstacle avoidance and collision detection.

While the robot is stationary it receives five sets of coordinate. Then it takes the average of these five readings and transmits that data out of the serial port to the other robot. We simply displayed the coordinates on the computer screen but it should be easy to use these coordinates to interact with each other. We ran into problems because we used IC for the receiver program. For some reason we could not get the second robot to retrieve the three coordinates and use them. I would recommend that the next person that does this to use ICC11. This compiler does not use the serial port like IC does and is more accurate in the timing department. This code in IC is very timing critical.

Experiments

The following experiments were conducted on this project.

Bump sensors-

These are the analog readings for the A/D port and their respective locations

Location	**Analog Value**
Front Right	8
Right Center	17
Center	30
Left Center	46
Front Left	80
Rear	129

Using these values I can accurately determine which part of the platform met with the obstacle.

Infrared Sensors-

The test I ran on the IR sensors was designed to find the distance threshold of the detectors. I held a white notebook at various distances away from the detector and recorded the analog results.

Distance(in)	Analog Port
1	129
2	129
3	125
4	121
5	114
6	112
7	110
8	107
9	105
10	102
11	97
12	94
13	91
14	88

These values are acceptable for my purposes. With two IR detectors on either side of the front collision avoidance is relatively simple. When one detector is reading a high level you turn the robot the other way.

Beacon Triangulation-

The sonar beacons are extremely accurate (about 6 inches variable) up to about 26 feet. After that the readings are not as accurate. Anywhere inside the triangle the readings are always within 6 inches of the desired target.

Conclusion

In summary, we were able to create an accurate triangulation system but could not use these coordinates to have the robots interact. In the future, there are several improvements that could be made to this project. First, the use of IC was a bad decision for a number of reasons. Timing is difficult with IC and serial port transfers are also difficult. If ICC11 were used you could use the input capture feature of the HC11 to make the readings more accurate. Second, instead of using wires to connect the beacons one could use another set of RF boards. This would increase the range of the system and would clean up the mess of having to use long wires. Third, adding more transducers to the receiver might help with the accuracy of the readings.

This is a project that when attempted again should be worked on hard all semester. There is a lot of debugging in the timing of the software and also a lot of work to be done with the sonar circuitry. I had no idea how much work would be involved in accomplishing this RPS system. Now that we have the basics worked out hopefully someone can continue our work and have more time to spend on the communication between robots.

References

It should be noted that this project could not have been accomplished without assistance from the following sources:

- the Mekatronics website, for the manual on assembly of theTalrik platform
- Scott Jantz, Aamir Qaiyumi, Patrick O'Malley, and the other members of the MIL who were kind enough to contribute suggestions and assistance
- Dr. Arroyo and Dr. Shwartz for their words of wisdom and patients

Appendix A:
Vendor Information

Bump switches:
Mekatronics
(purchased from Scott Jantz)

$0.75 each

IR emitters/detectors:

LED emitters: Mekatronics
316 NW 17 th St., Suite A
Gainesville, FL 32603
(407) 672-6780
http://www.mekatronics.com
(purchased from Scott Jantz)
$0.75 each

Detectors: Sharp GPIU58Y via Mekatronics
(purchased from Scott Jantz)
$3.00 each

Sonar:

- audio output transformer: Radio Shack (local store)
part # 273-1380
$2.00 (approximately)

- MAX chip: Maxim Integrated Products
120 San Gabriel Dr.
Sunnydale, CA 94086
(408) 737-7600
http://www.maxim-ic.com
Part # 266
Free sample of two

