

Handheld Ultrasonic Rangefinder

Introduction

Our ultrasonic rangefinder is capable of allowing the user to determine his or her distance from an object or wall.

When deciding on what type of project to design and construct, we decided that we wanted to create something that would have some practical use in life. Many groups in the past created video games, but we wanted to be different. We considered issues such as safety, user interface, and ease of use, and came up with the idea of making an ultrasonic rangefinder. A rangefinder can be used in various applications such as a measuring device or an obstacle detection device.

Rational of the project

As stated earlier, we wanted to create something that would be of practical use to people of various walks of life. For instance, our rangefinder could be used in the case of a blackout where a person needs to find his way through a dark building and cannot see where the walls are. Alternatively, it could be used by surveyors that are preparing a site for construction.

One group from a previous year had done a project using an ultrasonic rangefinder, but their project involved a pre-manufactured system. The bulk of their project was using AM transmission to communicate with the rangefinder module.

Our project is different in the fact that we built the rangefinder from the ground up.

Logical structure

The basic principal of the rangefinder is based on the simple concepts of SONAR (*Sound Navigation and Ranging*). First, an ultrasonic pulse is transmitted by a transducer (a device that converts a voltage signal into a sound wave and vice versa). This pulse then reflects off an object and is received by another transducer. Using the known speed of sound (340.29 m/s) and the time between the transmitted pulse (the ping) and the received pulse (the pong), one can simply calculate the distance traveled by the pulse.

In addition to determining the distance between the device and an object using the known speed of sound, our initial design included two additional modes of operation. One mode was a calibration mode while the other was a speed detection mode.

Since the speed of sound varies with altitude and atmospheric conditions, a calibration mode is quite useful. Our program is designed such that if in calibrate mode, the user can place the device a known distance 5 centimeters and send a pulse. The device then uses the time necessary for the pulse to reflect off the object and calculates a new value for the speed of sound. This new value is then used for all future distance calculations until the device is powered off.

The original speed detection mode was used to indicate to the user how quickly he or she is moving towards or away from an object. Since speed is determined by the change in distance divided by the change in time (dx/dt), speed can be determined quite simply using two pulse transmissions. When initiated, a first pulse is sent and received by the device. The distance is calculated and stored by the device. A second pulse is then sent and received by the device 0.25 seconds later and the results stored by the device. The MCU can now take the difference in distance, divide that by 0.25 seconds, and display the speed to the user on the LCD. We had

initially considered using Doppler shifts in frequency to calculate speed, but we decided against it since measuring frequency is a completely new task we would have to program (compared to just detecting a pulse) and would have required much more complicated and CPU intensive math. We also felt that the velocity of a handheld device would not be fast enough to actually incur significant Doppler shifts that would make calculations doable.

Due to noise issues while testing our original program, the method of calculating distance was changed. We are now taking multiple distance samples and calculating the average. This enables more accurate readings. Obviously, the larger the number of samples, the higher the accuracy. Since we are now taking multiple samples, it is possible to measure the speed of the device in the same operating mode.

Replacing the original speed detection mode is a mode in which the user can change the number of samples to use for the distance calculation. The user can increase or decrease the number of samples by multiples of 50, with 50 being the minimum. Our calibration mode was still implemented.

Background math

The math required for our project is very simple. The speed of sound is 340.29 m/s at sea level. This equates to 340,290 millimeters per second. Taking the inverse of this, the time necessary for a sound wave to travel one millimeter is 2.9387 microseconds. By counting the time between the ping and pong, the distance traveled by the pulse can be determined. One has to keep in mind however, that the distance traveled is actually twice the distance between the device and the object (since the pulse travels in one direction, is reflected, and travels back in the other direction).

The speed calculation is simply the first derivative with respect to time of the position (dx/dt). Two pulses are sent and received at a known time interval and the difference in distances is used to determine the speed of the device relative to the object. More specifically, $(x_2 - x_1)/(t_2 - t_1) = \text{speed}$.

Hardware/software tradeoffs

Since the basic principle of a rangefinder is rather straight forward (send pulse, receive pulse, calculate distance based on time difference) we decided to stick with the Atmel Mega32 MCU that we have been using the entire semester. This MCU can easily handle the software needed for the rangefinder and there was no reason not to use it. Using a different MCU would require learning all the intricacies associated with it and was not practical.

