Microcomputer Communications Project

Assessment 2003/2004

Syedur Rahman

TABLE OF CONTENTS

INTRODUCTION

After going to the introductory MCP lecture, I have to admit I was a bit intimidated (which is usually very hard for me to admit), by this assignment, which seemed rather complicated and tedious. All the talk about infra-red sensors, servo-motors and other things I haven't really handled in computer science courses before just made it seem that way. Later I realised all I had to do was stick to the basic principles taught in the previous hardware courses and not be that concerned with the other aspects of the system, especially those that brought back horrid memories of A-level physics.

The aim of this project was to integrate a Z80 processor system with a infra-red sensor attached to a rotating servo-motor to build some kind of radar (I realise it is not really a radar since that would require radio and not infra-red waves). Fortunately, this design would not be so low level, since this Z80 processor already came with its own data and address buses, RAM chips, external ports and an LCD display with a keypad.

There are three modes of function the completed system was supposed to be able to perform. First in radar mode, the system is supposed to sweep through a predefined angle and then display objects in the vicinity by showing their positions on the LCD display (as polar/cartesian co-ordinates) and also on the oscilloscope. There would also be a speaker to produce sound effects indicating whether an object is there when the sensor is at any particular angle.

In the tracker mode, the sensor is supposed to follow an object that is slowly being moved around and once again a sound effect indicates whether the object is still in the sensor's line of sight. Once again, the object's position would have to be shown on the LCD display and oscilloscope.

In the profiler mode, it sweeps through the predefined angle as in the radar mode, only after its done sweeping, it displays some kind of image on the oscilloscope which would show the shape of the object placed near the sensor. While the object is being scanned, the speaker would make some kind of sound indicative of the shape of the object.

My partner and I decided to stick to a very simple design as we did for CTS (even though that did not get us too many marks). Our main objective for this project was making a system that meets the minimum requirements with particular emphasis on simplicity, cost-effectiveness and of course minimum effort.

OVERVIEW OF HARDWARE/SOFTWARE

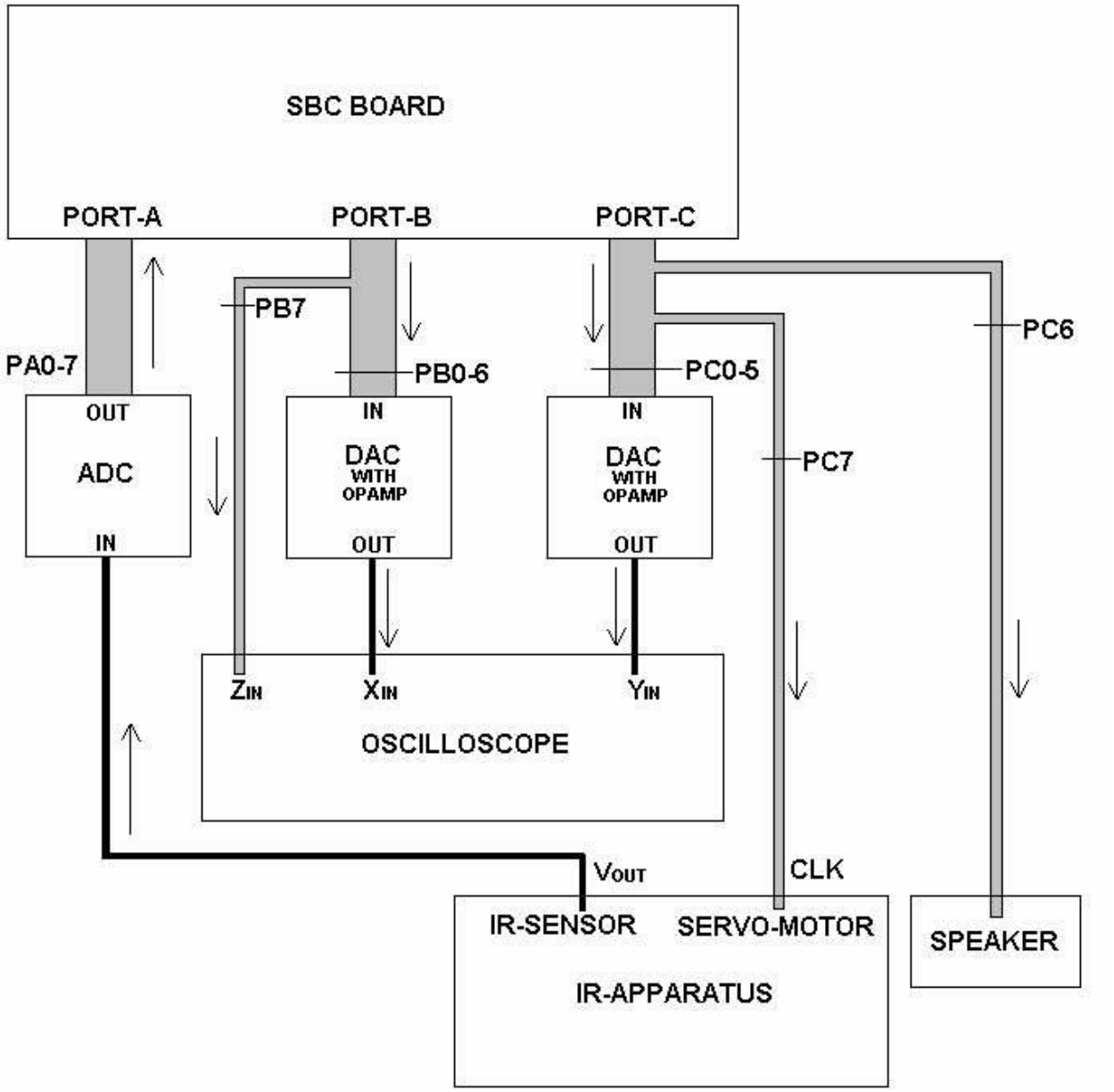
Fortunately, we were given the SBC board this time to construct our hardware around. This made this assignment quite ridiculously easy (to design at least, the implementation however was a nightmare as in any project). First of all the board, the RAM chips, the LCD display and the keypad are already wired into the Z80, so there was not much time spent in cutting out little bits of wire, sticking them to pins and later spending hours trying to figure out why our system was not working before discovering it was because of two wires switched around or a faulty breadboard (I was obviously very frustrated with that through out the CTS project). The fact that the Z80 address and data pins are in absolutely no meaningful order did not make things any easier in the previous assignment. The SBC board also comes with three 8-bit ports which make it very convenient to do I/O operations since these also act as latches. A program can also be downloaded from a linux terminal directly into the RAM, which makes testing a lot simpler than having to burn a ROM chip for every little change in code.

The basic external hardware required were:

1. Two digital-to-analogue converters (DACs) and opamps to plot points on the oscilloscope using digital data from the SBC board
2. An analogue-to-digital converter to convert the analogue reading from the infra-red sensor into digital data that can be understood and manipulated by the processor.

The ports on the SBC made it very easy to do I/O operations without involving the data bus or any address lines at all. This made the hardware construction very simple, however it also meant software required to run the system ended up being very complicated since we were using the three serial ports to drive six I/O devices (the two DACs, the ADC, the servo-motor, the speaker and the oscilloscope Z input).

Block Diagram of Overall System

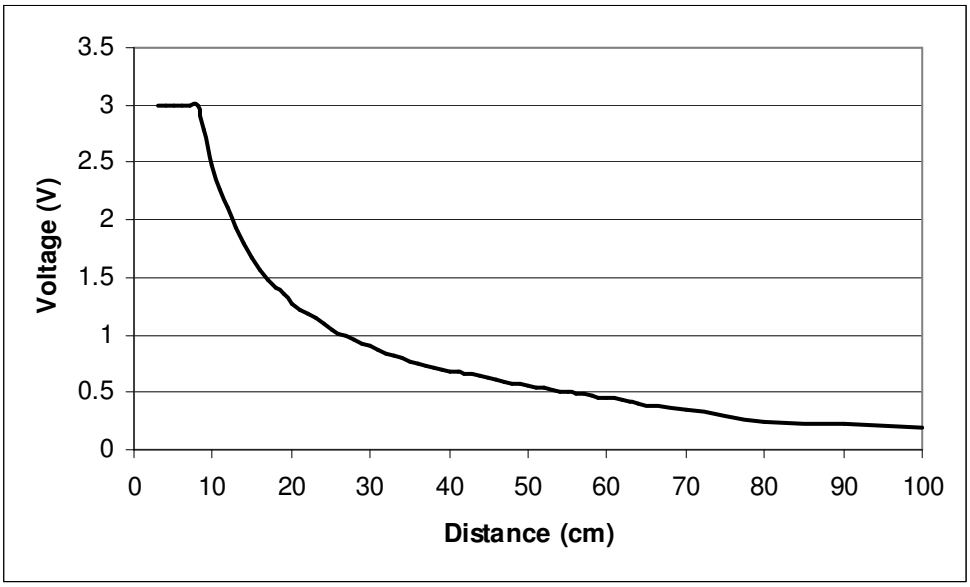


The block diagram above describes how the system works with arrows indicating data flow

Infra-red sensor

The infra-red sensor generates an analogue voltage depending on the distance between any obstacle and the sensor. This analogue reading is fed in to the input of the ADC with the appropriate reference voltage to produce a digital output which is later converted by software into distances using a reading-to-distance look up table stored in memory.

A graph of voltage against distance looks like this



Had this been a simple straight line we would have been able to use a simple equation for calculating distances from voltages (converted to digital values from the ADC). We had decided that our system would have a range between 10 and 60cm. As you can see at 10cm the voltage reading is about 2.1V. So the reference voltage we put in to the ADC was about 2.2V (using a potential divider, please see circuit diagrams for details).

Sensor Calibration

A simple program was written which took readings off the ADC and then displayed the digital reading received in hex on the LCD monitor. Several readings of ADC values were taken at different points between 10 and 60cm (which is the operational range of this system) at every 1cm. These values needed to be averaged (since the ADC values were inconsistent at times) and then a reading-to-distance table was constructed which mapped each binary value from the ADC to a distance value. Please see Testing/Evaluation for calibration tables.

In between binary values, which were not read for any specific distances (at each 1cm interval) simply mapped to the closest 1cm discrete value on the table.

Servo-Motor

The servo-motor is controlled by giving it a pulse of period 5ms, i.e. a frequency of 200Hz and the peak time of this pulse determines the angle it will rotate to. The Z80 has two internal clocks/timers, which came in very handy for this purpose. The timers can be initialised with reload values, and then the timer starts counting down beginning from the reload value. Once it has counted down to zero, it generates an interrupt (distinct for each timer) and then it starts counting down from the reload value again.

The interrupt routine was made such that it flips a bit on Port B every time the timer counts down to zero. After much experimentation while monitoring the output of the port using the oscilloscope we found that the timer takes 5ms to countdown with a reload value of 0600h. Controlling the motor was simple from then on, we could start with a certain peak time value with the reload register with the clock output on the port set to a 1 and then after its counted down to 0 (this representing a high on the output), we would flip the bit and at the same time set

the reload value to 0600h-peak (this representing a low on the output). So, we would have a wave of period 5ms and the peak period is controlled by the initial value put into the reload registers.

After making a few measurements, I discovered that there was a very linear relationship between angle and reload value. I took a few readings and plotted a graph and then using linear regression techniques I found a simple equation involving reload values and angle

Reload = 2.84444*angle + 208

Once again a table was constructed that mapped angles to reload values so that the program would put in the suitable reload value for each angle when it requires to move the sensor to that angle.

The Oscilloscope

The X-input of the oscilloscope was connected to the lower 7 bits of port B (via an digital to analogue converter and opamp) and the Y-input of the oscilloscope was connected to the lower 6 bits of port C (via an digital to analogue converter and opamp). I set the range of the system to be between 10 and 60cm. For a full sweep, this would require a semi-circle of radius 60 cm and let us assume we used one pixel per cm. Therefore the y-axis would need 60 divisions and the x-axis would need 120 division, so 6 bits (64 divisions) and 7 bits (128 divisions) were appropriate for x and y respectively. The Z-input was controlled by bit7 of port B. If the bit was set to a high (5V), this turns the beam off. This was very useful, since if we left the beam on while plotting between two points, there would be quite a few unwanted dots plotted on the screen in the mean time. So, this bit was set to 0 to turn the beam on, or set to 1 to turn it off.

For the x and y co-ordinates, polar co-ordinates (angle of sensor and distance read) had to be converted into cartesian co-ordinates. For this please see details on the get_coords module in the software modules section as well as the section on Design/Implementation

The Speaker

The speaker is controlled in a very similar way as the servo motor is. The processor has another timer, which can generate interrupts at a regular interval depending on the value put into its high and low reload registers. At each interval the speaker bit (bit6 on portC) is flipped giving a sound of a frequency dependent on the reload registers. In all cases, we are supposed to make a sound depending on the distance between the sensor and the object. This can be done by simply putting the ADC data read from the sensor (from portA) into the reload low register of timer1. I found that the sound is very different for objects at different distances, which is exactly what was required.

Software Overview

I decided to do a sweep of about 90 degrees, starting at 45 degrees and ending at 135 degrees at every one degree for the profiler and radar modes. However for the tracker mode, I decided to leave the range of angles between 0 and 180, since tracking something in a smaller range of angles just seems pointless.

The following shows a list of all modules in the program along with brief descriptions of what they do. I would have preferred to draw a diagram showing the relationships between modules but unfortunately this would take too much space and be too complicated to be legible at all.

Procedure (parameters)	**Description**	**Modules Called (parameters)**
main_menu	Displays the main menu options. Checks value of mode, then runs appropriate subroutine (1=radar, 2=tracker, 3=profiler)	clear display_menu radar tracker profiler
radar	Performs the radar mode	init_angle45 enable_servo_timer servo_delay_full disable_servo_timer rotate_to_angle(angle) get_distance clear display_radar_mode display_angle(angle) plot_outline (mode) get_coords (angle, distance) plot_point (x_coord, y_coord) beam_on beam_off display_angle_distance (angle, distance) make_sound
tracker	Performs the tracker mode	init_angle0 servo_delay_full rotate_to_angle (angle) get_distance clear display_tracker_seeking display_tracker_locked display_angle_distance (angle, distance) plot_outline(mode) get_coords (angle, distance) plot_point(x_coord, y_coord) beam_on beam_off make_sound show_profile
profiler	Performs the profiler mode	init_angle45 enable_servo_timer

		servo_delay_full disable_servo_timer rotate_to_angle(angle) get_distance clear display_radar_mode display_angle_distance (angle,distance) plot_outline(mode) get_coords (angle, distance) plot_point (x_coord, y_coord) beam_on beam_off make_sound show_profile
show_profile	Uses data from profiler (which enters values into the x_coords and y_coords table) to plot points displaying the profile of the object scanned earlier	plot_outline (mode) plot_point (x_coord, y_coord)
enable_servo_timer	Enables timer0 of the processor	
disable_servo_timer	Disables timer0 of the processor	
servo_timer_toggle	When timer0 generates an interrupt, the output to the servo motor (PortC B7) is toggled and it is set such that the next interrupt occurs at 5ms-previous time (i.e. giving a wave of period 5ms)	
enable_speaker_timer	Enables timer1 of the processor	
disable_speaker_timer	Disables timer1 of the processor	
speaker_timer_toggle	When timer1 generates an interrupt, the output to the speaker (PortC B6) is toggled, giving a square wave	
rotate_to_angle	Sets timer0's reload registers using appropriate values from the reload_high and reload_low tables corresponding to the angle given. Runs timer0 for a short time (enough for a maximum 5 degree rotation), allowing the servo to rotate to that angle and then disables timer0.	enable_servo_timer disable_servo_timer
get_coords (angle, radius)	Gets polar co-ordinates from angle and distance and converts them into cartesian co-ordinates and stores the results in global variables x_coord and y_coord to be plotted on the oscilloscope later Note: x_coord is actually the oscilloscope x-coordinate (with 64 as the center of our 128bit x-axis display)	

plot_point (x_coord, y_coord)	Puts x_coord into the lower 7 bits of PortB (oscilloscope x-input) and y_coord into the lower 6 bits of PortC (oscilloscope y-input). Turns the oscilloscope beam on for a short while to plot the point on the oscilloscope and then tunrs the beam off	beam_on beam_off
get_distance	Reads value from ADC and using the distance table converts the ADC value into distance from sensor and stores this value in the global variable distance. (Four readings of distances are taken and only the average is returned)	
display_angle_distanc e (angle, distance)	Displays current angle, distance, x-coordinate and y-coordinate on LCD screen (uses display_bin_to_dec to convert these numbers)	display_bin_to_dec (angle) display_bin_to_dec (distance) display_bin_to_dec (x_coord) display_bin_to_dec (y_coord) delay_keypad
display_bin_to_dec (number)	Converts the variable number, which is in binary into a 3 digit decimal number and displays each of these digits on the LCD	display_character (digit) delay_keypad
display_character (register a)	Uses the ASCII code stored in register A to display the character on the LCD and then waits for a very short time to allow the character to appear on it	delay_keypad
increase_angle3	Increments global variable angle by 3	
decrease_angle3	Decrements global variable angle by 3	
increase_angle2	Increments global variable angle by 2	
decrease_angle2	Decrements global variable angle by 2	
make_sound	Using the ADC value as the frequency, it enables the speaker for a short period indicating distance of the object (since it determines the frequency of the sound produced)	enable_speaker_timer delay_sound disable_speaker_timer
keypad_int	When the keypad is pressed, global variable mode is updated according to key pressed (if invalid mode i.e. not between 1 and 3, then mode is set to 0, which causes all mode modules to return to the main menu)	
clear	Sends command to LCD to clear the scren	
beam_on	Turns oscilloscope beam on by setting the Z-input to the oscilloscope (Port B Bit7) to a logic 0 (0V)	
beam_off	Turns oscilloscope beam on by setting	

	the Z-input to the oscilloscope (Port B Bit7) to a logic 1 (5V)	
plot_outline	Plots an outline on the oscilloscope. It shows all the extreme points plot-able by the system on the oscilloscope and it shows a small line of radius 10 pixels at an angle corresponding to global variable angle, indicating the direction the sensor is pointing For radar and profiler modes it displays two lines of radius 64, one at 44 degrees and one at 136 degrees indicating the sweep-able region	beam_on beam_off get_coords (angle, radius) plot_point
init_vector_table	Initialises the vector table, which the processor uses to call interrupt servicing routines when the keypad or the clocks generate an interrupt	
init_ports	Sets it such that PortA is used for input (ADC), PortB is used for output (Z and X inputs for the oscilloscope) and PortC is used for output (speaker, servo motor and oscilloscope-Y inputs)	
init_angle0	Sets angle to 0 degrees, loads appropriate reload values into timer0, enables it, gives the servo ample time (using delay_servo_full) to rotate to 0 degrees and then disables timer0	enable_servo_timer servo_delay_full disable_servo_timer
init_angle44	Sets angle to 44 degrees, loads appropriate reload values into timer0, enables it, gives the servo ample time (using delay_servo_full) to rotate to 44 degrees and then disables timer0	enable_servo_timer servo_delay_full disable_servo_timer
delay_servo_full	Produces a just about long enough delay for the servo to rotate a large angle (> 90 degrees)	
delay_servo_short	Produces a just about long enough delay for the servo to rotate a large angle (< 5 degrees)	
delay_keypad	Produces a just about long enough delay required for a character to appear on the LCD	
display_menu	Displays the main menu options on LCD	
display_radar_mode	Displays “Radar Mode” on LCD	
display_profiler_mode	Displays “Profiler Mode” on LCD	
display_tracker_seeking	Displays “Tracker-Seeking” on LCD	
display_tracker_locked	Displays “Tracker-Locked” on LCD	

display_rotating	Displays “Rotating Back” on LCD	
display_object	Displays “Object Detected” on LCD	

Global Variables

Since this a low level language, all variables used were global. They were declared as having specific memory locations at the start of the program in such locations that they did not overlap with any part of the program or loop up tables in memory.