Appendix B:
Code

Transmitter code-

```
*********************************************************
* JASON ISON AND CRAIG RUPPEL
* TRANSMITTER PROGRAM
*********************************************************


TOC3    EQU     $1A     ;OUTPUT COMPARE 3 REGISTER
TCTL1   EQU     $20     ;TIMER CONTROL REGISTER
TMSK1   EQU     $22     ;TIMER MASK1 REGISTER
TFLG1   EQU     $23     ;TIMER FLAG1 REGISTER
EEPROM  EQU     $B600
DDRD    EQU     $09
PORTD   EQU     $08
```

```
ADCTL      EQU         $30
TMSK2      EQU         $24
TFLG2      EQU         $25
REGS       EQU         $1000
BAUD    EQU     $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU     $102C  ; Serial Communication Control Register-1
SCCR2   EQU     $102D  ; Serial Communication Control Register-2
SCSR    EQU     $102E  ; Serial Communication Status Register
SCDR    EQU     $102F  ; Serial Communication Data Register

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


* MASKS
BIT7       EQU         %10000000

BIT6       EQU         %01000000
BIT5    EQU     %00100000
BIT4    EQU     %00010000


BIT10      EQU         %00000011
BIT1       EQU         %00000010
BIT0       EQU         %00000001
INV6       EQU         %10111111
INV5    EQU     %11011111


*
****************************************************
** DATA SECTION
****************************************************

      ORG     $00D9
      JMP     OC3_ISR ;OC3 INTERRUPT VECTOR
      ORG     $0100
Beacon  ZMB     1



****************************************************
*  Main
****************************************************

      ORG     $0110
Main        LDS         #$41
            LDX     #REGS
      LDAA    #$FF
      STAA    DDRD,X

            SEI

            JSR         InitSCI

* INITIALIZE OUTPUT COMPARE OC3 (Bit5:4=OM3:OL3  Hi=11,Lo=10)
      BCLR    TCTL1,X BIT5    ;OM3:OL3=11
      BSET    TCTL1,X BIT4    ;  for set to high

* ENABLE OC3 INTERRUPT
      BSET    TMSK1,X BIT5

* TURN ON INTERRUPT SYSTEM
      CLI
HERE    BRA     HERE


****************************************************
** TOC_3 pulse routine
****************************************************
OC3_ISR LDX     #REGS

* INTERRUPT FROM OC3?
    BRCLR   TFLG1,X BIT5 mid ;IGNORE ILLEGAL

* CLEAR OC3 FLAG
      BCLR    TFLG1,X INV5



      LDAA    Beacon
            CMPA        #5
            BEQ         First
            CMPA        #10
            BEQ         Second
            CMPA        #15
```

```
            BEQ         Third
            CMPA        #20
            BEQ         Period
            BRA         Wait
mid         RTI

First       INC         Beacon
           LDAA    #$04
           STAA    PORTD,X
            ldy         #$2000
wait1       dey
            bne         wait1
           LDAA        #$00
           STAA    PORTD,X

            ldy         #$0f00
wait7       dey
            bne         wait7


            LDAA    #$08
           STAA    PORTD,X

            ldy         #$2000
wait2       dey
            bne         wait2
           LDAA        #$00
           STAA    PORTD,X


            BRA         RT_OC3

Wait        INC         Beacon
            BRA         RT_OC3



Second
            INC         Beacon

            LDAA    #$04
             STAA    PORTD,X

            ldy         #$2000
wait3       dey
            bne         wait3
            LDAA        #$00
             STAA    PORTD,X

            ldy         #$0f00
wait8       dey
            bne         wait8


            LDAA    #$10
              STAA    PORTD,X

            ldy         #$2000
wait4       dey
            bne         wait4
            LDAA        #$00
                STAA    PORTD,X


            BRA         RT_OC3
Period      LDAA        #0
            STAA        Beacon
            RTI


Third
            INC         Beacon
             LDAA    #$04
             STAA    PORTD,X

            ldy         #$2000
wait5       dey
            bne         wait5
            LDAA        #$00
            STAA    PORTD,X

            ldy         #$0f00
wait9       dey
            bne         wait9


            LDAA    #$20
            STAA    PORTD,X
```

```
        ldy     #$2000
wait6   dey
        bne     wait6
        LDAA    #$00
        STAA    PORTD,X
        BRA     RT_OC3
RT_OC3  RTI
```