Initially, we wanted to try an infrared or laser rangefinder. However, since we were going to use a Mega32 with a 16 MHz crystal, this wouldn't be possible. The speed of light is $3e8$ m/s, meaning the time to travel one meter is 3.33 nanoseconds. With a 16 MHz crystal, the time resolution of the Mega32 is 62.5 nanoseconds. This means that a rangefinder based on a light wave would be accurate to only 18.75 meters. Definitely not practical!

The alternative we chose was to use a 40 kHz ultrasound transducer pair. Ultrasonic transducers are relatively cheap (the ones we bought cost less than \$7 for the pair) and easy to acquire. At 40 kHz, the human user would not be able to hear the pulse (human hearing is limited to approximately 20 Hz to 20 kHz) so irritancy is not an issue.

As stated earlier, the time resolution of sound is only about 3 microseconds, so speed and fast execution time of code was not of great importance (compared with some other application like NTSC video signals). This made ultrasound an easy decision.

Another hardware tradeoff that was taken into account when designing and construction the rangefinder was a matter of gain. When testing the transducers with a signal generator, we noticed that the effective range was

proportional to the input voltage of the transmitting transducer. Simply put, the higher the input voltage, the farther its signal could travel. Since the transmitter was being fed a signal from an MCU port pin, it would only receive 5 volts. This resulted in a poor range using the signal generator while testing (a range of only a few centimeters). By increasing the voltage to around 20 volts using the signal generator, we were able to get a range in excess of 1.5 meters. The tradeoff here comes down to the power supply used when making the device completely portable. We could have used a single 9 volt battery, but the gain on the transducer would be less than 2 to 1. Using two 9 volt batteries in series (giving a total of 18 volts) would give us a gain of slightly more than 3 to 1 (18 to 5, more specifically). Unfortunately, this makes the device heavier and cost more to operate. Initially, we decided that increased range was more of a pro than the con of one more battery. Unfortunately, using 18 volts for the gain ended up not working as desired. More on this later.

The main software tradeoff was using dx/dt to calculate distance rather than using Doppler shifts. Doppler shifts would have required measuring the frequencies of the square waves as well as determining frequency shifts. This would have required code almost as difficult to write as our whole project so we just decided to measure speed using dx/dt from elementary physics.

Another tradeoff was the decision to use CodeVision C instead of assembly to program the MCU. Assembly most likely would have resulted in more accurate timing and thus distance measurements, however using C had the benefit of being more user friendly in regards to programming. Using C enabled us to save time while programming and allowed us to find bugs in the code more easily.

Design standards

Since our project is a ground-up portable device that does not interact with any other devices, there are no such design standards (IEEE, ISO, ANSI, etc.) that were considered when designing our rangefinder. Since the program for the MCU was written in the CodeVision C programming language, that could be considered the only standard used in the development of the rangefinder.

Intellectual property

Rangefinders are nothing new and can be purchased off the shelf for various applications. However, since we designed our rangefinder using only the basic principles of SONAR without using any existing product as a basis, we do not believe we are in violation of any existing patents, copyrights, or trademarks. We did, however, receive assistance from Prof. Land and the TAs in regards to circuit design and software structure, and for that we are appreciative.

Hardware design

The hardware for the rangefinder can be broken down into three functional units, the receiving circuit, the transmitting circuit, and the MCU circuit. The receiver and transmitter circuits can work independently of the MCU, which made testing with a signal generator quite useful. This allowed one of us to work on the hardware while the other worked on the software independently and not require one to be dependent on the other for testing.

At the heart of the receiver circuit is one of the ultrasonic transducers. The transducer converts an incoming sound wave and converts it into a voltage signal. This signal needs to be cleaned of noise, amplified, and turned into a TTL-type signal for the MCU. The signal from the transducer is fed through a capacitor to filter out noise and then through a voltage divider to center the signal at 2.5 volts. From here, the signal needs to be amplified to guarantee true TTL levels. Initially, this was attempted using an LM358 op-amp. However, the LM358 could not switch fast enough and was unable to create a clean square wave. At the suggestion of Prof. Land, we tried using a 74HC04 high speed CMOS inverter as a high gain linear amplifier. Initially, we used resistor values of 1

MΩ and 100 kΩ to achieve a gain of 10. This created voltage levels well beyond +5 volts and resulted in a clipped signal at the output. This, however, is actually desired because the output now represented a more of a square wave than a sine wave. The output of the inverter was then fed into a SN74HC14N Schmitt trigger to convert it into a true TTL square wave. The input of the Schmitt trigger was connected to ground using a 22 pF capacitor to clean the signal from noise. The output of the Schmitt trigger can now be fed into the MCU as the received pulse.