Variable Name	Purpose	Modules used in
angle	Stores the current angle at which the sensor is positioned in all modes or the angle of the point being plotted on the screen.	radar tracker profiler increase_angle3 decrease_angle3 increase_angle decrease_angle rotate_to_angle get_coords display_angle_distance
distance	Stores the current distance of the object being detected in front of the sensor	get_coords display_angle_distance radar tracker profiler
objectdistance	Stores the distance between the centre of the last object detected and the sensor in radar mode	radar
objectangle	Stores the angle at which the centre of the last object detected was in radar mode.	radar
middistance	Stores the distance at which the object was previously in tracker mode	tracker
midangle	Stores the angle at which the centre of the object being tracked was last detected in tracker mode	tracker
register d	Temporarily holds the value of reload register high for the servo timer (timer 0)	clock_timer_toggle rotate_to_angle
register e	Temporarily holds the value of reload register low for the servo timer (timer 0)	clock_timer_toggle rotate_to_angle
radius	While converting polar to cartesian co-ordinates through the get_coord procedure, radius is used the parameter	radar tracker profiler get_coords
x_coord	Stores the x-coordinated of last point whose co-ordinates were converted from polar to cartesian using get_coords	get_coords radar tracker profiler display_angle_distance plot_point

y_coord	Stores the x-coordinated of last point whose co-ordinates were converted from polar to cartesian using get_coords	get_coords radar tracker profiler display_angle_distance plot_point
mode	Stores the mode that the program is now running (1, 2 or 3). When any key is pressed on the keypad the value of mode is changed accordingly.	menu tracker profiler radar

Basic Overview of Each Mode of Operation

Radar Mode

The radar module works by first setting the angle to 45 degrees and allowing the motor enough time to rotate to that angle. From then on it takes distance readings at each angle (incrementing and rotating to the next angle) and then uses them to find out whether there is an object in the vicinity. An object is often detected at two or more angles, however the actually angular position of the object is one where the distance reading is the lowest but in radar mode we'd like an object to show up only once on the scope.

So, when something is detected (i.e. distance < 60 cm), it is treated as the start of the object (ObjectAngle and ObjectDistance are set to current angle and distance), and for the next subsequent angles the object's angle and distance are updated in case the distance is less than what it was at the earlier angle (i.e. if distance < ObjectDistance reset ObjectDistance and ObjectAngle to current ones). After nothing is detected once again, the distance and angle of the object are displayed on the LCD and oscilloscope (using ObjectAngle and ObjectDistance which are subsequently set to 0 and 64 respectively, to allow the detection of a new object) and a sound is made on the speaker. After reaching 135 degrees, the radar procedure simply starts again at 45 degrees and does the sweep again. However, it keeps checking if the mode has been changed (by the keypad_int procedure which services interrupts from the keypad updating the mode), in which case it returns to the main menu.

Module Radar

```
      Repeat
            Call init_angle45    # set servo to start sweep at 45degrees
            Call enable_servo_timer
            Call servo_delay_full
            Call disable_servo_timer
            ObjectDistance = 64  # initialising last object detected at infinity


            Repeat
                  Call enable_servo_timer
                  Call delay_servo
                  Call get_distance    # get distance and put it in (distance)
                  Call display_angle_distance # lcd displays angle,
distance etc.
                  Call plot_outline    # show sweep-able region
```

```
            If distance < ObjectDistance then
                  #  new centre of object found
                  ObjectDistance = distance
                  ObjectAngle = angle
            End if

            If distance > 60 and ObjectDistance < 60 then
                  # if there is a blank space means end of object
                  # show details on object detected
                  Tempangle = angle
                  distance = ObjectDistance
                  angle = ObjectAngle
                  call display_angle_distance
                  call get_coords(angle, distance)
                  call plot_point(x_coord, y_coord)
                  call beam_on
                  call make_sound
                  # reset object details to infinity
                  ObjectAngle = 0
                  ObjectDistance = 64
                  Angle = Tempangle
            End if

            Call increase_angle # angle = angle + 1
      Until angle > 135 or mode <> 1
Until mode <> 1
```

End Radar

Profiler Mode

The profiler mode works very similarly to the radar mode, starting a sweep by rotating first to 45 degrees and then incrementing the angle by two while taking distance measurements on each angle. However, at each angle it also shows the angle, distance, x and y coordinates on the LCD display and plots these co-ordinates on the oscilloscope. It stores the x-coordinate and y-coordinate at each particular angle in two separate arrays using the angle as the offset. It also makes a sound at each angle at a frequency dependent on the distance reading. After its done sweeping to 135 degrees, it then shows the profile of the object scanned. This is done by the module show_profile, which simply takes out x and y coordinates out of the two arrays, starting the offset at 45 and plots each point, each time increasing the offset by 1 until it reaches 135 and then it starts again from 45 and keeps looping around, effectively showing the "profile" of the object on the oscilloscope screen.

Module Profiler

```
Call init_angle45    # set servo to start sweep at 45degrees
Call enable_servo_timer
Call servo_delay_full
Call disable_servo_timer

Repeat
      Call rotate_to_angle
      Call get_distance  # calculate distance and put it in variable
distance
      Call display_angle_distance
      Call plot_outline
```

```
      if distance < 60 then    # stores oscilloscope co-ordinates in
      arrays if distance is valid
            Call get_coords(angle, distance)
            Call plot_point(x_coord, y_coord)
            Call make_sound
            x_coords(angle) = x_coord
            y_coords(angle) = y_coord
      else
            x_coords(angle) = 0
            y_coords(angle) = 0
      End if

      Call increase_angle (angle = angle + 1)
Until angle > 135 or mode >< 3

Repeat      # keep showing profile until mode changed via keypress
      Call show_profile
Until mode >< 3
```

End Radar

Module show_profile

```
Call plot_outline  # display sweep-able region on scope

angle = 45   # start showing points from that at 45d

Repeat
# show all points stored in x_coords and y_coords unless they're blanks
      x_coord = x_coords(angle)
      y_coord = y_coords(angle)
      If x_coord > 0 and y_coord > 0 then
            Call plot_point(x_coord, y_coord)
      End if
      Call increase_angle (angle = angle + 1)
Until angle > 135

Call plot_outline
```

End show_profile

Tracker Mode

The tracker mode first uses a similar code as the radar (which I prefer to call a seeker) to sweep starting at 0 degrees and stops when it finds an object at 30cm (that is the operational range I set for the tracker).

Then it sets MidDistance and MidAngle as current distance and angle respectively. It then adds three degrees to the angle, rotates to the right and takes a new distance reading, if this distance is less than MidDistance (indicating the object has moved to the right) it updates MidDistance and MidAngle with current values. Otherwise, it subtracts three degrees from MidAngle, rotates to the left and takes a new distance reading, and then updates MidDistance and MidAngle if this distance reading is less than MidDistance (i.e. object has moved to the left). Another little enhancement to the tracker is that once it sees that the object is moved to the left, the next time it first checks to the left and the same thing when the object has moved to the right. This is very

sensible since we can assume the object would move in the same direction more often than change direction.

It keeps looping around like this until it sees that the object is no longer anywhere in that 6 degree vicinity, i.e. MidDistance > 60. In this case, it loops back to the seeker part and finds the object again before tracking it. It makes a sound if the object is within 30cm, or else it just keeps quiet.

Module tracker

```
Repeat

        ###### Seeker Part locates an object first within 30cm
        Repeat

                Call init_angle0 (angle = 0)
                Call servo_delay_full

                Repeat
                        Call rotate_to_angle
                        Call get_distance
                        Call display_angle_distance
                        Call plot_outline
                        call get_coords(angle, distance)
                        call plot_point(x_coord, y_coord)
                        Call increase_angle3 (angle = angle + 3)
                Until angle > 180 or mode <> 2 or distance < 30

        Until mode <> 2 or distance < 30
        ###### Seeker ends when an object is located at less than 30cm

        Call decrease_angle3 (angle = angle - 3)
        MidAngle = angle      # set current angle, distance as mid values
        MidDistance = distance

        Repeat
                call get_distance
                angle = MidAngle
                Middistance = distance
                Call display_angle_distance
                If distance < 30 then Call make_sound
                Call plot_outline
                call get_coords(angle, distance)
                call plot_point(x_coord, y_coord)

                If angle < 178 then
                        call increase_angle3 #checking right
                        call rotate_to_angle(angle)
                        call get_distance
                End if

                If distance < MidDistance then
                        MidAngle = angle #object moved right
                Else
                        If angle > 2 then
                                Repeat
                                        angle = MidAngle #checking left
                                        decrease_angle3
                                        call rotate_to_angle
```

```
                                        call get_distance
                                        If distance < MidDistance then
                                               MidAngle = angle #moved left
                                               Middistance = distance
                                               Call display_angle_distance
                                               If distance < 30 then Call make_sound
                                               Call plot_outline
                                               call get_coords(angle, distance)
                                               call plot_point(x_coord, y_coord)
                                        Else
                                               angle = MidAngle #reset angle
                                               rotate_to_angle
                                        End if
                                Until distance > MidDistance #keep checking left until
                        object no longer in left
                               End if
                        End if

                Until middistance > 60 or mode <> 2

        Until mode <> 2
```

End tracker

PseudoCode for Other Important Modules

Module rotate_to_angle (angle)

```
# sets appropriate reload values into timer 0 for angle and then enables timer 0 for long
enough for a short rotation (1-3 degrees)

        timer0_reload_high = reload_high_table(angle)
        timer0_reload_low = reload_low_table(angle)

        call enable_timer0
        call servo_delay
        call disable_timer0
```

End rotate_to_angle

Module make_sound

```
# outputs a wave to speaker to make a sound with frequency dependent on distance/ADC value

        Temp = ADC_Data
        Temp = Temp OR 0x10
        # Or with 0x10 otherwise too small a period would cause too frequent interrupts and
cause the program to behave abnormally
        timer1_reload_high = 0
        timer1_reload_low = Temp

        call enable_speaker_timer
        call delay_sound
        call disable_speaker_timer
```

End make_sound

Module display_bin_to_dec (number)

```
        # displays a binary number put into number as a 3 digit decimal number on the lcd

        H = 0
        # to find hundreds digit keeps subtracting hundred from number until number < 100
        Repeat
                number = number - 100
                H = H+1
        Until number < 100
        H = H-1
        If H > 0 then
                A = H + 0x30
                # 0x30 is 0 in ASCII, 1 is 0x31 and so on.
                Output_to_LCD(A)
                Call delay_lcd
        End if

        H = 0
        # to find tens digit keeps subtracting ten from number until number < 10
        Repeat
                number = number - 10
                H = H+1
        Until number < 10
        H = H-1
        A = H + 0x30
        Output_to_LCD(A)
        Call delay_lcd

        # whatever is left is the units digit
        H = number
        A = H + 0x30
        Output_to_LCD(A)
        Call delay_lcd
```

End display_hex_to_decimal

Module display_angle_distance (angle, distance)

```
        # Displays current angle, distance, x-coordinate and y-coordinate on LCD screen (uses
display_bin_to_dec to convert these numbers)

        Call display_hex_to_decimal(angle)
        Output_to_LCD("o ")

        If distance <= 60 then
                Call display_hex_to_decimal(distance)
                Output_to_LCD("cm ")

                Call get_coords

                # x_coord and y_coord hold oscilloscope co-ordinates, the x_coord needs to be
subtracted/added from 64 to give cartesian co-ordinates

                If x_coord <= 64 then
                        Output_to_LCD("-")
                        x_coord = 64 - x_coord
                else
                        x_coord = x_coord - 64
                end if
                Call display_hex_to_number(x_coord)
```

```
            Output_to_LCD(“,”)

            Call display_hex_to_number(y_coord)

      End if
```

End display_angle_distance

Module get_coords(angle, radius)

```
      # converts polar co-ordinates to oscilloscope co-ordinates
      x_coord = cosine_table(angle) x radius

      if angle < 90 then
            x_coord = 64 - x_coord
      else
            x_coord = 64 + x_coord
      end if

      y_coord = sine_table(angle) x radius
```

End get_coords

Module get_distance

```
      # reads in IR sensor from ADC and converts it to distance. Takes 4 readings and does
average for greater accuracy
      ADCSum = 0
      For I = 1 to 4
            ADCSum = ADCSum + Input_from_ADC
      Next

      distance = distance_table(ADCSum / 4)
```

End get_distance

DESIGN/IMPLEMENTATION DETAILS AND ALTERNATIVES

As I previously discussed briefly in the introduction, our main objective was to keep the hardware as simple as possible, not only because it is simply easier for us to put together but it also meant that the entire system would not break because of a simple mistake in wiring.

Input/Output

We decided to use the ports to perform all I/O operations. The good thing was that all these ports have dedicated data buses (or so to speak), which would mean they could be directly connected to each of the devices they are driving. So, there would be no need of lots of wires coming out of the data bus going into each of the devices. If we had used a discrete I/O scheme, we would also need a decoder to enable each particular device at a time (so that's one chip saved although I realise that hardly matters).

One major disadvantage of using just the three ports is that, we only had 24 bits to share among all the devices. The ADC definitely required an entire port since a port can only work as either an input or output device but not both at the same time. We knew we needed a bit each for the speaker, the timer and the oscilloscope's Z-input. This left us with 13 bits to be shared between the X and Y inputs. This actually made perfect sense. The X input would require twice the number of bits as the Y inputs (consider a semicircle with a certain radius, it would be twice as long horizontally than vertically). We decided to keep 7 bits (128 values) for X and 6 bits (64 values) for Y. I realise this might seem like a pretty low resolution but once again everything fell in place perfectly. I had decided to keep the operational range of the system between 10 and 60cm since beyond that the voltages returned by the sensor via the ADC were highly inconsistent (e.g. 70cm often would return the same voltage as 75cm). Even between 10 and 60cm, distances returned were not very accurate. I would say the accuracy was at its best at 1cm division between 10 and 30cm and about 2cm from then on. So, having one pixel to represent 1cm on the oscilloscope seemed like a very good idea, which is exactly what we had with 128 bits for X (64cm or pixels on each side) and 64 bits for Y (64 bits or pixels).

Look-Up Tables and Arrays

Look up tables for were used for almost all conversions, simply because some functions were simply too difficult to implement in assembly, or would take up too many clock cycles. Also, we had enough memory to incorporate these lookup tables without having to worry about any of the instructions or the stack overlapping with them (please see the appendix for these tables)

Lookup tables were also used to store ADC values that directly mapped to distances stored in hex, since the function relating them seemed to be exponential and it would be very difficult to do this conversion in assembly.

The values for high and low reload registers to be put into timer0 for each angle were also put in two separate lookup tables. Even though the desired function reload = 2.844*angle + 208 could be easily performed by the Z80 multiple instruction, it seemed like a waste of time doing this so often when looking up a table is so much less time-consuming.

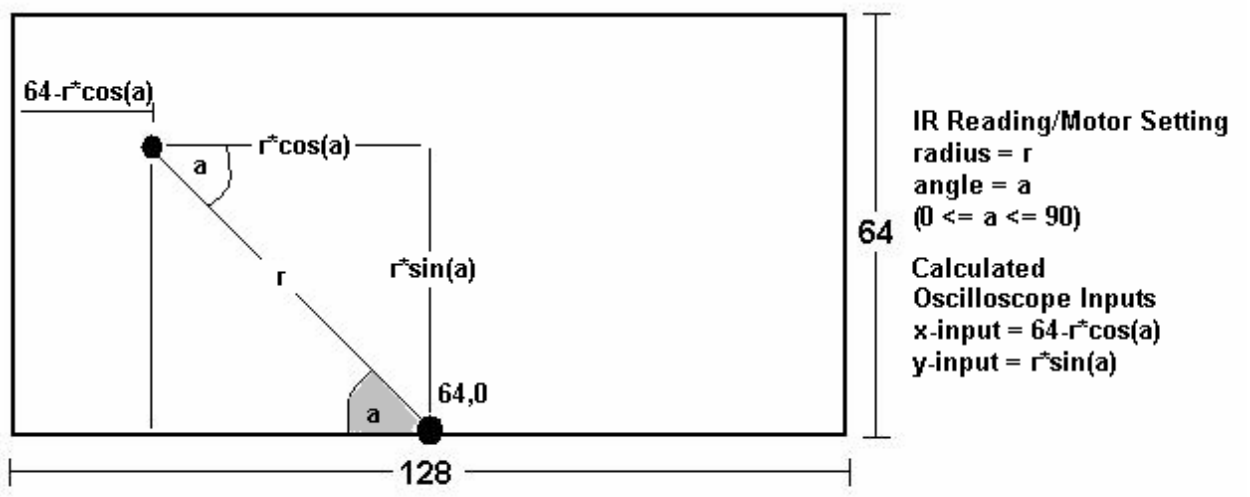
In the profiler mode two arrays are used to store x and y coordinates of IR readings converted into oscilloscope co-ordinates by the get_coords module. The x and y arrays start at memory locations FA00 and F900 respectively and use the angle as an offset. FA00 and F900 were

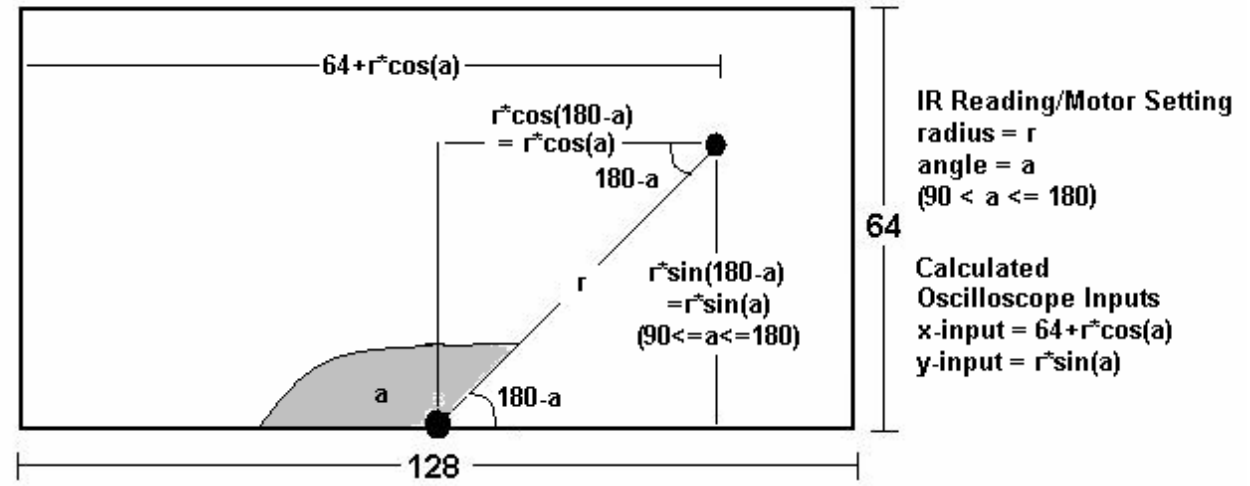
chosen since they are large enough to not overlap with any part of the program memory and small enough not to overlap with the stack, which starts at FFFF.

Please see next topic on the use of lookup tables for finding cartesian co-ordinates from polar ones. For a full list of lookup tables please see the appendices.

Conversion of Polar to Cartesian Co-ordinates

The procedure get_coords converted the polar co-ordinates stored in global variables angle and radius to Cartesian co-ordinates and stored them in global variables x_coord and y_coord. These were not really the orthodox polar or Cartesian co-ordinates, since the angle 0 started on the x-axis and went clockwise. The Cartesian co-ordinates were actually oscilloscope co-ordinates, so real (0,0) would be returned as (64,0) (since with a range of 0-128 on the x-axis, 64 is the centre). The lookup tables stored sine and cosine for each angle in 8 bits in a fixed point format (assuming the point is right before the first bit), e.g. Sine 90 = 1 is stored as FFh meaning 0.11111111, or Sin 30 = 0.5 is stored as 80h meaning 0.10000000. Only absolute cosine values were put into the table, these were later taken into account when calculating oscilloscope co-ordinates. The y-coordinate would simply be radius*sine(angle). This is done by putting the value for the corresponding angle from the sine table in one register (h) and then multiplying it with another register that contains the radius (l). The result is stored in hl, if we just ignore the last 8 bits of the results (ignore register l and only take into account h), we have the result of the multiplication without the fraction bit. The x-coordinate is calculated similarly by multiplication, only afterwards the multiplication result is added to 64 for angles >= 90 or subtracted from 64 for angles < 90. The diagram below explains the geometrical calculations/methods used to calculate the x and y inputs of the oscilloscope

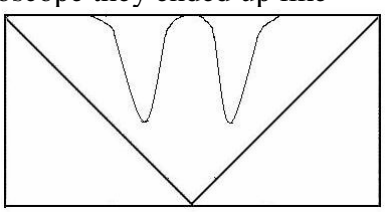
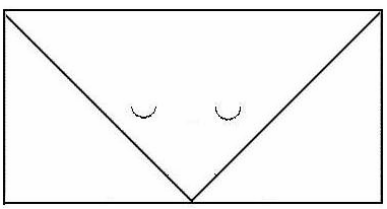




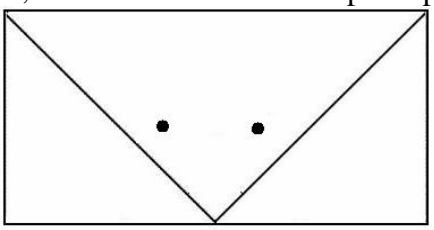
The only alternative to storing the sine and cosine values in tables would be to actually have a function that calculates sine and cosine values using Taylor expansions for sine and cosine which (like almost everything else) are simply too difficult and time consuming to program in assembly. There was also the incredibly insane idea of using two dimensional tables (one each for sine and cosine) which mapped distances and sine/cosine values to their multiplicands (or the Cartesian co-ordinates), however we were not given "that much" memory to work with.