Receiver Code –

This includes all of the necessary serial routines that can be found in serial.c

```
/*
  init_serial

  Initializes the SCI port on the 68HC11 to operate at 9600 baud.  This
  function must be called at the beginning of your program if you wish
  to use any of the functions in this library.

  Example:
    init_serial();
*/
void init_serial()
{
  bit_clear(0x1028, 0x20);
  poke(0x102B, 0x32);
  poke(0x102D, 0x0C);
}


/*
  get_char

  Waits for a character to be received by the serial port, then returns
  its ASCII value.

  Examples:
    x = get_char();
    if (get_char() == 'F') fd(0);
    get_char();
*/
int get_char()
{
  int test = 0;

  while (test == 0) {
    test = peek(0x102E);
    test = test & 0x20;
  }

  return(peek(0x102F) & 0x7F);
}


/*
  put_char

  Writes an ASCII character to the serial port.

  Examples:
    put_char(65);
    put_char('A');
*/
void put_char(int outchar)
{
  int test = 0;

  while (test == 0) {
    test = peek(0x102E);
    test = test & 0x80;
  }
```

```
  poke(0x102F, outchar);
}


/*
  write

  Writes a string of text to the serial port.  The only control character
  supported is the newline character (\n).

  Examples:
    write("Hello, world!\n");
    write("one\ntwo\nthree\n");
*/
void write(char string[])
{
  int index = 0;

  while(string[index] != 0) {
    if (string[index] == '\' && string[index+1] == 'n') {
      put_char(13);
      put_char(10);
      index += 2;
    }
    else {
      put_char(string[index]);
      index++;
    }
  }
}


/*
  put_int

  Writes an integer to the serial port.
*/
void put_int(int number)
{
  int count;
  int first_digit = 0;

  if (number < 0) {
    put_char('-');
    number *= -1;
  }

  count = 0;
  while (number > 9999) {
    number -= 10000;
    count++;
  }

  if (count > 0) {
    put_char(count + 48);
    count = 0;
    first_digit = 1;
  }

  count = 0;
  while (number > 999) {
    number -= 1000;
    count++;
  }

  if (count > 0 || first_digit == 1) {
    put_char(count + 48);
    count = 0;
    first_digit = 1;
  }
```

```
  count = 0;
  while (number > 99) {
    number -= 100;
    count++;
  }

  if (count > 0 || first_digit == 1) {
    put_char(count + 48);
    count = 0;
    first_digit = 1;
  }

  count = 0;
  while (number > 9) {
    number -= 10;
    count++;
  }

  if (count > 0 || first_digit == 1) {
    put_char(count + 48);
    count = 0;
  }

  put_char(number + 48);
}

/*
  write_int

  Print an integer to the serial port prefixed by a space and followed
  by a newline.

  Examples:
    write_int(x);
    write_int(analog(7));
*/
void write_int(int number)
{
  put_char(' ');
  put_int(number);
  put_char(13);
  put_char(10);
}


/*
   print

   Limited version of printf().  The only control characters allowed
   are newline (\n) and integer (%d).  One (no more, no less) integer
   can be inserted in a string.

   Examples:
     print("The variable x is %d.\n", x);
     print("Sensor 1 is reading %d and", analog(0));
     print("sensor 2 is reading %d.\n", analog(1));
*/
void print(char string[], int number)
{
  int index = 0;

  while(string[index] != 0) {
    if (string[index] == '\' && string[index+1] == 'n') {
      put_char(13);
      put_char(10);
      index += 2;
    }
    else if (string[index] == '%' && string[index+1] == 'd') {
      put_int(number);
      index += 2;
    }
```

```
    else {
      put_char(string[index]);
      index++;
    }
  }
}



int counter, first, second, third, firsttime;
int secondtime, thirdtime, timer;
int sonar, RF;
int myarray[15];
int i = 0;
void main()
{

init_serial();
start();
}

void start()
{

timer=0;
RF = 128;
sonar = 8;

while(peek(0x6000) != 128)
;
{ while(1)
  {
   timer=0;

        while((peek(0x6000) != 128) && (peek(0x6000) != 136))
           {
            timer +=1;
            }

    if(timer>260)
       { tof();
        }

 }
}}



void tof()
{
counter=0;


while(peek(0x6000) != 8)
       counter +=1;

       myarray[i]=counter;
       counter = 0;

while(peek(0x6000) != 128)
;
while(peek(0x6000) != 8)
       counter +=1;

       myarray[i+1]=counter;

        counter = 0;

while(peek(0x6000) != 128)
;
```

```
while(peek(0x6000) != 8)
        counter +=1;


        myarray[i+2]=counter;
        if(i == 12)
         {
          i = 0;
          first = myarray[0] + myarray[3] + myarray[6] + myarray[9] + myarray[12];
          second = myarray[1] + myarray[4] + myarray[7] + myarray[10] + myarray[13];
          third = myarray[2] + myarray[5] + myarray[8] + myarray[11] + myarray[14];
          first = ((first / 5) - 93) * 5;
          second = ((second / 5) - 93) * 5;
           third = ((third / 5) - 93) * 5;
           print("the first beacon is %d (feet*10) away\n",first);
          print("the second beacon is %d (feet*10) away\n",second);
           print("the third beacon is %d (feet*10) away\n",third);
        }

        else
           i+=3;

  return;
}
```