During testing, however, we noticed that this did not quite create a nice clean square wave and was influenced greatly by noise. To combat this, we changed the gain of the inverter amplifier by replacing the 100 kΩ resistor with a 10 kΩ resistor for a gain of 100. We then fed the signal through three cascaded Schmitt triggers to clean up the signal much more than previously. There is a single 22 pF capacitor on the input of the last Schmitt trigger to ground.

Unlike the receiver circuit which tries to generate a TTL square wave, the transmitter circuit starts with a TTL square wave. This square wave is generated by the MCU as a series of pulses. The goal of the transmitting circuit is to amplify the pulse train enough so that it can travel a significant distance and reflect back towards the receiver. Again, we had initially tried using a LM358 op-amp running into the same switching problems. We then tried using a 2N3904 BJT sourced by 9 volts at the collector as a simple switch. This ended up not working because the transducer did not create a DC path for current to flow through the BJT. In the end, a LM311 high speed comparator was used as the amplifier. In the spec sheet, an example of a zero-crossing detector was given. Changing that to a 2.5 volt-crossing detector was just a matter of changing the negative input terminal to 2.5 volts instead of GND and the VCC- pin to GND instead of -5 volts. By supplying the LM311's VCC+ pin with +18 volts, we were able to get high gain for the square wave generated by the MCU to transmit through the transducer.

Unfortunately, the high gain caused by the +18 volt supply also amplified noise in the circuit. The source of this noise is still unknown, but is likely due to the physical locations of the various circuits on the solder board. Many wires were crossing each other and are the likely culprit. As such, the amplified noise gave rise to incorrect readings by the receiver circuit. When we dropped the supply voltage back down to +9 volts, the range and accuracy of the device improved.

The Mega32 MCU is used as the brain of the rangefinder. Pins 4, 5, and 6 of PortB were used as the programming port. This allowed us to program the MCU using the STK500 development board without removing the MCU from the solder board. Pins 1, 2, and 3 of PortD were connected to SPST pushbuttons that are used to interface with the rangefinder. Depending on which mode the device is in, each button performs different functions. PortD.5 was used to supply the transmitter circuit with the TTL pulse train (ping), while PortD.7 was used as the input from the receiver circuit (pong). PortC was used for the LCD display. The LCD displays the calculated distance and speed to the user. PortA of the MCU was not used.

Power to the device was supplied by one 9 volt battery hooked up in series with a SPST pushbutton as the on/off switch. The +9 volts was fed into a LM340 voltage regulator to generate 5 volts for the MCU, ICs, and LCD. A 9 volt line was used to power the LM311 comparator to generate gain.

Acknowledgement must be paid towards Prof. Land and the TAs. Without their help, the hardware for the receiver and transmitter would have never been successful. Thanks.

Software design

The program for the rangefinder essentially does two things: 1) transmit a 40 kHz pulse, and 2) detect when the pulse is received. All that was needed to be done was to count the time that passed between the time of

transmission of the ping and the reception of the pong. There were also 3 modes of operation: normal mode, change sample number mode, and calibration mode.

In order to generate the ping and count the time between ping and pong, we used two of the MCU's timers. Timer1 was running at 40 kHz in PWM mode to generate a square wave for transmission using PortD.5. Timer0 was running at 250 kHz. Timer0's purpose is to calculate time between the send pulse and the receive pulse.

In normal operation, the user pushes a button to send out the ping. The MCU then counts the number of ticks it takes before the receiver detects the pong. The number of ticks is then divided by a factor of two to represent the number of millimeters between the device and the object. As stated earlier, multiple distance samples are taken and then averaged to create more accurate readings. We noticed during testing that using one sample (or only a few) gave erroneous measurements due to picking up random noise spikes. By increasing the number of samples to a minimum of 50, we obtained pretty accurate measurements. The average of all the measurements is then displayed on the LCD. In addition, normal mode also displays the speed of the device relative to a stationary object. The speed is determined by using the distance values of the middle sample and the last sample using the dx/dt formula mentioned earlier. The LCD displays the measured distance in millimeters and the speed in millimeters/second.

In the sample number changing mode you can change how many samples you want to take with a minimum number of samples being 50. This is to allow you to get a more precise measurement taking more measurements and eliminating randomness.