Radar

I faced one particular problem when detecting objects in radar mode. The cylindrical tubes we were provided with gave proper readings of distances only the sensor detected the centre of the tubes. E.g. an object at 20cm and 90 degrees, would be read so only at 90 degrees, however at 92 or 88 degrees (which are the circular edges of the tube), a distance reading of 50 cm might be returned. So for two objects picked up during a sweep, when points were plotted on the oscilloscope they ended up like


when you would expect

.

The ideal situation would be to show a dot each for each object detected. This obviously was a software flaw. I had to rewrite the radar module such that, once something is detected < 60cm, it is treated as the beginning of the object and its angle and distance are stored in variables ObjectAngle and ObjectDistance respectively (which were initialised to 0 and 60 at the start of a sweep). As subsequent points of the object are detected, ObjectAngle and ObjectDistance are updated if the current distance reading is less than ObjectDistance, such that ObjectDistance finally stores the nearest distance and ObjectAngle thus holds the angle at which the centre of the object is located. When a reading >=60cm is detected again, we know that is the end of the object and thus the co-ordinates of the object are plotted on the scope and displayed on the LCD (using ObjectDistance and ObjectAngle, which are then reset to 64 and 0 respectively, allowing the detection of a new object). So, in the end the oscilloscope display looked like this



Speaker

We also had a major problem getting the speaker working. The speaker was set such that a square wave of frequency dependent on the ADC value was being outputted to bit6 of PortC. However this did not make any sound when the speaker was connected. I checked the output of the bit6 using a probe on the oscilloscope to discover that the waveform was well within the audible frequency range. Strangely enough, when I wrote a separate procedure to test the speaker, it worked perfectly. My partner was having similar problems and his test procedures worked perfectly too but the speaker stopped working when it was integrated into the main program (for the radar, tracker or profiler). It later occurred to us that it may be because, only one chip is used to drive all the ports, all of which are used in the main program but not in our test speaker programs. Since the speaker had a high impedance and other bits of all the ports were being set to 5Vs at the same time, the output was simply not powerful enough to drive the speaker. But then, we had absolutely no idea how to solve this problem. This is when we had a stroke of genius. I don't remember which one of us had this brilliant idea, I would like to think it was my idea though, which was to ask the lab technician Ron what could be done about this. He suggested putting in a capacitor in series with the speaker and that worked perfectly, we were finally getting a nice smooth sound quite distinct over different distance readings. We really owed Ron for this one and we also owe him for being very understanding when we somehow managed to blow up the CMOS chip that drives the ports (in a completely unrelated incident when simply everything stopped working and I had no idea why until later I found out that the ports were not responding at all).

TESTING AND EVALUATION

As you can see a more or less bottom up approach was used to put the system together. Each component was first tested separately using separate test programs and then the entire program (i.e. each of the operation modes) was tested later.

Testing separate components

I/R Sensor Testing

As explained earlier, the sensor was calibrated by putting an object at distances between 10 and 60cm at every 1cm and then taking reading the value given by the ADC (I had initially used LEDs to display these values, but later I wrote a program which displayed the value in hex on the LCD display). Three readings were taken at each distance and the average was taken, forming a ADC reading-to-distance table (please see tables in appendix). A test procedure was written to check whether this works by simply incorporating the table into this test program and then using the get_distance module (see Software modules) to convert the ADC readings into distances, which were displayed on the LCD. I placed objects at different distances and took down values the LCD showed and checked whether the distance displayed was the same as the actual distance from the sensor the object was placed at. I was very happy to see these readings were nearly perfect between 10 and 30cm, had an error of +/-1 from 30-40cm, +/-2 from 40-50 cm and about +/-4 from 50-60cm.

The following table shows the calibration readings for the IR sensor

Distance(cm)	Reading1	Reading2	Reading3	Average	Adjusted
9	F3	F6	F9	F6	
10	E0	E4	E5	E3	E3
11	D1	D0	CC	CF	CF
12	C2	C0	C1	C1	C1
13	B0	B3	B0	B1	B1
14	A2	A5	A5	A4	A4
15	9B	97	99	99	99
16	91	92	8D	90	90
17	86	89	89	88	88
18	80	81	85	82	82
19	7C	7E	7A	7C	7C
20	77	74	74	75	75
21	73	71	6C	70	70
22	6A	6D	6D	6C	6C
23	69	67	6B	69	69
24	61	65	66	64	64
25	61	5F	60	60	60
26	5D	5F	5B	5D	5D
27	5A	5D	5A	5B	5B
28	58	57	59	58	58
29	54	57	54	55	55
30	52	53	54	53	53

31	51	50	4F	50	50
32	4E	4E	4B	4D	4D
33	4A	4C	4B	4B	4B
34	49	4B	47	49	49
35	47	49	45	47	47
36	48	44	43	45	45
37	43	44	42	43	43
38	3F	42	42	41	41
39	42	40	3E	40	40
40	3F	3D	41	3F	3F
41	3E	3F	3D	3E	3E
42	3C	3E	3D	3D	3D
43	3A	3D	3D	3C	3C
44	3D	3A	3A	3B	3B
45	3A	3C	38	3A	3A
46	36	37	3B	38	38
47	35	38	38	37	37
48	37	34	34	35	35
49	33	33	36	34	34
50	31	36	31	32	33
51	32	32	35	33	32
52	32	33	33	32	31
53	31	2F	30	30	30
54	2F	2E	33	30	2F
55	2F	2D	2D	2D	2E
56	2E	2A	32	2E	2D
57	2A	2B	2D	2B	2C
58	2B	2C	2A	2B	2B
59	2A	29	28	29	2A
60	28	2B	2B	2A	29
61	26	28	2A	28	28
62	29	27	26	27	27
63	28	26	27	27	26
64	25	25	26	25	25

(The average readings were later adjusted to form an ADC-to-Distance table since at some distances e.g. 53 and 54 the average ADC reading seemed to be the same at 30h, so I had adjusted 54 to be 2Fh instead. Please see Appendix C for the final lookup table made from this data. Please note that my partner and I calibrated the sensor together, so we are very likely to have almost the same distance look-up table)

Tests run with different distances to check if sensor calibrated properly

Actual Distance	Distance Reading1	Distance Reading2	Distance Reading3	Distance Reading4	Distance Reading5	Average Distance	Max Error +/-	Standard Deviation
10	10	10	10	10	10	10.00	0	0.000
11	11	11	11	11	11	11.00	0	0.000
12	12	12	12	12	12	12.00	0	0.000
13	13	13	13	13	13	13.00	0	0.000
14	14	14	14	14	14	14.00	0	0.000
15	15	15	15	15	15	15.00	0	0.000
16	16	16	16	16	16	16.00	0	0.000
17	17	17	17	17	17	17.00	0	0.000
18	18	18	18	18	18	18.00	0	0.000
19	19	19	19	19	19	19.00	0	0.000
20	20	20	20	20	20	20.00	0	0.000
21	21	21	21	21	21	21.00	0	0.000
22	22	22	22	22	22	22.00	0	0.000
23	23	23	23	23	23	23.00	0	0.000
24	24	24	24	24	24	24.00	0	0.000
25	25	25	25	25	25	25.00	0	0.000
26	26	26	26	26	26	26.00	0	0.000
27	27	27	27	27	27	27.00	0	0.000
28	28	28	28	28	28	28.00	0	0.000
29	29	29	29	29	29	29.00	0	0.000
30	30	30	30	30	30	30.00	0	0.000
31	31	31	31	31	31	31.00	0	0.000
32	32	32	32	32	32	32.00	0	0.000
33	33	33	34	33	33	33.20	1	0.400
34	34	35	34	33	34	34.00	1	0.632
35	35	34	34	34	35	34.40	1	0.490
36	36	36	36	36	36	36.00	0	0.000
37	37	37	38	37	38	37.40	1	0.490
38	38	38	38	37	39	38.00	1	0.632
39	39	39	40	40	38	39.20	1	0.748
40	41	40	41	39	40	40.20	1	0.748
41	41	41	42	42	42	41.60	1	0.490
42	42	41	43	42	41	41.80	1	0.748
43	43	45	42	42	44	43.20	2	1.166
44	44	44	43	46	45	44.40	2	1.020
45	46	45	45	45	45	45.20	1	0.400
46	46	47	46	45	47	46.20	1	0.748
47	45	47	48	47	47	46.80	2	0.980
48	49	48	48	48	50	48.60	2	0.800
49	49	49	50	49	50	49.40	1	0.490
50	50	50	50	50	51	50.20	1	0.400
51	51	51	52	51	51	51.20	1	0.400
52	52	53	50	52	53	52.00	2	1.095
53	53	53	55	52	50	52.60	3	1.625

54	54	55	54	52	55	54.00	2	1.095
55	56	55	57	54	57	55.80	2	1.166
56	57	53	58	56	57	56.20	3	1.720
57	58	60	56	56	59	57.80	3	1.600
58	59	58	60	57	59	58.60	2	1.020
59	60	61	57	62	59	59.80	3	1.720
60	62	59	63	60	59	60.60	3	1.625
61	64	58	58	62	57	59.80	4	2.713
62	62	N	59	60	63			
63	64	N	62	N	N			
64	N	60	N	N	62			

(N = Nothing detected)

Servo Motor Testing

The increase/decrease angle procedures along with the rotate_to_angle procedures were used to rotate the sensor to different angles. The rotate_to_angle procedure uses the reload lookup tables to put an appropriate value into the timer such that the required pulse is produced. I tried several angles and then later sweeps (slowly increasing the angle after a short period of time) to check whether the motor worked properly and whether in fact the lookup tables generated using linear regression was right and as expected everything was fine. Finally, after each sweep between certain angles, I made it rotate back to its start angle and repeat the sweep.

Speaker Testing

The speaker was initially tested by feeding its timer different reload values just to check whether it made a sound, and later a speaker test was added to the IR sensor test program, such that the speaker made a sound depending on the distance read (this was done by putting the ADC value read into the speaker timer's reload register). This made a very distinct sound at different distances however the program seemed to crash when the object was too far away, this was because when the ADC value was too small, the frequency was too high and the speaker timer generated too many interrupts, thus stopping the rest of the program from executing. This problem was fixed by doing an OR 0x10 on the ADC value before putting it into the speaker timer.

Oscilloscope/Cartesian Coordinates Testing

I started off by first writing a procedure that made simple shapes such as boxes or lines appear on the oscilloscope using preset x and y co-ordinates (using the plot_point procedure). The z-input was checked by switching the beam off for certain points and later seeing whether these points appeared on the oscilloscope and indeed they did not (or at least they appear very lightly).

The get_coords procedure converted polar co-ordinates to cartesian co-ordinates (rather polar to oscilloscope co-ordinates), by looking up the sine and cosine tables for each angle and then multiplying them with distances. I wrote a simply test program which stored these x and y coordinates in memory and checked whether get_coords works by running the program on the ZIM emulator for the Z80. I then wrote the display_angle_distance which displayed current angle, distance, x and y coordinates on the LCD displays and this seemed fine for all test values to. I later used plot_point to plot some points on the oscilloscope. A more comprehensive test was carried out I suppose, when I wrote a procedure that starts with a certain distance and plots

points for angles 0 to 180 degrees and then increments the distance, plot points for angles 0 to 180 degrees and so on. This gave images of semicircles of increasing radii on the oscilloscope (with the beam moving clockwise), which is exactly what we would expect.

LCD details of some points while plotting semi-circles

```
00■ 10cm -10,00
30■ 10cm -08,05
60■ 10cm -05,09
90■ 10cm 00,10
120■ 10cm 05,09
150■ 10cm 09,05
180■ 10cm 10,00

00■ 25cm -25,00
20■ 25cm -23,08
40■ 25cm -19,16
60■ 25cm -12,22
80■ 25cm -04,25
100■ 25cm 05,25
120■ 25cm 13,22
140■ 25cm 20,16
160■ 25cm 24,08
180■ 25cm 25,00

00■ 40cm -40,00
15■ 40cm -38,10
30■ 40cm -34,20
45■ 40cm -28,28
60■ 40cm -20,35
75■ 40cm -10,39
90■ 40cm 00,40
105■ 40cm 11,39
120■ 40cm 20,35
135■ 40cm 29,28
150■ 40cm 35,20
165■ 40cm 39,10
180■ 40cm 40,00

00■ 45cm -45,00
10■ 45cm -44,08
20■ 45cm -42,15
30■ 45cm -38,22
40■ 45cm -34,29
50■ 45cm -28,34
60■ 45cm -22,39
70■ 45cm -15,42
80■ 45cm -07,44
90■ 45cm 00,45
100■ 45cm 08,44
110■ 45cm 16,42
120■ 45cm 23,39
130■ 45cm 30,34
140■ 45cm 35,29
150■ 45cm 40,22
160■ 45cm 43,15
170■ 45cm 45,08
180■ 45cm 45,00

00■ 60cm -60,00
45■ 60cm -42,42
90■ 60cm 00,60
135■ 60cm 43,42
180■ 60cm 60,00

00■ 55cm -55,00
18■ 55cm -52,17
36■ 55cm -44,32
54■ 55cm -32,44
72■ 55cm -16,52
90■ 55cm 00,55
108■ 55cm 18,52
126■ 55cm 33,44
144■ 55cm 45,32
162■ 55cm 53,17
180■ 55cm 55,00
```

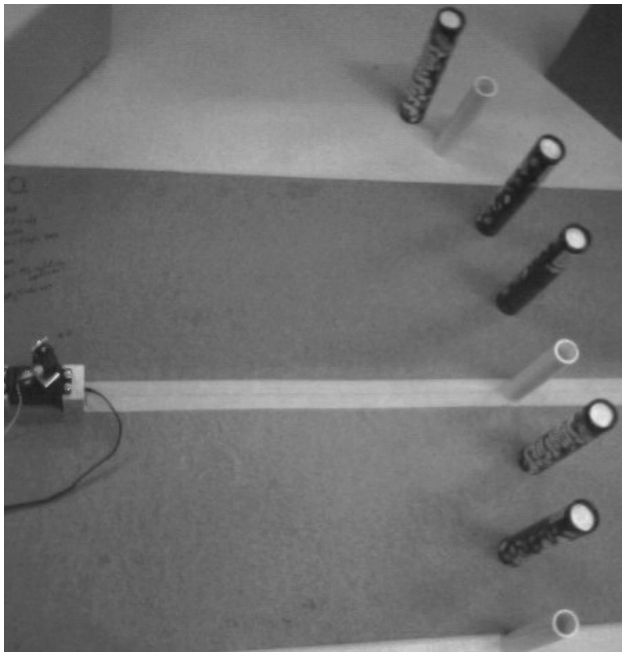
(■ = Degree Sign in LCD font)

System Testing

Each of the modes were put together first and test individually.

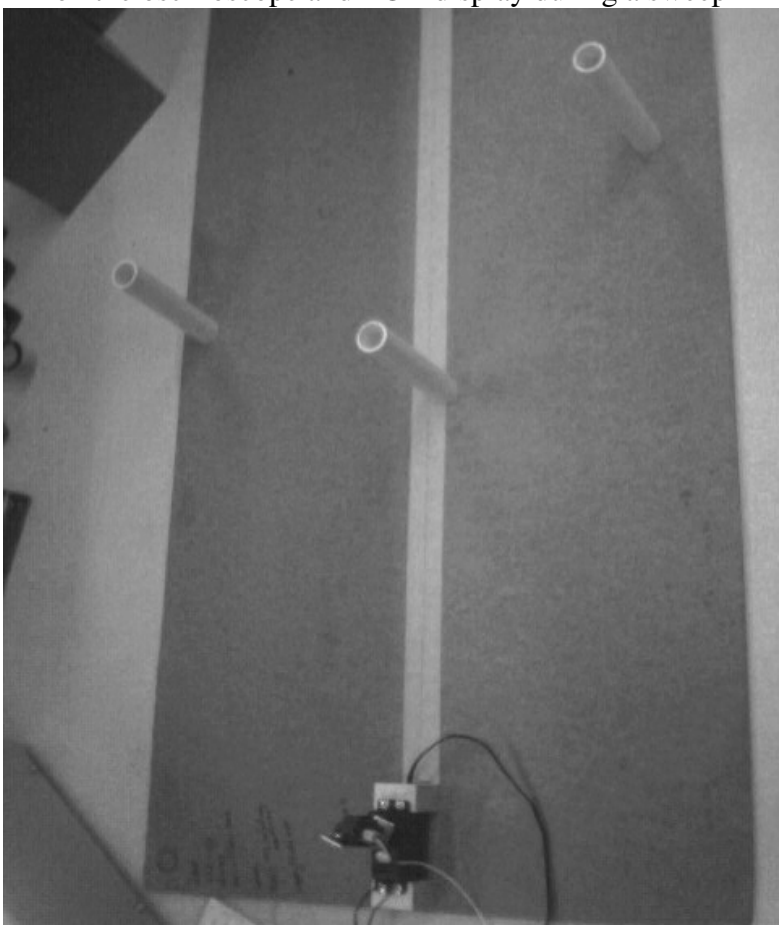
Radar Mode

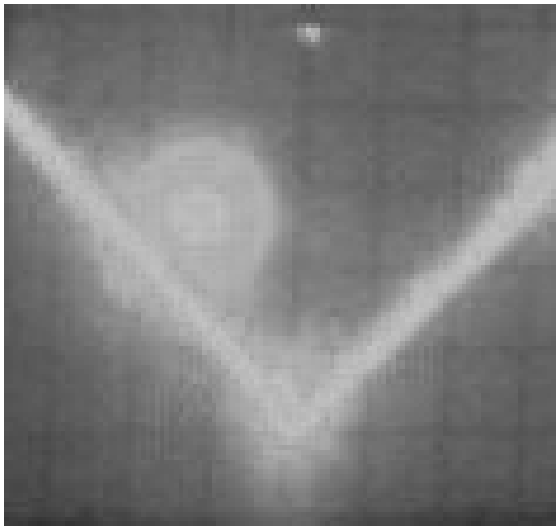
This was a very simple test. Some tubes were placed in front of the sensor and the program was run to see if the oscilloscope gave the proper positions of the objects and whether it gave just one dot on the scope for one object. I realise the specifications required the radar mode to pick up at least 16 targets in the sweep-able angular range, but unfortunately we were provided with just one tube to test our programs on. I used my highly persuasive skills to borrow some others from other benches and improvised by using some smarties tubes. At a far enough distance, the radar picked up ten distinct objects. Since the detection of an object requires there to be a black space between them, and a sweep-able area of 90 degrees at 60cm, this gives us a circumference of 94cm. The side of the tube visible to the sensor is roughly 3cm (i.e. half is circumference). This would allow the detection of at least 20 objects, provided there is enough blank space between two objects placed next to each other. The sounds made by the speaker were quite decent and it corresponded to the distance between the object and the sensor.



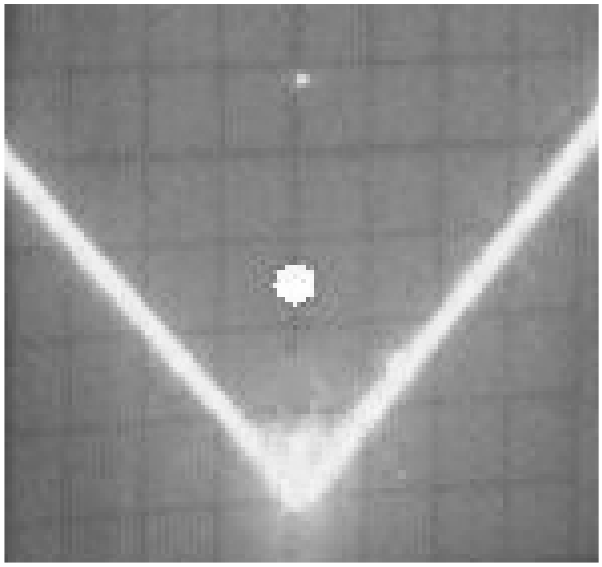

Improvising using Smarties tubes. This would no doubt make a very good Smarties commercial.

The pictures below show the positioning of three objects in radar mode and how they appeared on the oscilloscope and LCD display during a sweep

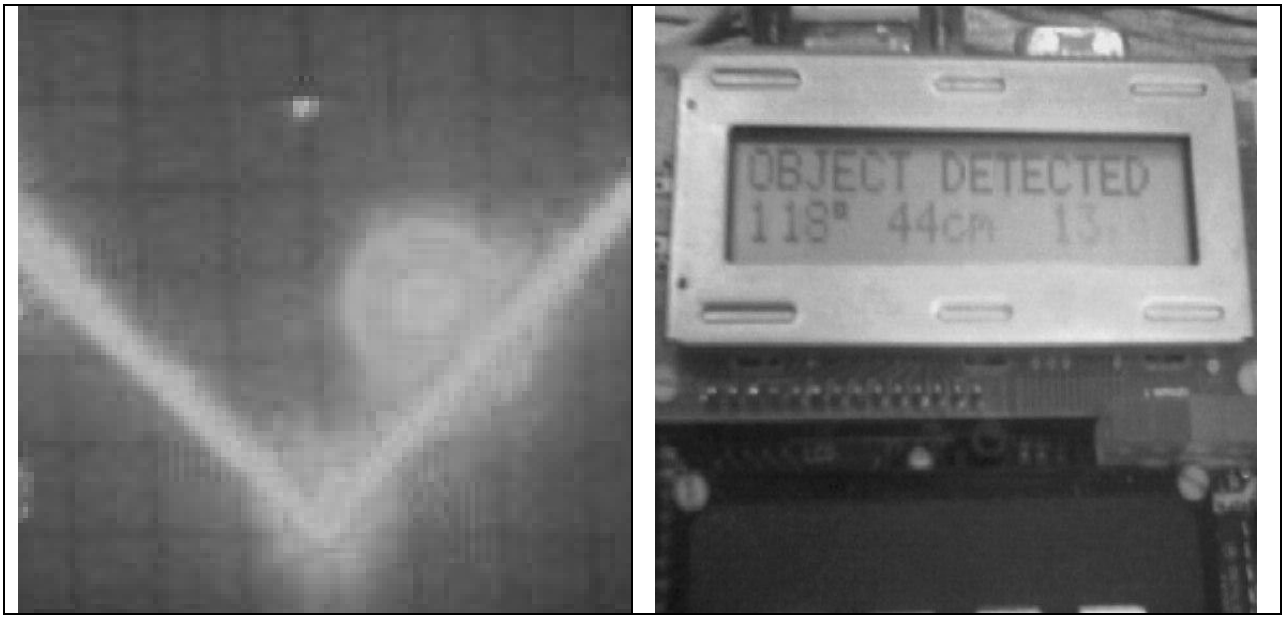





OBJECT DETECTED
64° 38cm -16.34
1
2
3
A

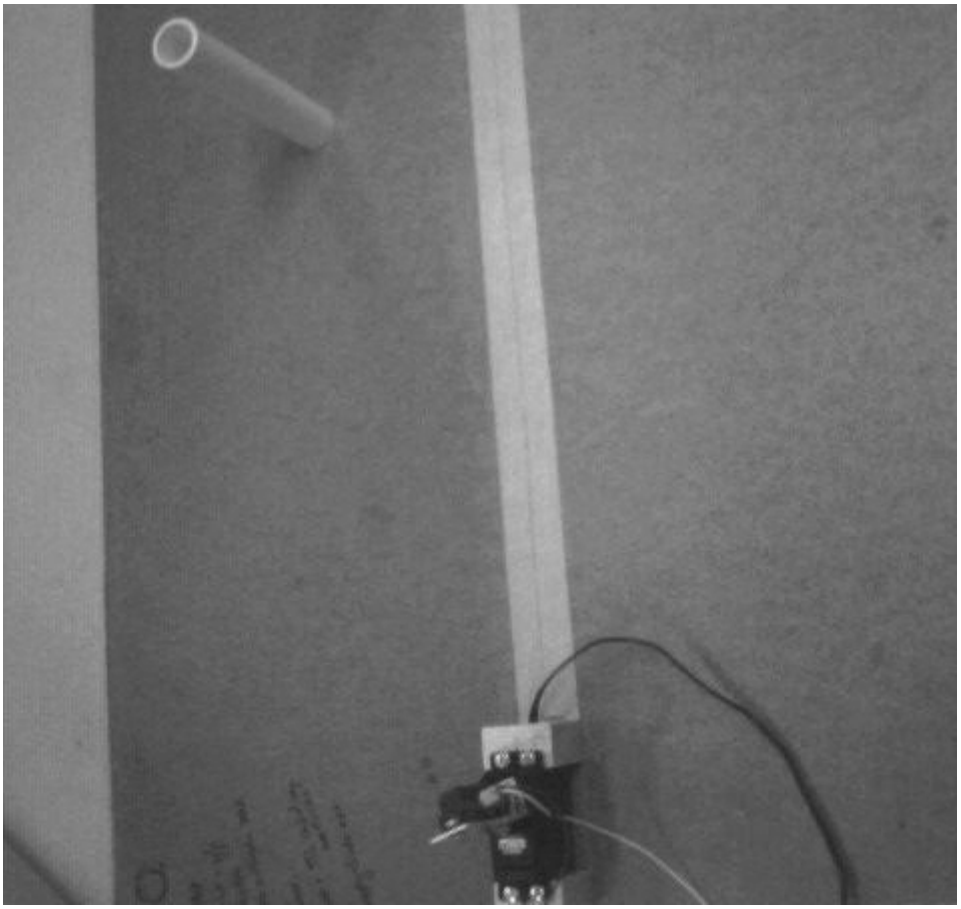



DETECTED

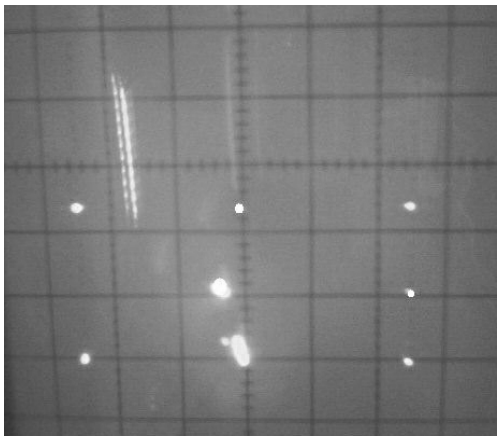


Tracker Mode

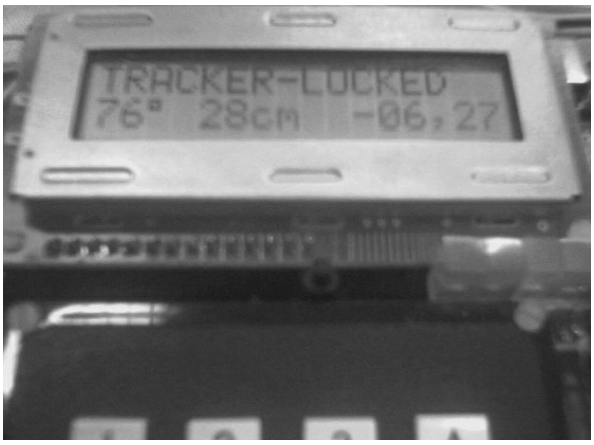
The first part of the tracker test was simply to check whether the motor properly swept until it locked on to an object. Once that proved successful, the actual tracker bit was tested and it seemed to follow the object around quite well (provided it moved slow enough at around 2cm/s). I then moved the object out of range several times and checked whether it made the tracker loop back to the zero degree position and start seeking for the object again. I had one problem though. I had initially forgot to put in checks for the extreme angles 0 and 180 degrees, so everything just went haywire when the object was around those position since then the new reload values were no longer on the reload tables, so a very strange input would be put into the servo motor. I made it so that there were checks at 0 and 180 degrees such that in those instances the tracker would not check to the left or right respectively. The sound effects produced were quite satisfactory as well.



Target placed in front of sensor in tracker mode



Oscilloscope showing position of target



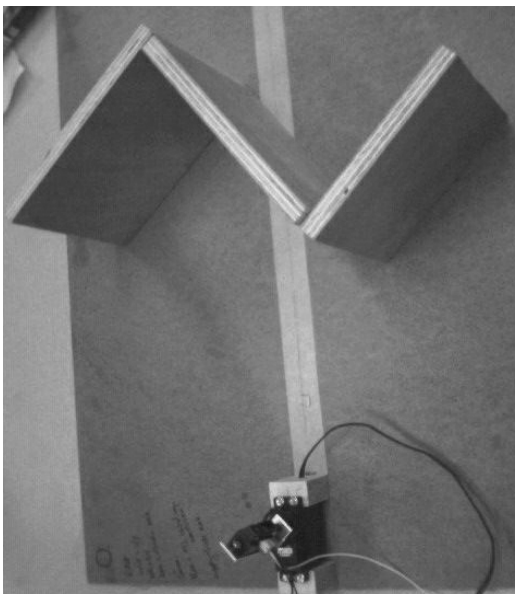
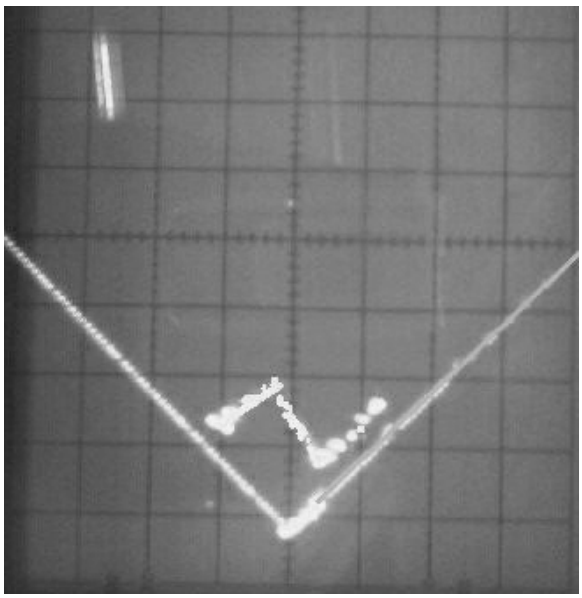
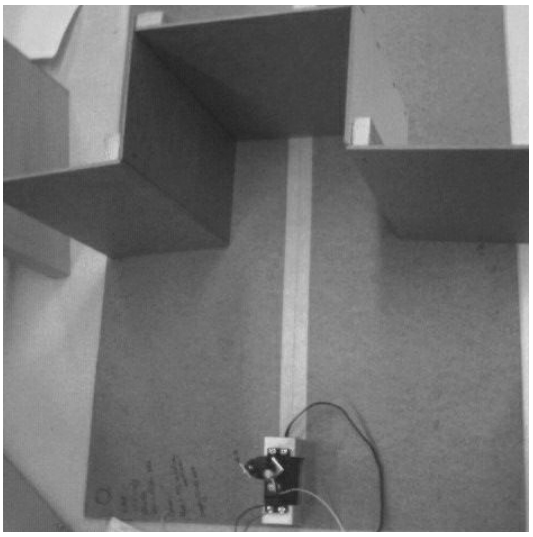
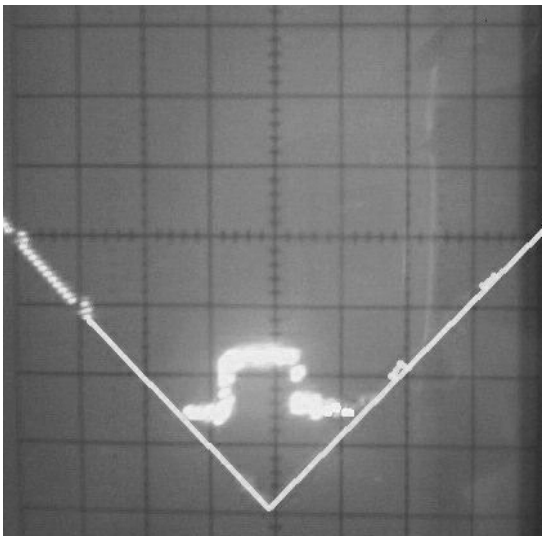
LCD displaying co-ordinates of target

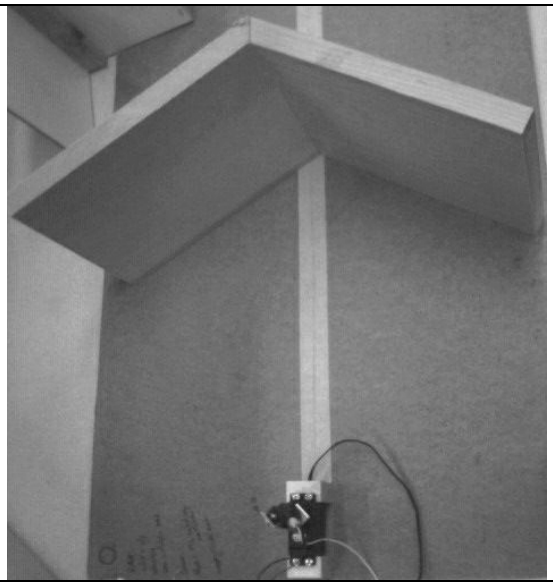
Profiler Mode

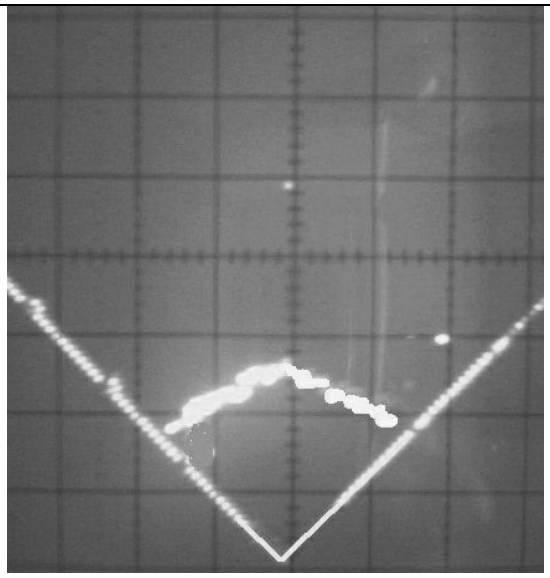
The profiler was tested with differently shaped objects placed in front of the sensor and then checking what kind of image was formed on the oscilloscope. The distances on the LCD display also came out more or less right. Many different shapes were used to check the image formed on the screen and the shapes more or less coincided, as well as can be expected from our particular resolution and our sensor. The speaker makes distinct sounds at distinct distances, so the sound gave the listener a rough idea of the shape of the object. For example, a concave object would

give a more or less same sound through out but a straight object would start off with a high frequency, then get lower and then get higher again.

The following are some pictures of objects placed on the sensor and the profile shown on the oscilloscope after a sweep.

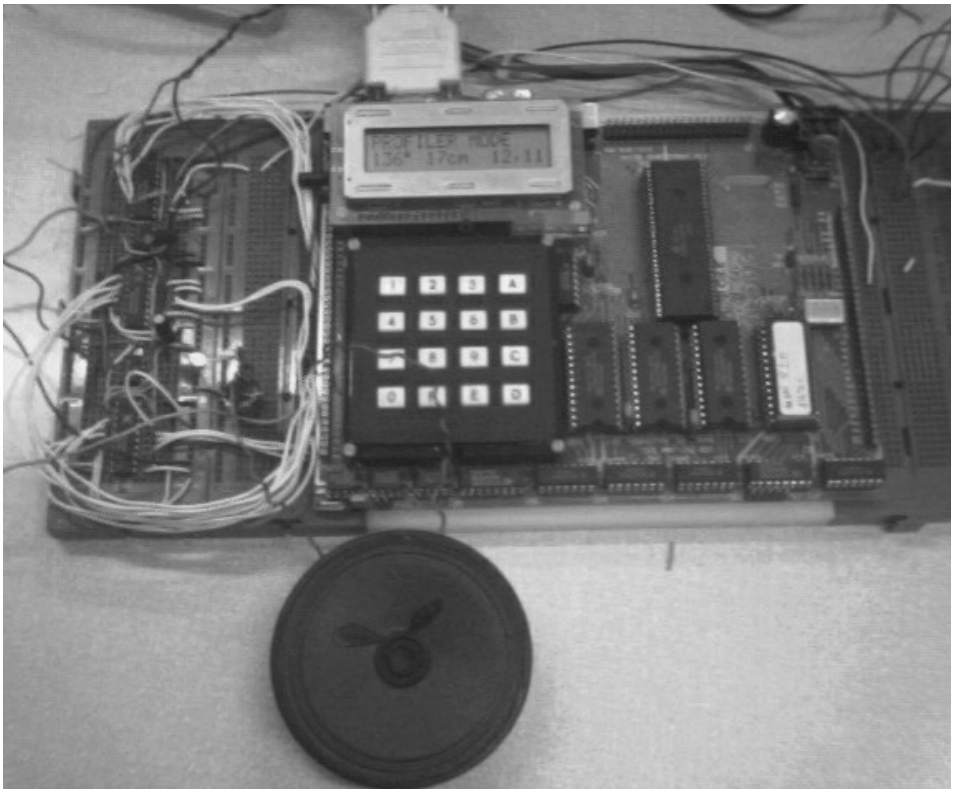
Object Placed	Profile Displayed
	
	





Full System Testing

All this required was to test whether all the modes worked when put in the same program file and whether the menu itself worked. One major problem I faced was the overlapping of labels, which often meant that even after I got rid of all duplicate label names, I had forgotten to change the jump instructions accordingly, so often the tracker looped into the radar and vice versa until I fixed all this. The system seemed to respond fine to all keypad entries, taking it to the appropriate mode for keys 1-3 or back to the main menu for other keys.



Obviously really proud of the simplest hardware design in the lab

TECHNICAL SPECIFICATIONS

InfraRed Sensor	
Operational Range	10-60cm
Distance Accuracy	10-30cm: Max Error +/-0cm, Standard Deviation: 0.00 30-40cm: Max Error +/-1cm, Standard Deviation: 0.41 40-50cm: Max Error +/-2cm, Standard Deviation: 0.81 50-60cm: Max Error +/-4cm, Standard Deviation: 1.19
Servo-Motor	
Angle of sweep	0-180 degrees
Speed of rotation	0.20 seconds/degree
Radar Mode	
Angle of sweep	45 to 135 degrees (every 1 degree)
Maximum Objects Detectable	20
Average Time for Full Sweep	25 seconds
Radar Range	10-60cm
Tracker Mode	
Angle of sweep	0 to 180 degrees (every 3 degrees)
Tracker Accuracy	+/-3 degrees
Seeker Range	10-30cm
Tracker Range	10-30cm
Maximum Speed Track-able	2cm/s
Profiler Mode	
Angle of sweep	45 to 135 degrees (every 1 degree)
Average Time for Full Sweep	30 seconds
Profiler Range	10-60cm
Oscilloscope Display	
Resolution	128*64
X-coordinates	128 discrete values (1 pixel per cm)
Y-coordinates	64 discrete values (1 pixel per cm)
LCD Display	
Display size	16 characters per line * 2 lines
Accuracy of numbers displayed	Dependent on IR sensor reading Values correct to whole numbers (0 decimal places)

COSTING

Component	Qty	Purpose	Unit Price £	Total Price £
Z80 CPU SBC Board	1	The SBC Board with the Z80 Processor, RAM chips, LCD, Keypad etc.	100.00	100.00
Servo Motor	1	Rotates IR sensor	7.50	7.50
IR Sensor	1	Infra-red sensor to detect objects	9.00	9.00
8 ohm Speaker	1	To produce required sound effects	3.38	3.38
DAC0832LCN D/A Converter	2	1 – Perform D/A conversion for X axis 2 – Perform D/A conversion for Y axis	2.74	5.48
LF356N Op Amp	2	To amplify analogue signals for each axis on each D/A converter	0.63	1.26
ADC0804LCN A/D Converter	1	To convert analogue voltage read by the IR sensor to be read	2.70	5.40
Capacitors	5	For speaker, ADC, Z-input, smoothing opamps outputs	0.11	0.55
Resistors	6	For providing reference voltages to ADC and DACs	0.05	0.30
Wires, misc.		For connecting components	1.00	1.00
			TOTAL £	**133.87**

(Please note these costs are for putting together one unit of the system, however if it were to be mass-produced then the costs per unit would be slightly less)

LIMITATIONS AND FUTURE IMPROVEMENTS

As far as limitations go, the most apparent one would probably be the resolution of our oscilloscope display i.e. 128*64. The most obvious reason for doing that was to save time (and a chip) so that we could only use the ports for I/O. Most other teams were using a 256*256 resolution. However I believe this is unnecessary because of the inaccuracy of the IR sensor. An improvement I would suggest is the use of a more accurate/sensitive IR sensor such that distances would be calculated with greater accuracy and then a higher resolution could be used to give more near to exact positions of objects on the oscilloscope. On top of that, if a 16-bit ADC were used to encode the IR-sensor data, the distance measurements would have greater detail (perhaps in millimetres).