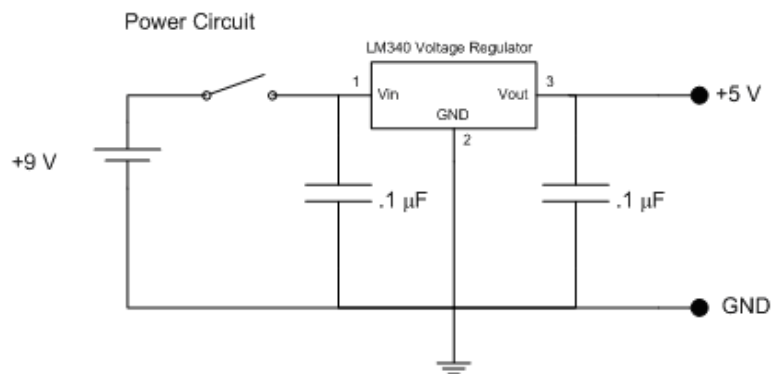
Calibration mode is used in cases when the speed of sound may differ from that at sea level. Factors such as altitude and humidity can affect the speed of sound. To maintain correct measurements, the user places the device exactly 5 centimeters from a flat wall or object. By pressing the pulse button, a pulse is sent out and received by the device. The MCU then uses the time between ping and pong to calculate the new number of ticks per meter. The MCU uses this value for all future measurements until the device is powered off.

The code was very straight forward in theory. The transducers that we used operate at 40 kHz. We first attempted to use a clock running at 80 kHz to toggle a port pin to generate a 40 kHz square wave. This did not work as well as expected. When we hooked up the port pin to the oscilloscope for testing, we noticed that the output was not an ideal 40 kHz square wave. To get around this we decided to use the PWM mode of Timer1. We configured the timer to toggle OC1A on compare match so we could pull a 40 kHz square wave off PortD.5. This seemed to solve the problem of producing a 40 kHz square wave from a port pin. To change the number of pulses in each pulse sequence for transmission we just turned on and off the toggle at certain times.

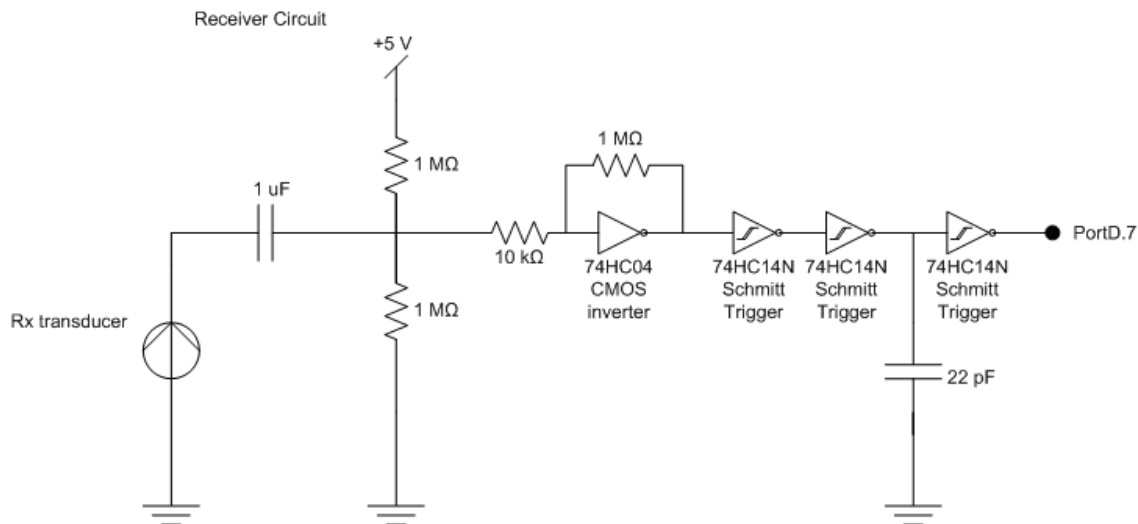
Another tricky part of the program was the timing and accuracy. As mentioned earlier, we programmed a timer tick to be 250 kHz. This translates roughly to 1.36 millimeters per second. To get the right distances required some floating point arithmetic. Finally knowing where to start and stop the counter clock was a bit tricky. We did not want to count any useless ticks that did not represent when the pulse was actually sent. These useless ticks are a result of the MCU performing other operations such as floating point calculation and conditionals. It was tricky to figure out where in the code to actually start the counter.