One problem with sharing the same ports for timers and the y-axis meant that while the timers were running it was not possible to change the position of points on the oscilloscope screen. This meant that our screen was slightly more flickering than that of other teams. This is one reason why we really should have implemented a discrete I/O scheme, leaving the timer bits separate from anything else.

A better IR sensor would also increase the operational range of all our modes.

At the moment, the radar mode only shows objects as it sweeps but does not store the locations of the objects. It does however give a nice effect of an object popping up on the oscilloscope and then fading away as the next object is detected. But I suppose, if I stored the positions of all objects in memory and then displayed the previously scanned objects in the same sweep while scanning, it would look much more sophisticated and give more information.

Among other things, I would have also preferred to have a faster servo-motor, which would mean faster sweeps for the radar and profiler modes, and the tracker mode would also be able to track faster moving objects.

I would also have liked to allow the user to manually enter angle of sweeps, acceptable distances ranges etc. for each of the modes using the keypad.

Using some fancy vector graphics, it would have been nice to show the angle, distance etc. on the oscilloscope display itself as graphics. All this would require is to have bitmap images of numbers and then showing them on the screen starting at a particular points of it.

Most of these suggested improvements are base on what I saw done by other teams doing the MCP course. Despite all the limitations of our system, I think I can safely say that ours is probably the simplest and cheapest hardware solution and I do feel rather smug about what I have put together.

APPENDIX A – THE CODE

```
jp main

.space 29

vector_table:             # setting up vector table for servicing interrupts

.int 0x0000               # interrupt 0 unused
.int keypad_int           # interrupt 1 generated by keypad key press
.int servo_timer_toggle   # interrupt 2 generated by timer0 (servo timer)
.int speaker_timer_toggle # interrupt 3 generated by timer1 (speaker timer)

################ MEMORY LOCATIONS DECLARED FOR VARIABLES, CONTROL REGISTERS ETC
################
lcd_out = 0xB9            # output for lcd screen in ASCII
keypad_in = 0xB4          # input from keypad
ADC = 0xB0                # port A connected to ADC
X = 0xB1                  # port B connected to Z and X inputs
Y = 0xB2                  # port C connected to servo, speaker and Y inputs
CSR = 0xB3                # command status register
rr_high0 = 0x0F           # high reload register for timer 0 (controlling servo)
rr_low0 = 0x0E            # low reload register for timer 0 (controlling servo)
rr_high1 = 0x17           # high reload register for timer 1 (controlling speaker)
rr_low1 = 0x16            # low reload register for timer 1 (controlling speaker)
TCR = 0x10                # timer control register - used to enable timers 0 & 1
TDR0 = 0x0C               # lower byte of timer data register for timer 0
TDR1 = 0x14               # lower byte of timer data register for timer 1
IVLR = 0x33               # interrupt vector low register
ITCR = 0x34               # interrupt/trap control register
distance = 0xFC10         # stores current distance read
radius = 0xFC15           # radius in polar co-ordinates to be passed to procedure
get_coords as parameter
angle = 0xFC11            # stores current angle of servo or point being plotted
tangle = 0xFC12           # temporarily holds angle for conversion
number = 0xFC17           # number in binary to be passed to procedure
display_bin_to_dec as parameter
x_coord = 0xFC18          # holds x-coordinate of point converted from polar to
cartesian by get_coords
y_coord = 0xFC19          # holds y-coordinate of point converted from polar to
cartesian by get_coords
mode = 0xFC1A       # stores current mode (1=Radar, 2=Tracker, 3=Profiler)
midangle = 0xFC35         # stores middle angle in tracker mode before checking left
or right
middistance = 0xFC80      # stores distance at the middle angle in tracker mode
objang = 0xFC26           # stores angle at which centre of last object in radar
mode was detected
objdis = 0xFC25           # stores distance at which centre of last object in radar
mode was detected
################################################################################
############

main:
      call init_port      # initialise ports such that A=input, B=output, C=output
      call init_vector_table    # initialise vector table for interrupts

      ld a, 0x00          # sets mode to 0 (menu) initially
      ld (mode), a

main_menu:
```

```
	call clear          # clear lcd screen
	call display_menu   # lcd displays message showing which key to press for
which mode

menu:

	ei                  # enable interrupt to allow keys on pad being pressed and
mode being changed by keypad_int

	ld a, (mode)        # loop around if mode=0
	cp 0x00
	jp z, menu

	ld a, (mode)        # run radar-mode if mode=1
	cp 0x01
	jp z, radar

	ld a, (mode)        # run tracker-mode if mode=2
	cp 0x02
	jp z, tracker

	ld a, (mode)        # run profiler-mode if mode=3
	cp 0x03
	jp z, profiler

	jp menu


radar:                          # start of radar-mode
	call init_angle45       # set servo to start sweep at 45degrees
	call enable_servo_timer # run servo timer starting rotation

	call clear              # clear lcd
	call display_rotating   # lcd displays message saying servo rotating
	call delay_servo_full   # allow servo enough delay to rotate to 45degrees

	ld a, 0x40              # initialising last object detected at 0x40
(infinty)
	ld (objdis), a

radar_loop:

	ld a, (mode)            # if mode changed return to main menu
	cp 0x01
	jp nz, main_menu

	call clear              # clear lcd
	call get_distance       # calculate current distance and put it in
(distance)
	call display_radar_mode # lcd displays message indicating radar mode
	call display_angle      # lcd displays angle of servo


	ld a, (distance)        # if distance > 60 then there is no object detected
	cp 0x3C
	jp z, checkblank
	ld b, a                 # b = current distance
	ld a, (objdis)          # a = previous central distance of current object
	cp b                    # prev distance - current distance
	jp c, checkblank
```

```
      ld a, (distance)          # if current distance < previous distance
      ld (objdis), a            # set new object distance and angle as current
values
      ld a, (angle)
      ld (objang), a

      jp next_angle


checkblank:                     # if there is a blank space means end of object

      ld a, (objdis)            # if there was no previous object (object distance >
60) then rotate to next angle
      cp 0x3C
      jp nc, next_angle


      ld a, (angle)       # temporarily stores angle in stack since calculations
affect it
      push af


      ld a, (objang)            # get cartesian coordinates for (objang, objdis)
      ld (angle), a
      ld a, (objdis)
      ld (radius), a
      call get_coords

      call clear
      call display_object # lcd displays message saying objected detected
      call display_angle_distance      # display objang, objdis, cartesian
coordinates on LCD


      call plotoutline          # plot outline on oscilloscope showing sweep-able
region
      call plot_point           # plots point using cartesian co-ordinates of object
detected
      call beam_on              # leave oscilloscope beam on

      call make_sound           # use speaker to make sound indicating detection of
object

      call plotoutline          # plot outline on oscilloscope showing sweep-able
region
      call plot_point           # plots point using cartesian co-ordinates of object
detected
      call beam_on              # leave oscilloscope beam on

      pop af                    # restore value of angle
      ld (angle), a

      ld a, 0x40                # set object central distance as 64 (infinity), i.e.
no object being detected
      ld (objdis), a


      ld a, (angle)       # if angle >= 135 re-run radar (rotate back to 45 and
sweep again)
      cp 0x87
      jp nc, radar
```

```
next_angle:

      call increase_angle # if angle < 135, increment angle and then rotate servo to
that angle
      call rotate_to_angle


jp radar_loop


tracker:                          # main module for tracker mode

      call clear
      call beam_off

seeker:
      call init_angle0          # set servo to start sweep at 0degrees
      call enable_servo_timer   # run servo timer starting rotation
      call clear                # clear lcd
      call display_rotating     # lcd displays message saying servo rotating
      call delay_servo_full     # allow servo enough delay to rotate to 45degrees

seeker_loop:

      ld a, (mode)
      cp 0x02
      jp nz, main_menu


      call clear                # clear lcd screen
      call get_distance         # calculate current distance and put it in
(distance)
      call display_tracker_seeking     # lcd displays message indicating tracker
seeking
      call display_angle_distance      # display current angle and distance read
      call plot_outline
      call get_coords
      call plot_point           # plot current reading on oscilloscope
      call beam_on

      ld a, (distance)          # if object located at distance<30 then go to
trackingmode
      cp 0x1E
      jp c, tracker_start


change_sangle:

      #check if angle is >= 180
      ld a, (angle)
      cp 0xB2

      jp c, next_sangle

      jp seeker                 # if angle >-180 then restart seeker from 0 degrees


next_sangle:

      call increase_angle3      # increase angle by 3
```

```
      call rotate_to_angle      # run timer for enough time to allow rotation

jp seeker_loop


tracker_start:
      ld a, (angle)       # store current angle and distance as middle angle and
distance
      ld (midangle), a

      call get_distance
      ld a, (distance)
      ld (middistance), a

right_check:

      ld a, (mode)              # check if mode has been changed via keypad
      cp 0x02
      jp nz, main_menu

      ld a, (midangle)          # rotate to current angle
      ld (angle), a
      call rotate_to_angle

      call get_distance         # set current distance as mid distance
      ld a, (distance)
      ld (middistance), a

      ld a, (distance)          # if mid distance >= 64 it means object has moved
out of range, so start seeking again
      cp 0x3F
      jp nc, tracker

      call clear                # clears lcd
      call display_tracker_locked     # display message saying tracker locked
      call display_angle_distance     # display angle, distance, cartesian
coordinates
      call plotoutline          # show sweep-able region on oscilloscope
      call get_coords           # get oscilloscope co-ordinates for
midangle,middistance
      call plot_point           # plot point showing current angle,distance on
oscilloscope
      call make_sound           # make sound indicating distance from sensor
      call plotoutline          # show sweep-able region on oscilloscope
      call get_coords           # get oscilloscope co-ordinates for
midangle,middistance
      call plot_point           # plot point showing current angle,distance on
oscilloscope

right:
      ld a, (midangle)          # if angle >= 178 do not check right
      cp 0xB0
      jp nc, left
      add a, 0x03               # angle = midangle + 3
      ld (angle), a
      call rotate_to_angle      # enable timer and allow servo time to rotate to
midangle+3
      call get_distance         # read current distance into (distance)

      ld a, (middistance)
      ld b, a                   # b = middistance
```

```
      ld a, (distance)          # a = rightdistance
      cp b                      # a-b = rightdis-middis
      jp nc, left               # if rightdis>=middis then object has not moved to
right, so check left
      ld a, (angle)             # if rightdis<middis then object moved to right, set
midangle to current angle and so check to the right again
      ld (midangle), a

      jp right_check

left:                           # checking if object moved to left
      ld a, (midangle)          # if angle < 3 do not check left
      cp 0x03
      jp c, right_check
      sub 0x03                  # angle = midangle - 3
      ld (angle), a

      call rotate_to_angle      # enable timer, allow servo time to rotate to
midangle-3

      call get_distance         # read current distance into (distance)
      ld a, (middistance)
      ld b, a                   # b = middistance
      ld a, (distance)          # a = leftdistance
      cp b                      # a-b = leftdis-middis
      jp nc, right_check        # if leftdis>=middis then object has not moved to
left, so check right
      ld a, (angle)             # if leftdis<leftdis then object moved to right, set
midangle to current angle and so check to the right again
      ld (midangle), a

left_check:
      ld a, (mode)              # if mode changed via keypad then return to main
menu
      cp 0x02
      jp nz, main_menu

      call clear                # clears lcd
      call display_tracker_locked     # display message saying tracker locked
      call display_angle_distance     # display angle, distance, cartesian
coordinates
      call plotoutline          # show sweep-able region on oscilloscope
      call get_coords           # get oscilloscope co-ordinates for
midangle,middistance
      call plot_point           # plot point showing current angle,distance on
oscilloscope
      call make_sound           # make sound indicating distance from sensor
      call plotoutline          # show sweep-able region on oscilloscope
      call get_coords           # get oscilloscope co-ordinates for
midangle,middistance
      call plot_point           # plot point showing current angle,distance on
oscilloscope

      jp left



profiler:
      call init_angle45         # set servo to start sweep at 45degrees
      call enable_servo_timer   # run servo timer starting rotation
```

```
	call clear		# clear lcd
	call display_rotating	# lcd displays message saying servo rotating
	call delay_servo_full	# allow servo enough delay to rotate to 45degrees

	ld a, 0x40		# initialising last object detected at 0x40
(infinty)
	ld (objdis), a

profiler_loop:

	ld a, (mode)		# if mode changed return to main menu
	cp 0x03
	jp nz, main_menu

	call clear		# clear lcd
	call get_distance	# calculate current distance and put it in
(distance)
	call display_profiler_mode# lcd displays message indicating profiler mode
	ld a, (distance)
	ld (radius), a
	call get_coords		# get cartesian coordinates for (angle, distance)
	call display_angle_distance	# display angle, distance, cartesian
coordinates on LCD


	########### Storing Profile ############# Stores points in cartesian co-
ordinates for each angle if something is detected. X-coordinates for each angle
stored in 0xFA00+angle and Y-coordinates in 0xF900+angle
	ld (distance), a	# if nothing detected (distance>60) no point stored
	cp 0x3C
	jp nc, noppoint


	ld a, (angle)		# Starting at location FA00 and using angle as offset
store x-coordinate for current angle
	ld l, a
	ld h, 0xFA
	ld a, (x_coord)
	ld (hl), a

	ld a, (angle)		# Starting at location F900 and using angle as offset
store y-coordinate for current angle
	ld l, a
	ld h, 0xF9
	ld a, (y_coord)
	ld (hl), a

	call plot_outline	# plot outline indicating sweep-able region
	call plot_point		# plot current point on screen
	call make_sound		# make sound indicating the distance read

	jp change_angle

	noppoint:		# if distance>64, store the point as 0,0 which is not
plotted by show_profile
	ld a, (angle)
	ld l, a
	ld h, 0xFA
	ld a, 0x00
	ld (hl), a
```

```
	ld a, (angle)
	ld l, a
	ld h, 0xF9
	ld a, 0x00
	ld (hl), a

	##########


change_pangle:

	ld a, (angle)	# if angle < 135 then increment angle and keep sweeping
	cp 0x87
	jp c, next_pangle

	imager:		# if angle > 135 show profile
	ei		# allow mode to be changed via keypad interrupt
	call show_profile	# display profile of object swept
	di

	ld a, (mode)	# if mode changed go to main menu
	cp 0x03
	jp nz, main_menu

	jp imager


next_pangle:

	call increase_angle # if angle < 135, increment angle and then rotate servo to
that angle
	call rotate_to_angle

jp profiler_loop




keypad_int:		# interrupt routine if key on keypad pressed, changes mode
according to key pressed
	in a, (keypad_in)

	ld a, 0x00	# initially set mode as 0 (menu-mode)
	ld (mode), a

	in a, (keypad_in)	# read data from keypad indicating which key pressed

	cpl		# converts keypad data into integers (1 for 1 pressed, 2
for 2 pressed, 3 for 3 pressed)
	sub 0x0F

	cp 0x00		# if key pressed is between 1 and 3, set mode to key,
otherwise leave mode as 1
	jp z, end_keypad_int
	cp 0x04
	jp nc, end_keypad_int

	ld (mode), a

end_keypad_int:
```

```
      ei
reti


clear:                    # clears LCD
      ld a, 0x01
      out (0xB8), a       # sends command to LCD control to clear LCD
      call delay_servo_short    # allows enough delay to clear screen
ret


make_sound:         # uses speaker to make sound indicating current distance reading
      # puts in (ADC-data or 0x10) into reload registers so that frequency is
indicative of distance. Or 0x10 is done such that a too small ADC reading does not
give a very high frequency which may cause the program to behave abnormally due to
too many clock interrupts being generated
      ld a, 0x00
      out0 (rr_high1), a
      in a, (ADC)
      or 0x10
      out0 (rr_low1), a

      ei
      call enable_speaker_timer # speaker enabled
      call delay_speaker        # give speaker enough time to make an audible sound
      di
      call disable_speaker_timer# speaker disabled
ret


delay_speaker:      # delay procedure to give speaker enough time to make an audible
sound
      push bc
      ld bc, 0x0600# do 600h nops
delay_speaker_loop:
      nop
      dec bc
      ld a, 0x00
      cp b
      jp nz, delay_speaker_loop
      pop bc
ret


servo_timer_toggle: # when servo clock interrupt occurs this toggles the servo-input
to produce required wave

      in0 a, (TCR) # required to be read to put down interrupt
      in0 a, (TDR0)

      #toggle servo bit to create pulse
      in a, (Y)
      xor 0x80
      out (Y), a

      ld a, d             # d,e hold rr_high0 and rr_low0
      cpl
      ld d, a

      ld a, e
```

```
	cpl
	ld e, a			# -de found in 1's complement form

	push hl

	ld h, 0x06
	ld l, 0x01

	add hl, de		# next pulse should have period 600h-de, since -de is in
1C form 1 is added, so 601h-de performed
	ld d, h
	ld e, l

	pop hl

	out0 (rr_low0), e	# new reload values put in to generate wave
	out0 (rr_high0), d

	ei
reti


speaker_timer_toggle:	# when speaker clock interrupt occurs this toggles the
speaker-input to produce required wave
	in0 a, (TCR) # required to be read to put down interrupt
	in0 a, (TDR1)

	in a, (Y)		# toggle speaker input to produce square wave
	xor 0x40
	out (Y), a
reti


beam_off:			# turns oscilloscope beam off by setting portB bit7 to 1
leaving other bits unchanged
	in a, (X)
	or 0x80
	out (X), a
ret

beam_on:			# turns oscilloscope beam on by setting portB bit7 to 0
leaving other bits unchanged
	in a, (X)
	and 0x7F
	out (X), a
ret


plotoutline: # plots outline indicating extreme x & y co-ordinates and sweep-able
region

	ld a, 0x00		# plot bottom leftmost point
	ld (x_coord), a
	ld (y_coord), a
	call plot_point

	ld a, 0x7F		# plot top rightmost point
	ld (x_coord), a
	ld a, 0x3F
	ld (y_coord), a
	call plot_point
```

```
      ld a, 0x00          # plots top leftmost point
      ld (x_coord), a
      ld a, 0x3F
      ld (y_coord), a
      call plot_point

      ld a, 0x7F          # plot bottom rightmost point
      ld (x_coord), a
      ld a, 0x00
      ld (y_coord), a
      call plot_point

      ld a, (angle)       # temporarily stores current angle in tangle since angle
is used by procedures
      ld (tangle), a


      ld a, 0x00
      ld (radius), a
      ld a, 0x2B
      ld (angle), a

      ld a, (mode)        # two lines at 45d and 135d are drawn to indicate sweep-
able areas for profiler and radar mode, but these are skipped for the tracker
      cp 0x02
      jp z, pointer

      ld b, 0x00
      ld c, 0x40

      downline:           # draws a line at 45degrees (grad=-1) from (0,3F) to
(40,0)
      ld a, b
      cpl
      or 0x80             # turn beam off
      out (X), a
      ld a, c
      cpl
      and 0x3F            # make sure servo and speaker bit unaffected
      out (Y), a
      call beam_on
      call beam_off
      inc b
      dec c
      ld a, c
      cp 0x00
      jp nz, downline

      upline:             # draws a line at 135degrees (grad=1) from (40,0) to
(7F,3F)
      ld a, b
      cpl
      or 0x80             # turn beam off
      out (X), a
      ld a, c
      cpl
      and 0x3F            # make sure servo and speaker bit unaffected
      out (Y), a
      call beam_on
      nop
```

```
        nop
        call beam_off
        inc b
        inc c
        ld a, c
        cp 0x3F
        jp nz, upline

        call beam_off


pointer:                    # draws a line of radius 10 at angle given

        ld a, 0x01          # initialise radius at 1
        ld (radius), a

        linept:
        ld a, (radius)
        call get_coords     # get oscilloscope co-ordinates for (angle, radius)
        call plot_point     # plot point on oscilloscope
        ld a, (radius)
        inc a
        ld (radius), a
        cp 0x0B      # keep plotting points at (angle, radius) until radius > 10
        jp nz, linept
ret

show_profile:               # displays profile of object scanned by profiler

        call plotoutline    # plots outline showing extreme points and sweep-able
region

        ld a, 0x2C          # start with angle at 45
        ld (angle), a

nextpoint:

        # get x-coordinate for point at angle using FA00 as start address and angle as
offset
        ld a, (angle)
        ld l, a
        ld h, 0xFA
        ld a, (hl)
        ld (x_coord), a

        # get y-coordinate for point at angle using F900 as start address and angle as
offset
        ld a, (angle)
        ld l, a
        ld h, 0xF9
        ld a, (hl)
        ld (y_coord), a

        ld b, a
        ld a, (x_coord)
        add a, b            # a = x_coord + y_coord

        cp 0x00
        jp z, endplot
```

```
	# if x-coord and y-coord are both not 0 (i.e. x-coord + y-coord not = 0) then
plot the point on oscilloscope, otherwise simply go to next angle

	call plot_point

	endplot:

	ld a, (angle)# angle = angle + 1
	inc a
	ld (angle), a

	ld a, (angle)# if angle < 136 then show point for next angle, otherwise entire
profile shown so exit
	sub 0x88

	jp c, nextpoint

ret


rotate_to_angle:    # loads timer0 with appropriate values for angle given and runs
it for long enough for a 1-3 degree rotation

	ld hl, reload_high_table  # load high register value into timer for angle from
the reload_high_table using angle as offset
	ld b, 0x00
	ld a, (angle)
	ld c, a
	add hl, bc
	ld a, (hl)
	ld d, a
	out0 (rr_high0), a

	ld hl, reload_low_table   # load low register value into timer for angle from
the reload_low_table using angle as offset
	ld b, 0x00
	ld a, (angle)
	ld c, a
	add hl, bc
	ld a, (hl)
	ld e, a
	out0 (rr_low0), a

	call enable_servo_timer          # enables timer0
	call delay_servo_short           # gives servo enough delay for 1-3 degree
rotation
	call disable_servo_timer         # disables timer0
ret



init_vector_table:
	ld hl, vector_table
	ld a, h
	ld i, a                          # setting up higher order vector byte
	ld a, l

	and 0xE0                         # mask out the unused data
	out0 (IVLR), a

	#enables interrupts that were previously disabled by default
```

```
	in0 a, (ITCR)
	or 0x04
	out0 (ITCR), a

	in0 a, (IVLR)
	or 0x04

	out0 (IVLR), a
ret


init_port:
	#initalise ports, A=input, B=output, C=output
	ld a, 0x90
	out0 (CSR), a

	#initialising portC such that all bits are zero, i.e timer bits are 0 and
oscilloscope Y-input is 000000
	ld a, 0x00
	out (Y), a

	#initialising portB such that all bits except bit7 are zero, i.e. oscilloscope
x-input is 0000000, and z-input = 1 (i.e. beam turned off)
	ld a, 0x80
	out (X), a

ret

init_angle45:	# sets angle to 45degrees and loads in appropriate reload values
into timer 0
	ld a, 0x2D
	ld (angle), a

	ld d, 0x01	# angle = 45
	ld e, 0x50

	out0 (rr_high0), d
	out0 (rr_low0), e
ret

init_angle0:	# sets angle to 0degrees and loads in appropriate reload values
into timer 0
	ld a, 0x00
	ld (angle), a

	ld d, 0x00	# angle = 0
	ld e, 0xD0

	out0 (rr_high0), d
	out0 (rr_low0), e

ret

enable_servo_timer: # turns timer0 (which is the servo motor input) on
	ei

	# make sure pulse starts on a low or else we would have an inverted wave to
what we desire
	in a, (Y)
	and 0x7F
```

```
	out (Y), a

	# turn timer0 on
	in0 a, (TCR)
	or 0x11
	out0 (TCR), a
ret

disable_servo_timer:	# turns timer0 (which is the servo motor input) off
	di
	# turn timer0 off
	in0 a, (TCR)
	and 0xEE
	out0 (TCR), a
ret

enable_speaker_timer:	# turns timer1 (which is the speaker input) on
	in0 a, (TCR)
	or 0x22
	out0 (TCR), a
ret

disable_speaker_timer:	# turns timer1 (which is the speaker output) on
	in0 a, (TCR)
	and 0xDD
	out0 (TCR), a
ret

delay_servo_full:	# gives servo enough delay for a full 180 or 90 degree
rotation
	push bc
	ld bc, 0xFFFF
full_delay_loop:	# do FFFFh nop's
	nop
	dec bc
	ld a, 0x00
	cp b
	jp nz, full_delay_loop
	pop bc
ret


get_distance:	# reads ADC value, converts it to distance

	ld l, 0x00
	ld h, 0x00
	ld b, 0x00

	in a, (ADC)  # reads ADC data, adds it to hl
	ld c, a
	cp 0x28	# if ADC value < 0x28 (distance >= 64) it means nothing was
detected
	jp c, nothingthere
	add hl, bc

	in a, (ADC) # reads ADC data, adds it to hl
	ld c, a
	cp 0x28
	jp c, nothingthere
	add hl, bc
```

```
	in a, (ADC)  # reads ADC data, adds it to hl
	ld c, a
	cp 0x28      # if ADC value < 0x28 (distance >= 64) it means nothing was
detected
	jp c, nothingthere
	add hl, bc

	in a, (ADC)  # reads ADC data, adds it to hl
	ld c, a
	cp 0x28      # if ADC value < 0x28 (distance >= 64) it means nothing was
detected
	jp c, nothingthere
	add hl, bc

	# hl contains sum of four ADC readings

	# using shifts such that a = hl/4, i.e. a = 2 LSB of h and 6 MSB of l =
average ADC value

	srl l
	srl l
	sla h
	sla h
	sla h
	sla h
	sla h

	ld a, l
	or h

	jp endget

	nothingthere:       # if nothing detected once set average ADC value to 0x00
	ld a, 0x00

	endget:             # use distance_table and average ADC value as offset to
get distance and put it in (distance)
	ld b, 0x00
	ld c, a
	ld hl, distance_table
	add hl, bc
	ld a, (hl)
	ld (distance), a
ret

display_angle:            # displays current angle on lcd

	ld a, (angle)
	ld (number), a

	call display_bin_to_dec   # displays angle in decimal

	ld a, 0xDF                # displays degree sign
	out (lcd_out), a
	call plotoutline

ret

display_angle_distance:         # displays angle, distance, cartesian co-ordinates

	call display_angle        # displays angle
```

```
ld a, distance            # if distance > 64 then do not display anything else
cp 0x40
jp nc, endofdisplay

ld a, 0x14                # displays space
out (lcd_out), a
call plotoutline

ld a, (distance)          # displays distance in decimal
ld (number), a
call display_bin_to_dec

ld a, 0x63                # displays "cm "
out (lcd_out), a
call delay_keypad

ld a, 0x6D
out (lcd_out), a
call delay_keypad

ld a, 0x14
out (lcd_out), a
call delay_keypad
ld a, 0x14
out (lcd_out), a
call delay_keypad

call get_coords

ld a, (x_coord)
cp 0x40
jp c, xpos

xneg:               # if x_coord < 64 then display 64 - x_coord (negative)
ld a, 0x2D          # display "-"
out (lcd_out), a
call delay_keypad

ld a, (x_coord)
cpl
inc a               # a = -x_coord (2C)
add 0x40            # a = -x_coord + 64
jp showx

xpos:               # if x_coord > 64 then display x_coord - 64

ld a, (x_coord)
sub 0x40            # a = x_coord - 64

showx:
ld (number), a      # display cartesian x_coord
call display_bin_to_dec

ld a, 0x2C          # display ","
out (lcd_out), a
call delay_keypad

showy:       # display y-coordinate
ld a, (y_coord)
ld (number), a
```

```
        call display_bin_to_dec
        call delay_keypad

endofdisplay:

ret


increase_angle3:
        ld a, (angle)       # angle = angle + 3
        add a, 0x03
        ld (angle), a
ret



decrease_angle3:
        ld a, (angle)       # angle = angle - 3
        sub 0x03
        ld (angle), a
ret



increase_angle:
        ld a, (angle)       # angle = angle + 1
        inc a
        ld (angle), a
ret


decrease_angle:
        ld a, (angle)       # angle = angle - 1
        dec a
        ld (angle), a
ret


delay_keypad:               # give enough delay to print character on lcd
        push bc
        ld bc, 0x00FF
delay_keypad_loop:  # do FFh no-ops
        nop
        dec bc
        ld a, 0x00
        cp b
        jp nz, delay_keypad_loop
        pop bc
ret


get_coords:                 # gets oscilloscope co-ordinates for angle,radius and puts
it in x_coord and y_coord

        ld a, 0x00
        ld h, a
        ld l, a

        # uses angle as offset to retrieve cosine value from cosine_table
        ld a, (angle)
        ld c, a
```

```
      ld hl, cosine_table
      ld b, 0x00
      add hl, bc
      ld a, (hl)

      ld h, a
      ld a, (radius)
      ld l, a

      mlt hl              # h = cosine(angle)*radius (number part), l =
cosine(angle)*radius (fraction part)

      ld a, (angle)# check if angle > 90
      cp 0x5A  #90
      jp nc, nonneg

      ld a, h             # if angle < 90 x_coord = 64-h
      cpl
      inc a
      ld h, a

      nonneg:             # if angle >= 90 x_coord = 64+h
      ld a, h
      add a, 0x40
      ld (x_coord), a

      # uses angle as offset to retrieve cosine value from sine_table
      ld a, (angle)
      ld c, a
      ld hl, sine_table
      ld b, 0x00
      add hl, bc
      ld a, (hl)

      ld h, a
      ld a, (radius)
      ld l, a
      mlt hl # h = sine(angle)*radius (number part), l = sine(angle)*radius
(fraction part)
      ld a, h
      ld (y_coord), a
ret


plot_point:         # plots point on oscilloscope given by co-ordinates x_coord and
y_coord

      ld a, (x_coord)     # sets lower 7 bits of portB (oscilloscope x-input) as
x_coord
      and 0x7F
      ld h, a
      in a, (X)
      and 0x80            # set bit7 (Z-input) to 1 (turn beam off)
      or h
      out (X), a

      ld a, (y_coord)     # sets lower 6 bits of portC (oscilloscope y-input) as
y_coord
      and 0x3F            # make sure bit6 and bit 7 set to 0
      ld h, a
```

```
      in a, (Y)
      and 0xC0            # make sure bit6 and bit7 (speaker and servo input)
unaffected
      or h
      and 0xBF
      out (Y), a

      call beam_on        # turn beam on for short time and then turn it off
      nop
      nop
      nop
      nop
      nop
      call beam_off
ret


display_bin_to_dec: # display binary number in decimal on lcd

      ld h, 0x00
      ld a, (number)

start100:                 # keep subtracting 100 from a and incrementing h until a <
0
      sub 0x64
      jp c, end100
      inc h
      ld (number), a
      jp start100

end100:                   # h stores hundreds digit
      ld a, h
      cp 0x00
      jp z, init10
      ld a, h             # display h as number (in ASCII, 0 is at 30, so e.g. 1 is
at 31) on lcd
      add a, 0x30
      out (lcd_out), a
      call plotoutline


init10:
      ld a, (number)
      ld h, 0x00

start10:                  # keep subtracting 10 from a and incrementing h until a <
0
      sub 0x0A
      jp c, end10
      inc h
      ld (number), a
      jp start10

end10:
      ld a, h             # h stores tens digit
      add a, 0x30
      out (lcd_out), a    # display h as number (in ASCII, 0 is at 30, so e.g. 1 is
at 31) on lcd
      call delay_keypad

      ld a, (number)
```

```
	ld h, a

	ld a, h			# h stores units digit
	add a, 0x30		# display h as number (in ASCII, 0 is at 30, so e.g. 1 is
at 31) on lcd
	out (lcd_out), a
	call delay_keypad
ret


display_character:  # displays ASCII character put in reg a onto lcd screen
	out (lcd_out), a
	call delay_keypad
	ld a, 0x20
ret

display_menu:
	# display 'SELECT MODE' and go to next line
	ld a, 0x53
	call display_character

	ld a, 0x45
	call display_character

	ld a, 0x4C
	call display_character

	ld a, 0x45
	call display_character

	ld a, 0x43
	call display_character

	ld a, 0x54
	call display_character

	ld a, 0x20
	call display_character

	ld a, 0x4D
	call display_character

	ld a, 0x4F
	call display_character

	ld a, 0x44
	call display_character

	ld a, 0x45
	call display_character

	ld a, 0x20
	call display_character
	call display_character
	call display_character
	call display_character
	call display_character
	call display_character
	call display_character
	call display_character
	call display_character
```

```
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character
call display_character

# display '1RAD 2TRK 3PRF'
ld a, 0x31
call display_character

ld a, 0x52
call display_character

ld a, 0x41
call display_character

ld a, 0x44
call display_character

ld a, 0x20
call display_character

ld a, 0x32
call display_character

ld a, 0x54
call display_character

ld a, 0x52
call display_character

ld a, 0x4B
call display_character

ld a, 0x20
call display_character

ld a, 0x33
call display_character

ld a, 0x50
call display_character

ld a, 0x52
call display_character
```

```
        ld a, 0x46
        call display_character
ret

display_radar_mode:
        # display 'RADAR MODE' and go to next line
        ld a, 0x52
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x44
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x20
        call display_character

        ld a, 0x4D
        call display_character

        ld a, 0x4F
        call display_character

        ld a, 0x44
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x20
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
```

```
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character

ret

display_tracker_seeking:
        # display 'TRACKER-SEEKING' and go to next line
        ld a, 0x54
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x43
        call display_character

        ld a, 0x4B
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x2D
        call display_character

        ld a, 0x53
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x4B
        call display_character

        ld a, 0x49
        call display_character

        ld a, 0x4E
        call display_character

        ld a, 0x47
        call display_character

        ld a, 0x20
        call display_character
        call display_character
        call display_character
        call display_character
```

```
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
ret

display_tracker_locked:
        # display 'TRACKER-LOCKED' and go to next line
        ld a, 0x54
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x43
        call display_character

        ld a, 0x4B
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x2D
        call display_character

        ld a, 0x4C
        call display_character

        ld a, 0x4F
        call display_character

        ld a, 0x43
        call display_character

        ld a, 0x4B
        call display_character
```

```
        ld a, 0x45
        call display_character

        ld a, 0x44
        call display_character

        ld a, 0x20
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
ret

display_profiler_mode:
        # display 'PROFILER MODE' and go to next line
        ld a, 0x50
        call display_character

        ld a, 0x52
        call display_character

        ld a, 0x4F
        call display_character

        ld a, 0x46
        call display_character

        ld a, 0x49
        call display_character

        ld a, 0x4C
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x52
        call display_character
```

```
        ld a, 0x20
        call display_character

        ld a, 0x4D
        call display_character

        ld a, 0x4F
        call display_character

        ld a, 0x44
        call display_character

        ld a, 0x45
        call display_character

        ld a, 0x20
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
ret

display_rotating:
        # display 'ROTATING BACK' and go to next line
        ld a, 0x52
        call display_character

        ld a, 0x4F
        call display_character

        ld a, 0x54
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x54
        call display_character
```

```
        ld a, 0x49
        call display_character

        ld a, 0x4E
        call display_character

        ld a, 0x47
        call display_character

        ld a, 0x20
        call display_character

        ld a, 0x42
        call display_character

        ld a, 0x41
        call display_character

        ld a, 0x43
        call display_character

        ld a, 0x4B
        call display_character
ret

display_object:
        # display 'OBJECT DETECTED' and go to next line
        ld a, 'O'
        call display_character

        ld a, 'B'
        call display_character

        ld a, 'J'
        call display_character

        ld a, 'E'
        call display_character

        ld a, 'C'
        call display_character

        ld a, 'T'
        call display_character

        ld a, ' '
        call display_character

        ld a, 'D'
        call display_character

        ld a, 'E'
        call display_character

        ld a, 'T'
        call display_character

        ld a, 'E'
        call display_character
```

```
        ld a, 'C'
        call display_character

        ld a, 'T'
        call display_character

        ld a, 'E'
        call display_character

        ld a, 'D'
        call display_character

        ld a, ' '
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
        call display_character
ret


sine_table:         # table storing sine values for each angle, using angle as offset
maps angles to sine values in fixed point binary fractions (binary point before MSB)
.byte 0x00, 0x04, 0x08, 0x0D, 0x11, 0x16, 0x1A, 0x1F, 0x23, 0x28, 0x2C, 0x30, 0x35,
0x39, 0x3D, 0x42
.byte 0x46, 0x4A, 0x4F, 0x53, 0x57, 0x5B, 0x5F, 0x64, 0x68, 0x6C, 0x70, 0x74, 0x78,
0x7C, 0x80, 0x83
.byte 0x87, 0x8B, 0x8F, 0x92, 0x96, 0x9A, 0x9D, 0xA1, 0xA4, 0xA7, 0xAB, 0xAE, 0xB1,
0xB5, 0xB8, 0xBB
.byte 0xBE, 0xC1, 0xC4, 0xC6, 0xC9, 0xCC, 0xCF, 0xD1, 0xD4, 0xD6, 0xD9, 0xDB, 0xDD,
0xDF, 0xE2, 0xE4
.byte 0xE6, 0xE8, 0xE9, 0xEB, 0xED, 0xEF, 0xF0, 0xF2, 0xF3, 0xF4, 0xF6, 0xF7, 0xF8,
0xF9, 0xFA, 0xFB
.byte 0xFC, 0xFC, 0xFD, 0xFE, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF
.byte 0xFE, 0xFE, 0xFD, 0xFC, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF4, 0xF3,
0xF2, 0xF0, 0xEE
.byte 0xED, 0xEB, 0xE9, 0xE7, 0xE6, 0xE4, 0xE2, 0xDF, 0xDD, 0xDB, 0xD9, 0xD6, 0xD4,
0xD1, 0xCF, 0xCC
```

```
.byte 0xC9, 0xC6, 0xC4, 0xC1, 0xBE, 0xBB, 0xB8, 0xB4, 0xB1, 0xAE, 0xAB, 0xA7, 0xA4,
0xA1, 0x9D, 0x99
.byte 0x96, 0x92, 0x8F, 0x8B, 0x87, 0x83, 0x7F, 0x7C, 0x78, 0x74, 0x70, 0x6C, 0x68,
0x63, 0x5F, 0x5B
.byte 0x57, 0x53, 0x4F, 0x4A, 0x46, 0x42, 0x3D, 0x39, 0x35, 0x30, 0x2C, 0x27, 0x23,
0x1F, 0x1A, 0x16
.byte 0x11, 0x0D, 0x08, 0x04, 0x00

cosine_table:       # table storing cosine values for each angle, using angle as
offset maps angles to cosine values in fixed point binary fractions (binary point
before MSB)
.byte 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFE, 0xFD, 0xFC, 0xFC, 0xFB, 0xFA,
0xF9, 0xF8, 0xF7
.byte 0xF6, 0xF4, 0xF3, 0xF2, 0xF0, 0xEE, 0xED, 0xEB, 0xE9, 0xE8, 0xE6, 0xE4, 0xE2,
0xDF, 0xDD, 0xDB
.byte 0xD9, 0xD6, 0xD4, 0xD1, 0xCF, 0xCC, 0xC9, 0xC6, 0xC4, 0xC1, 0xBE, 0xBB, 0xB8,
0xB5, 0xB1, 0xAE
.byte 0xAB, 0xA7, 0xA4, 0xA1, 0x9D, 0x9A, 0x96, 0x92, 0x8F, 0x8B, 0x87, 0x83, 0x7F,
0x7C, 0x78, 0x74
.byte 0x70, 0x6C, 0x68, 0x63, 0x5F, 0x5B, 0x57, 0x53, 0x4F, 0x4A, 0x46, 0x42, 0x3D,
0x39, 0x35, 0x30
.byte 0x2C, 0x28, 0x23, 0x1F, 0x1A, 0x16, 0x11, 0x0D, 0x08, 0x04, 0x00, 0x04, 0x08,
0x0D, 0x11, 0x16
.byte 0x1A, 0x1F, 0x23, 0x28, 0x2C, 0x30, 0x35, 0x39, 0x3D, 0x42, 0x46, 0x4A, 0x4F,
0x53, 0x57, 0x5B
.byte 0x5F, 0x64, 0x68, 0x6C, 0x70, 0x74, 0x78, 0x7C, 0x80, 0x83, 0x87, 0x8B, 0x8F,
0x92, 0x96, 0x9A
.byte 0x9D, 0xA1, 0xA4, 0xA8, 0xAB, 0xAE, 0xB1, 0xB5, 0xB8, 0xBB, 0xBE, 0xC1, 0xC4,
0xC7, 0xC9, 0xCC
.byte 0xCF, 0xD1, 0xD4, 0xD6, 0xD9, 0xDB, 0xDD, 0xDF, 0xE2, 0xE4, 0xE6, 0xE8, 0xE9,
0xEB, 0xED, 0xEF
.byte 0xF0, 0xF2, 0xF3, 0xF4, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFC, 0xFD,
0xFE, 0xFE, 0xFF
.byte 0xFF, 0xFF, 0xFF, 0xFF, 0xFF

distance_table: # table storing distance values for each ADC reading, using ADC
reading as offset maps ADC readings to distances (in cm)
.byte 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40, 0x40, 0x40
.byte 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40, 0x40, 0x40
.byte 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x3F, 0x3E, 0x3D, 0x3C, 0x3B, 0x3A, 0x39,
0x38, 0x37, 0x36
.byte 0x35, 0x34, 0x33, 0x32, 0x31, 0x30, 0x2F, 0x2E, 0x2E, 0x2D, 0x2C, 0x2C, 0x2B,
0x2A, 0x2A, 0x28
.byte 0x27, 0x26, 0x25, 0x25, 0x24, 0x24, 0x23, 0x23, 0x22, 0x22, 0x21, 0x21, 0x20,
0x20, 0x1F, 0x1F
.byte 0x1F, 0x1E, 0x1E, 0x1E, 0x1D, 0x1D, 0x1C, 0x1C, 0x1C, 0x1B, 0x1B, 0x1B, 0x1A,
0x1A, 0x19, 0x19
.byte 0x19, 0x18, 0x18, 0x18, 0x18, 0x17, 0x17, 0x17, 0x17, 0x16, 0x16, 0x16, 0x16,
0x15, 0x15, 0x15
.byte 0x15, 0x14, 0x14, 0x14, 0x14, 0x14, 0x13, 0x13, 0x13, 0x13, 0x13, 0x13, 0x13,
0x12, 0x12, 0x12
.byte 0x12, 0x12, 0x12, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x10, 0x10, 0x10, 0x10,
0x10, 0x10, 0x10
.byte 0x10, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0E, 0x0E, 0x0E,
0x0E, 0x0E, 0x0E
.byte 0x0E, 0x0E, 0x0E, 0x0E, 0x0E, 0x0D, 0x0D, 0x0D, 0x0D, 0x0D, 0x0D, 0x0D, 0x0D,
0x0D, 0x0D, 0x0D
.byte 0x0D, 0x0D, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C,
0x0C, 0x0C, 0x0C
```

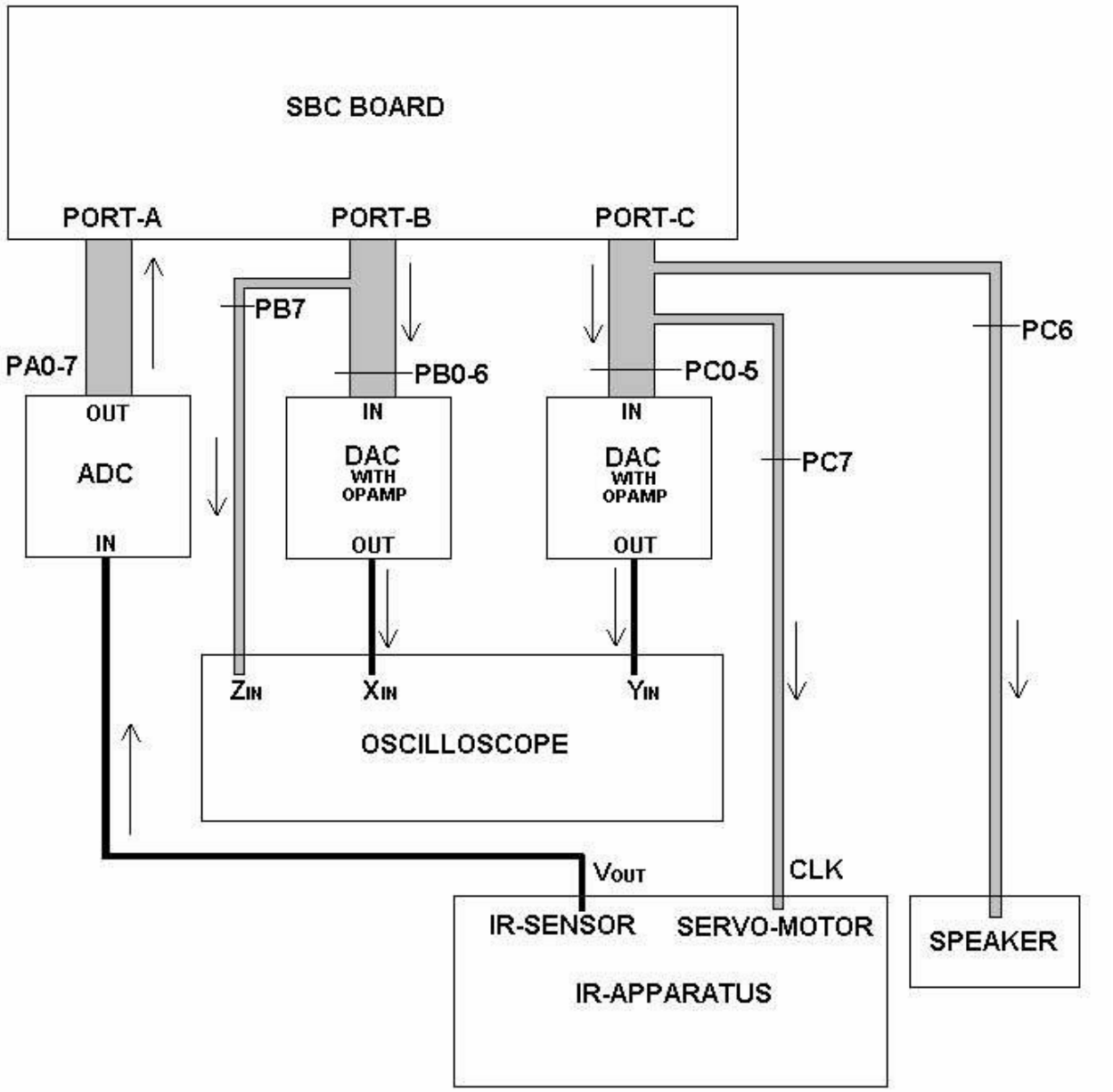
```
.byte 0x0C, 0x0C, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A,
0x0A, 0x0A, 0x0A
.byte 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A,
0x0A, 0x0A, 0x0A
.byte 0x0A, 0x0A, 0x0A, 0x0A, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09,
0x09, 0x09, 0x09
.byte 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40, 0x40, 0x40


reload_low_table:  # table mapping each angle to the required low reload register
value to be put into timer0 to rotate the servo to that angle
.byte 0xD0, 0xD2, 0xD5, 0xD8, 0xDB, 0xDE, 0xE1, 0xE3, 0xE6, 0xE9, 0xEC, 0xEF, 0xF2,
0xF4, 0xF7, 0xFA
.byte 0xFD, 0x00, 0x03, 0x06, 0x08, 0x0B, 0x0E, 0x11, 0x14, 0x17, 0x19, 0x1C, 0x1F,
0x22, 0x25, 0x28
.byte 0x2B, 0x2D, 0x30, 0x33, 0x36, 0x39, 0x3C, 0x3E, 0x41, 0x44, 0x47, 0x4A, 0x4D,
0x50, 0x52, 0x55
.byte 0x58, 0x5B, 0x5E, 0x61, 0x63, 0x66, 0x69, 0x6C, 0x6F, 0x72, 0x74, 0x77, 0x7A,
0x7D, 0x80, 0x83
.byte 0x86, 0x88, 0x8B, 0x8E, 0x91, 0x94, 0x97, 0x99, 0x9C, 0x9F, 0xA2, 0xA5, 0xA8,
0xAB, 0xAD, 0xB0
.byte 0xB3, 0xB6, 0xB9, 0xBC, 0xBE, 0xC1, 0xC4, 0xC7, 0xCA, 0xCD, 0xD0, 0xD2, 0xD5,
0xD8, 0xDB, 0xDE
.byte 0xE1, 0xE3, 0xE6, 0xE9, 0xEC, 0xEF, 0xF2, 0xF4, 0xF7, 0xFA, 0xFD, 0x00, 0x03,
0x06, 0x08, 0x0B
.byte 0x0E, 0x11, 0x14, 0x17, 0x19, 0x1C, 0x1F, 0x22, 0x25, 0x28, 0x2B, 0x2D, 0x30,
0x33, 0x36, 0x39
.byte 0x3C, 0x3E, 0x41, 0x44, 0x47, 0x4A, 0x4D, 0x50, 0x52, 0x55, 0x58, 0x5B, 0x5E,
0x61, 0x63, 0x66
.byte 0x69, 0x6C, 0x6F, 0x72, 0x74, 0x77, 0x7A, 0x7D, 0x80, 0x83, 0x86, 0x88, 0x8B,
0x8E, 0x91, 0x94
.byte 0x97, 0x99, 0x9C, 0x9F, 0xA2, 0xA5, 0xA8, 0xAB, 0xAD, 0xB0, 0xB3, 0xB6, 0xB9,
0xBC, 0xBE, 0xC1
.byte 0xC4, 0xC7, 0xCA, 0xCD, 0xD0

reload_high_table: # table mapping each angle to the required high reload register
value to be put into timer0 to rotate the servo to that angle
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
.byte 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01
.byte 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01
.byte 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01
.byte 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01
.byte 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01
.byte 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x02, 0x02,
0x02, 0x02, 0x02
.byte 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02, 0x02, 0x02
.byte 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02, 0x02, 0x02
.byte 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02, 0x02, 0x02
.byte 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02, 0x02, 0x02
.byte 0x02, 0x02, 0x02, 0x02, 0x02
```

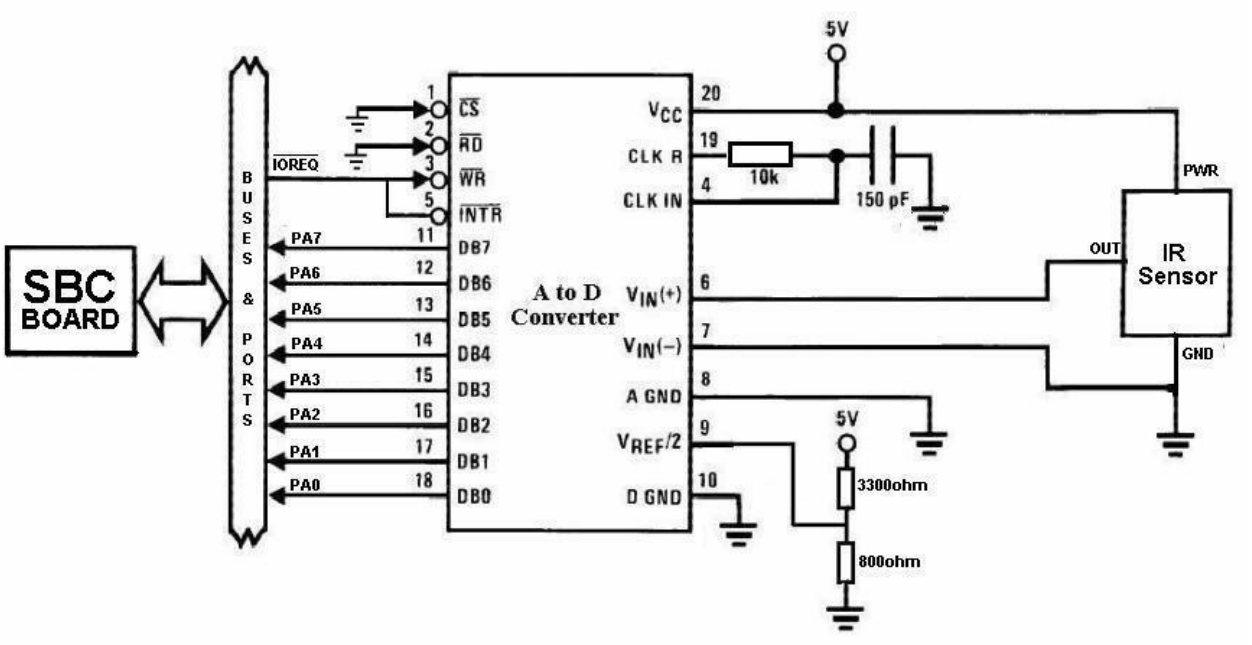
APPENDIX B – CIRCUIT DIAGRAMS

Block Diagram of Complete System



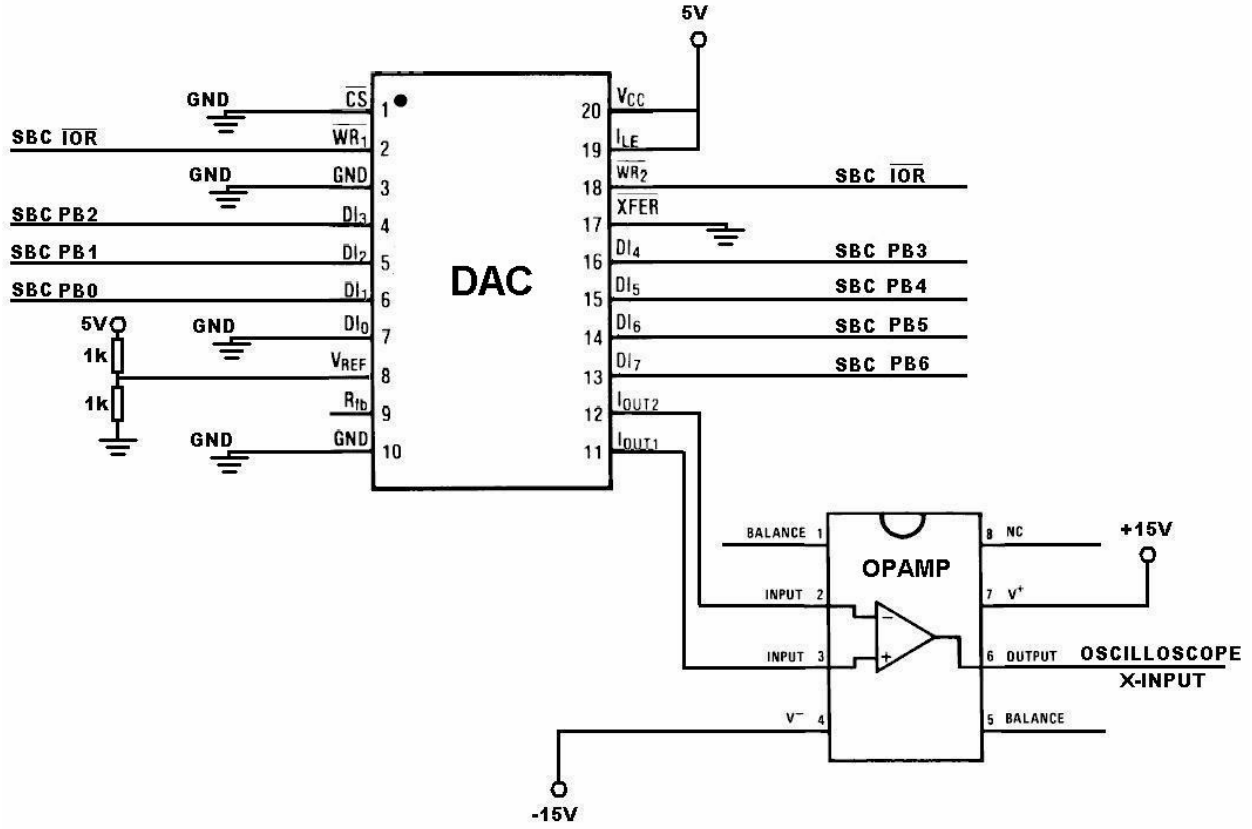
Please note:
Arrows indicate data flow
Shaded lines indicate digital data
Black lines indicate analogue data

IR Sensor and ADC Connections

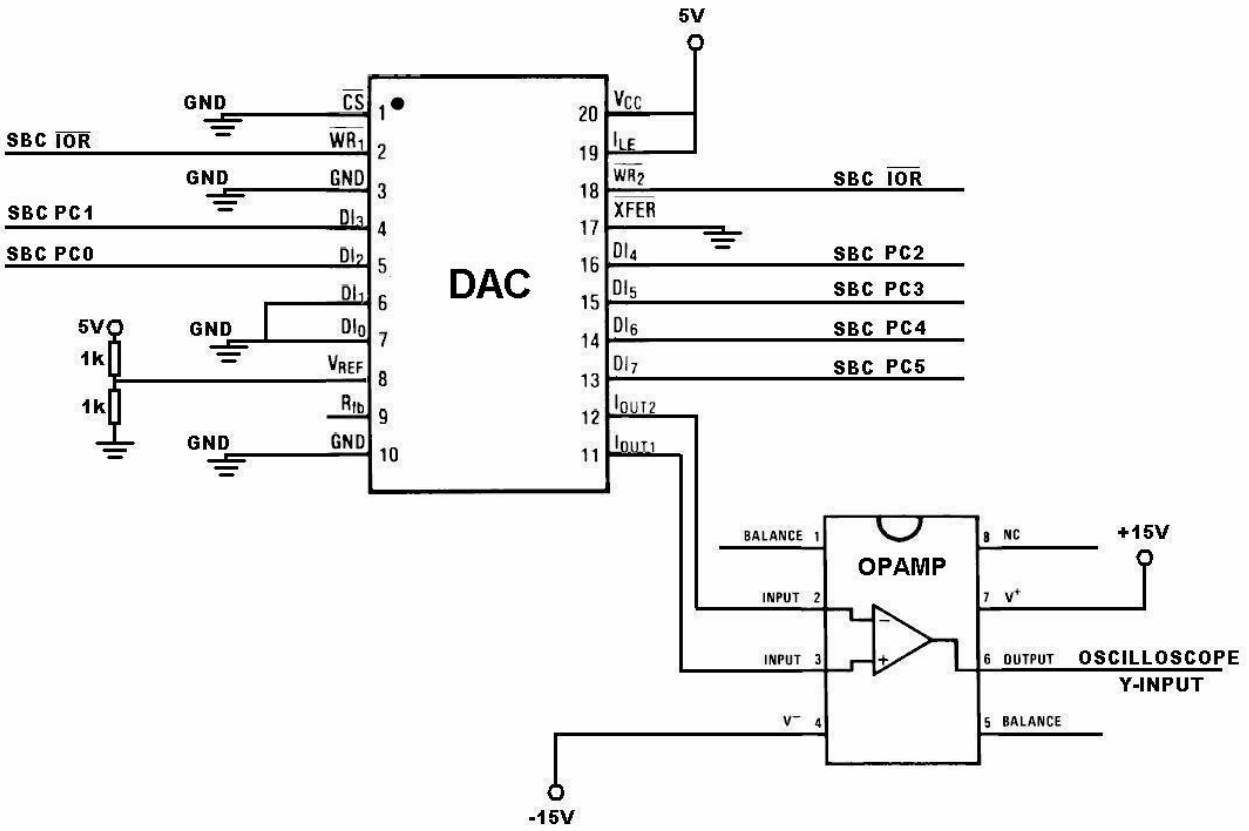


Chips Used: ADC0804LCN A/D Converter

DACs, OpAmps and Oscilloscope X & Y inputs

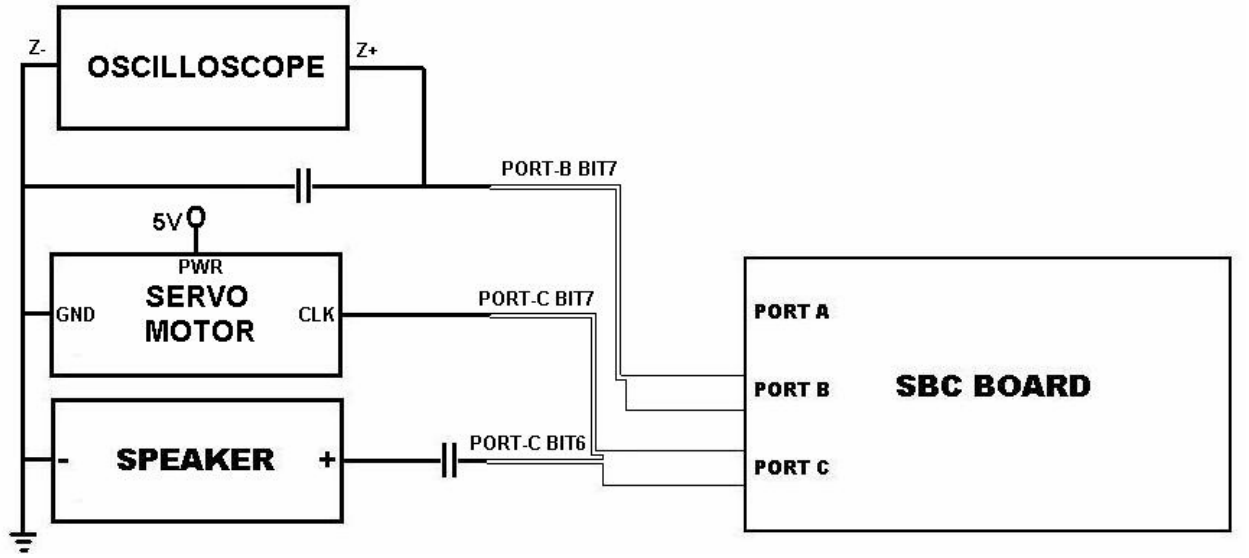


Chips Used: DAC0832LCN D/A Converter & LF356N Op Amp



Chips Used: DAC0832LCN D/A Converter & LF356N Op Amp

Servo Motor, Speaker and Oscilloscope Z-input



APPENDIX C – LOOK-UP TABLES

Reload High and Reload Low tables

The following shows the reload_high_table and reload_low_table tables which use the angle as offset to retrieve the corresponding reload value for each angle to put into the timer for rotating to that angle.

angle (offset)	angle (hex)	reload value (decimal)	reload value (hexadecimal)	**reload_high_ table entry**	**reload_low_ table entry**
000	00	208	00D0	00	D0
001	01	211	00D2	00	D2
002	02	214	00D5	00	D5
003	03	217	00D8	00	D8
004	04	219	00DB	00	DB
005	05	222	00DE	00	DE
006	06	225	00E1	00	E1
007	07	228	00E3	00	E3
008	08	231	00E6	00	E6
009	09	234	00E9	00	E9
010	0A	236	00EC	00	EC
011	0B	239	00EF	00	EF
012	0C	242	00F2	00	F2
013	0D	245	00F4	00	F4
014	0E	248	00F7	00	F7
015	0F	251	00FA	00	FA
016	10	254	00FD	00	FD
017	11	256	0100	01	00
018	12	259	0103	01	03
019	13	262	0106	01	06
020	14	265	0108	01	08
021	15	268	010B	01	0B
022	16	271	010E	01	0E
023	17	273	0111	01	11
024	18	276	0114	01	14
025	19	279	0117	01	17
026	1A	282	0119	01	19
027	1B	285	011C	01	1C
028	1C	288	011F	01	1F
029	1D	290	0122	01	22
030	1E	293	0125	01	25
031	1F	296	0128	01	28
032	20	299	012B	01	2B
033	21	302	012D	01	2D
034	22	305	0130	01	30
035	23	308	0133	01	33
036	24	310	0136	01	36

037	25	313	0139	01	39
038	26	316	013C	01	3C
039	27	319	013E	01	3E
040	28	322	0141	01	41
041	29	325	0144	01	44
042	2A	327	0147	01	47
043	2B	330	014A	01	4A
044	2C	333	014D	01	4D
045	2D	336	0150	01	50
046	2E	339	0152	01	52
047	2F	342	0155	01	55
048	30	345	0158	01	58
049	31	347	015B	01	5B
050	32	350	015E	01	5E
051	33	353	0161	01	61
052	34	356	0163	01	63
053	35	359	0166	01	66
054	36	362	0169	01	69
055	37	364	016C	01	6C
056	38	367	016F	01	6F
057	39	370	0172	01	72
058	3A	373	0174	01	74
059	3B	376	0177	01	77
060	3C	379	017A	01	7A
061	3D	382	017D	01	7D
062	3E	384	0180	01	80
063	3F	387	0183	01	83
064	40	390	0186	01	86
065	41	393	0188	01	88
066	42	396	018B	01	8B
067	43	399	018E	01	8E
068	44	401	0191	01	91
069	45	404	0194	01	94
070	46	407	0197	01	97
071	47	410	0199	01	99
072	48	413	019C	01	9C
073	49	416	019F	01	9F
074	4A	418	01A2	01	A2
075	4B	421	01A5	01	A5
076	4C	424	01A8	01	A8
077	4D	427	01AB	01	AB
078	4E	430	01AD	01	AD
079	4F	433	01B0	01	B0
080	50	436	01B3	01	B3
081	51	438	01B6	01	B6
082	52	441	01B9	01	B9

083	53	444	01BC	01	BC
084	54	447	01BE	01	BE
085	55	450	01C1	01	C1
086	56	453	01C4	01	C4
087	57	455	01C7	01	C7
088	58	458	01CA	01	CA
089	59	461	01CD	01	CD
090	5A	464	01D0	01	D0
091	5B	467	01D2	01	D2
092	5C	470	01D5	01	D5
093	5D	473	01D8	01	D8
094	5E	475	01DB	01	DB
095	5F	478	01DE	01	DE
096	60	481	01E1	01	E1
097	61	484	01E3	01	E3
098	62	487	01E6	01	E6
099	63	490	01E9	01	E9
100	64	492	01EC	01	EC
101	65	495	01EF	01	EF
102	66	498	01F2	01	F2
103	67	501	01F4	01	F4
104	68	504	01F7	01	F7
105	69	507	01FA	01	FA
106	6A	510	01FD	01	FD
107	6B	512	0200	02	00
108	6C	515	0203	02	03
109	6D	518	0206	02	06
110	6E	521	0208	02	08
111	6F	524	020B	02	0B
112	70	527	020E	02	0E
113	71	529	0211	02	11
114	72	532	0214	02	14
115	73	535	0217	02	17
116	74	538	0219	02	19
117	75	541	021C	02	1C
118	76	544	021F	02	1F
119	77	546	0222	02	22
120	78	549	0225	02	25
121	79	552	0228	02	28
122	7A	555	022B	02	2B
123	7B	558	022D	02	2D
124	7C	561	0230	02	30
125	7D	564	0233	02	33
126	7E	566	0236	02	36
127	7F	569	0239	02	39
128	80	572	023C	02	3C

129	81	575	023E	02	3E
130	82	578	0241	02	41
131	83	581	0244	02	44
132	84	583	0247	02	47
133	85	586	024A	02	4A
134	86	589	024D	02	4D
135	87	592	0250	02	50
136	88	595	0252	02	52
137	89	598	0255	02	55
138	8A	601	0258	02	58
139	8B	603	025B	02	5B
140	8C	606	025E	02	5E
141	8D	609	0261	02	61
142	8E	612	0263	02	63
143	8F	615	0266	02	66
144	90	618	0269	02	69
145	91	620	026C	02	6C
146	92	623	026F	02	6F
147	93	626	0272	02	72
148	94	629	0274	02	74
149	95	632	0277	02	77
150	96	635	027A	02	7A
151	97	638	027D	02	7D
152	98	640	0280	02	80
153	99	643	0283	02	83
154	9A	646	0286	02	86
155	9B	649	0288	02	88
156	9C	652	028B	02	8B
157	9D	655	028E	02	8E
158	9E	657	0291	02	91
159	9F	660	0294	02	94
160	A0	663	0297	02	97
161	A1	666	0299	02	99
162	A2	669	029C	02	9C
163	A3	672	029F	02	9F
164	A4	674	02A2	02	A2
165	A5	677	02A5	02	A5
166	A6	680	02A8	02	A8
167	A7	683	02AB	02	AB
168	A8	686	02AD	02	AD
169	A9	689	02B0	02	B0
170	AA	692	02B3	02	B3
171	AB	694	02B6	02	B6
172	AC	697	02B9	02	B9
173	AD	700	02BC	02	BC
174	AE	703	02BE	02	BE

175	AF	706	02C1	02	C1
176	B0	709	02C4	02	C4
177	B1	711	02C7	02	C7
178	B2	714	02CA	02	CA
179	B3	717	02CD	02	CD
180	B4	720	02D0	02	D0

Sine and Cosine tables

The sine and cosine tables map angles to sine and (absolute) cosines values respectively

angle	sine (angle)	**sine table entry**	cosine (angle)	**cosine table entry**
000	0.00	00	1.00	FF
001	0.02	04	1.00	FF
002	0.03	08	1.00	FF
003	0.05	0D	1.00	FF
004	0.07	11	1.00	FF
005	0.09	16	1.00	FF
006	0.10	1A	0.99	FE
007	0.12	1F	0.99	FE
008	0.14	23	0.99	FD
009	0.16	28	0.99	FC
010	0.17	2C	0.98	FC
011	0.19	30	0.98	FB
012	0.21	35	0.98	FA
013	0.22	39	0.97	F9
014	0.24	3D	0.97	F8
015	0.26	42	0.97	F7
016	0.28	46	0.96	F6
017	0.29	4A	0.96	F4
018	0.31	4F	0.95	F3
019	0.33	53	0.95	F2
020	0.34	57	0.94	F0
021	0.36	5B	0.93	EE
022	0.37	5F	0.93	ED
023	0.39	64	0.92	EB
024	0.41	68	0.91	E9
025	0.42	6C	0.91	E8
026	0.44	70	0.90	E6
027	0.45	74	0.89	E4
028	0.47	78	0.88	E2
029	0.48	7C	0.87	DF
030	0.50	80	0.87	DD
031	0.52	83	0.86	DB
032	0.53	87	0.85	D9
033	0.54	8B	0.84	D6
034	0.56	8F	0.83	D4
035	0.57	92	0.82	D1

angle	sine (angle)	**sine table entry**	cosine (angle)	**cosine table entry**
036	0.59	96	0.81	CF
037	0.60	9A	0.80	CC
038	0.62	9D	0.79	C9
039	0.63	A1	0.78	C6
040	0.64	A4	0.77	C4
041	0.66	A7	0.75	C1
042	0.67	AB	0.74	BE
043	0.68	AE	0.73	BB
044	0.69	B1	0.72	B8
045	0.71	B5	0.71	B5
046	0.72	B8	0.69	B1
047	0.73	BB	0.68	AE
048	0.74	BE	0.67	AB
049	0.75	C1	0.66	A7
050	0.77	C4	0.64	A4
051	0.78	C6	0.63	A1
052	0.79	C9	0.62	9D
053	0.80	CC	0.60	9A
054	0.81	CF	0.59	96
055	0.82	D1	0.57	92
056	0.83	D4	0.56	8F
057	0.84	D6	0.54	8B
058	0.85	D9	0.53	87
059	0.86	DB	0.51	83
060	0.87	DD	0.50	7F
061	0.87	DF	0.48	7C
062	0.88	E2	0.47	78
063	0.89	E4	0.45	74
064	0.90	E6	0.44	70
065	0.91	E8	0.42	6C
066	0.91	E9	0.41	68
067	0.92	EB	0.39	63
068	0.93	ED	0.37	5F
069	0.93	EF	0.36	5B
070	0.94	F0	0.34	57
071	0.95	F2	0.33	53

072	0.95	F3	0.31	4F
073	0.96	F4	0.29	4A
074	0.96	F6	0.28	46
075	0.97	F7	0.26	42
076	0.97	F8	0.24	3D
077	0.97	F9	0.22	39
078	0.98	FA	0.21	35
079	0.98	FB	0.19	30
080	0.98	FC	0.17	2C
081	0.99	FC	0.16	28
082	0.99	FD	0.14	23
083	0.99	FE	0.12	1F
084	0.99	FE	0.10	1A
085	1.00	FF	0.09	16
086	1.00	FF	0.07	11
087	1.00	FF	0.05	0D
088	1.00	FF	0.03	08
089	1.00	FF	0.02	04
090	1.00	FF	0.00	00
091	1.00	FF	0.02	04
092	1.00	FF	0.04	08
093	1.00	FF	0.05	0D
094	1.00	FF	0.07	11
095	1.00	FF	0.09	16
096	0.99	FE	0.10	1A
097	0.99	FE	0.12	1F
098	0.99	FD	0.14	23
099	0.99	FC	0.16	28
100	0.98	FC	0.17	2C
101	0.98	FB	0.19	30
102	0.98	FA	0.21	35
103	0.97	F9	0.23	39
104	0.97	F8	0.24	3D
105	0.97	F7	0.26	42
106	0.96	F6	0.28	46
107	0.96	F4	0.29	4A
108	0.95	F3	0.31	4F
109	0.95	F2	0.33	53
110	0.94	F0	0.34	57
111	0.93	EE	0.36	5B
112	0.93	ED	0.37	5F
113	0.92	EB	0.39	64
114	0.91	E9	0.41	68
115	0.91	E7	0.42	6C
116	0.90	E6	0.44	70
117	0.89	E4	0.45	74
118	0.88	E2	0.47	78
119	0.87	DF	0.49	7C
120	0.87	DD	0.50	80
121	0.86	DB	0.52	83
122	0.85	D9	0.53	87
123	0.84	D6	0.54	8B
124	0.83	D4	0.56	8F
125	0.82	D1	0.57	92
126	0.81	CF	0.59	96
127	0.80	CC	0.60	9A
128	0.79	C9	0.62	9D
129	0.78	C6	0.63	A1
130	0.77	C4	0.64	A4
131	0.75	C1	0.66	A8
132	0.74	BE	0.67	AB
133	0.73	BB	0.68	AE
134	0.72	B8	0.69	B1
135	0.71	B4	0.71	B5
136	0.69	B1	0.72	B8
137	0.68	AE	0.73	BB
138	0.67	AB	0.74	BE
139	0.66	A7	0.75	C1
140	0.64	A4	0.77	C4
141	0.63	A1	0.78	C7
142	0.62	9D	0.79	C9
143	0.60	99	0.80	CC
144	0.59	96	0.81	CF
145	0.57	92	0.82	D1
146	0.56	8F	0.83	D4
147	0.54	8B	0.84	D6
148	0.53	87	0.85	D9
149	0.51	83	0.86	DB
150	0.50	7F	0.87	DD
151	0.48	7C	0.87	DF
152	0.47	78	0.88	E2
153	0.45	74	0.89	E4
154	0.44	70	0.90	E6
155	0.42	6C	0.91	E8
156	0.41	68	0.91	E9
157	0.39	63	0.92	EB
158	0.37	5F	0.93	ED
159	0.36	5B	0.93	EF
160	0.34	57	0.94	F0
161	0.33	53	0.95	F2
162	0.31	4F	0.95	F3
163	0.29	4A	0.96	F4
164	0.28	46	0.96	F6
165	0.26	42	0.97	F7
166	0.24	3D	0.97	F8
167	0.22	39	0.97	F9
168	0.21	35	0.98	FA
169	0.19	30	0.98	FB
170	0.17	2C	0.98	FC
171	0.16	27	0.99	FC
172	0.14	23	0.99	FD
173	0.12	1F	0.99	FE

174	0.10	1A	0.99	FE
175	0.09	16	1.00	FF
176	0.07	11	1.00	FF
177	0.05	0D	1.00	FF
178	0.03	08	1.00	FF
179	0.02	04	1.00	FF
180	0.00	00	1.00	FF

Distance Table

Maps ADC readings from IR sensor to distances of object from sensor, uses hex reading as offset

ADC Reading (decimal)	**Hex Reading**	Distance (cm)	**Table entry (hex)**
000	00	64	40
001	01	64	40
002	02	64	40
003	03	64	40
004	04	64	40
005	05	64	40
006	06	64	40
007	07	64	40
008	08	64	40
009	09	64	40
010	0A	64	40
011	0B	64	40
012	0C	64	40
013	0D	64	40
014	0E	64	40
015	0F	64	40
016	10	64	40
017	11	64	40
018	12	64	40
019	13	64	40
020	14	64	40
021	15	64	40
022	16	64	40
023	17	64	40
024	18	64	40
025	19	64	40
026	1A	64	40
027	1B	64	40
028	1C	64	40
029	1D	64	40
030	1E	64	40
031	1F	64	40
032	20	64	40
033	21	64	40
034	22	64	40
035	23	64	40
036	24	64	40
037	25	64	40
038	26	63	3F
039	27	62	3E
040	28	61	3D
041	29	60	3C
042	2A	59	3B
043	2B	58	3A
044	2C	57	39
045	2D	56	38
046	2E	55	37
047	2F	54	36
048	30	53	35
049	31	52	34
050	32	51	33
051	33	50	32
052	34	49	31
053	35	48	30
054	36	47	2F
055	37	46	2E
056	38	46	2E
057	39	45	2D
058	3A	44	2C
059	3B	44	2C
060	3C	43	2B
061	3D	42	2A
062	3E	42	2A
063	3F	40	28
064	40	39	27
065	41	38	26
066	42	37	25
067	43	37	25
068	44	36	24
069	45	36	24
070	46	35	23
071	47	35	23
072	48	34	22
073	49	34	22
074	4A	33	21
075	4B	33	21
076	4C	32	20
077	4D	32	20
078	4E	31	1F
079	4F	31	1F
080	50	31	1F
081	51	30	1E
082	52	30	1E
083	53	30	1E
084	54	29	1D
085	55	29	1D
086	56	28	1C
087	57	28	1C
088	58	28	1C
089	59	27	1B
090	5A	27	1B
091	5B	27	1B
092	5C	26	1A
093	5D	26	1A
094	5E	25	19
095	5F	25	19
096	60	25	19
097	61	24	18
098	62	24	18
099	63	24	18
100	64	24	18
101	65	23	17
102	66	23	17
103	67	23	17
104	68	23	17
105	69	22	16
106	6A	22	16
107	6B	22	16
108	6C	22	16
109	6D	21	15
110	6E	21	15
111	6F	21	15
112	70	21	15
113	71	20	14

ADC Reading (decimal)	**Hex Reading**	Distance (cm)	**Table entry (hex)**
114	72	20	14
115	73	20	14
116	74	20	14
117	75	20	14
118	76	19	13
119	77	19	13
120	78	19	13
121	79	19	13
122	7A	19	13
123	7B	19	13
124	7C	19	13
125	7D	18	12
126	7E	18	12
127	7F	18	12
128	80	18	12
129	81	18	12
130	82	18	12
131	83	17	11
132	84	17	11
133	85	17	11
134	86	17	11
135	87	17	11
136	88	17	11
137	89	16	10
138	8A	16	10
139	8B	16	10
140	8C	16	10
141	8D	16	10
142	8E	16	10
143	8F	16	10
144	90	16	10
145	91	15	0F
146	92	15	0F
147	93	15	0F
148	94	15	0F
149	95	15	0F
150	96	15	0F
151	97	15	0F
152	98	15	0F
153	99	15	0F
154	9A	14	0E
155	9B	14	0E
156	9C	14	0E
157	9D	14	0E
158	9E	14	0E
159	9F	14	0E
160	A0	14	0E

ADC Reading (decimal)	**Hex Reading**	Distance (cm)	**Table entry (hex)**
161	A1	14	0E
162	A2	14	0E
163	A3	14	0E
164	A4	14	0E
165	A5	13	0D
166	A6	13	0D
167	A7	13	0D
168	A8	13	0D
169	A9	13	0D
170	AA	13	0D
171	AB	13	0D
172	AC	13	0D
173	AD	13	0D
174	AE	13	0D
175	AF	13	0D
176	B0	13	0D
177	B1	13	0D
178	B2	12	0C
179	B3	12	0C
180	B4	12	0C
181	B5	12	0C
182	B6	12	0C
183	B7	12	0C
184	B8	12	0C
185	B9	12	0C
186	BA	12	0C
187	BB	12	0C
188	BC	12	0C
189	BD	12	0C
190	BE	12	0C
191	BF	12	0C
192	C0	12	0C
193	C1	12	0C
194	C2	11	0B
195	C3	11	0B
196	C4	11	0B
197	C5	11	0B
198	C6	11	0B
199	C7	11	0B
200	C8	10	0A
201	C9	10	0A
202	CA	10	0A
203	CB	10	0A
204	CC	10	0A
205	CD	10	0A
206	CE	10	0A
207	CF	10	0A
208	D0	10	0A
209	D1	10	0A

ADC Reading (decimal)	**Hex Reading**	Distance (cm)	**Table entry (hex)**
210	D2	10	0A
211	D3	10	0A
212	D4	10	0A
213	D5	10	0A
214	D6	10	0A
215	D7	10	0A
216	D8	10	0A
217	D9	10	0A
218	DA	10	0A
219	DB	10	0A
220	DC	10	0A
221	DD	10	0A
222	DE	10	0A
223	DF	10	0A
224	E0	10	0A
225	E1	10	0A
226	E2	10	0A
227	E3	10	0A
228	E4	09	09
229	E5	09	09
230	E6	09	09
231	E7	09	09
232	E8	09	09
233	E9	09	09
234	EA	09	09
235	EB	09	09
236	EC	09	09
237	ED	09	09
238	EE	09	09
239	EF	09	09
240	F0	09	09
241	F1	09	09
242	F2	09	09
243	F3	09	09
244	F4	09	09
245	F5	09	09
246	F6	09	09
247	F7	64	40
248	F8	64	40
249	F9	64	40
250	FA	64	40
251	FB	64	40
252	FC	64	40
253	FD	64	40
254	FE	64	40
255	FF	64	40

BIBLIOGRAPHY

- University of York Computer Science Department – Microcomputer Communications Project Module Website by Nick Pears http://www-course.cs.york.ac.uk/mcp
- DC Power Supply GPC-M Series Analogue Digital Type GW-Instek User Manual
- GOS – 6xxG Family Dual Trace Oscilloscope GW-Instek User Manual
- Zilog Z80 Family CPY User Manual UM0080020202
- The National Semiconductor Website www.national.com (for chip datasheets)
- Farnell InOne Website www.farnell.com (for chip costing)

Please note that some of the hardware diagrams in Appendix B are modified versions of those found in the National Semiconductor Website's datasheets for certain chips.