Schematics/pictures

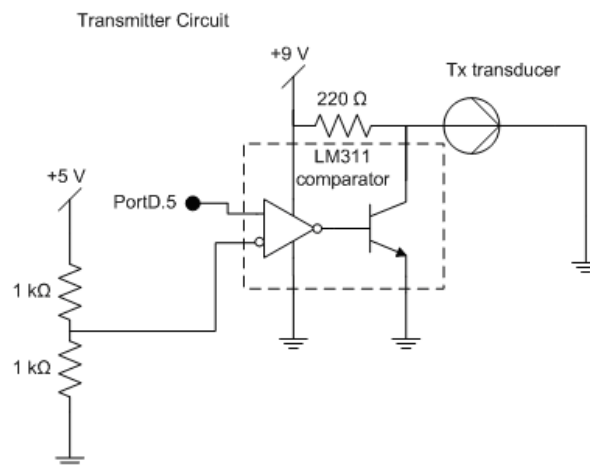
Power circuit



Receiver circuit



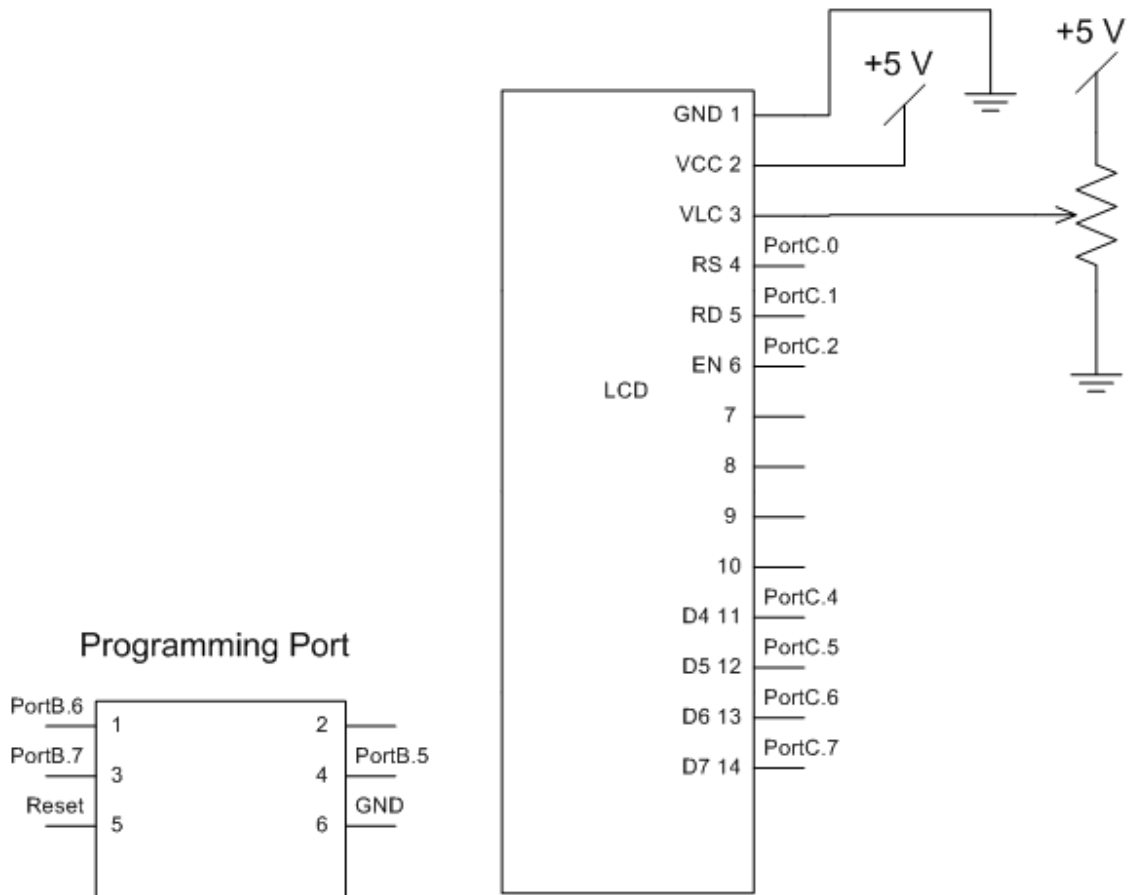
Transmitter circuit



LCD circuit

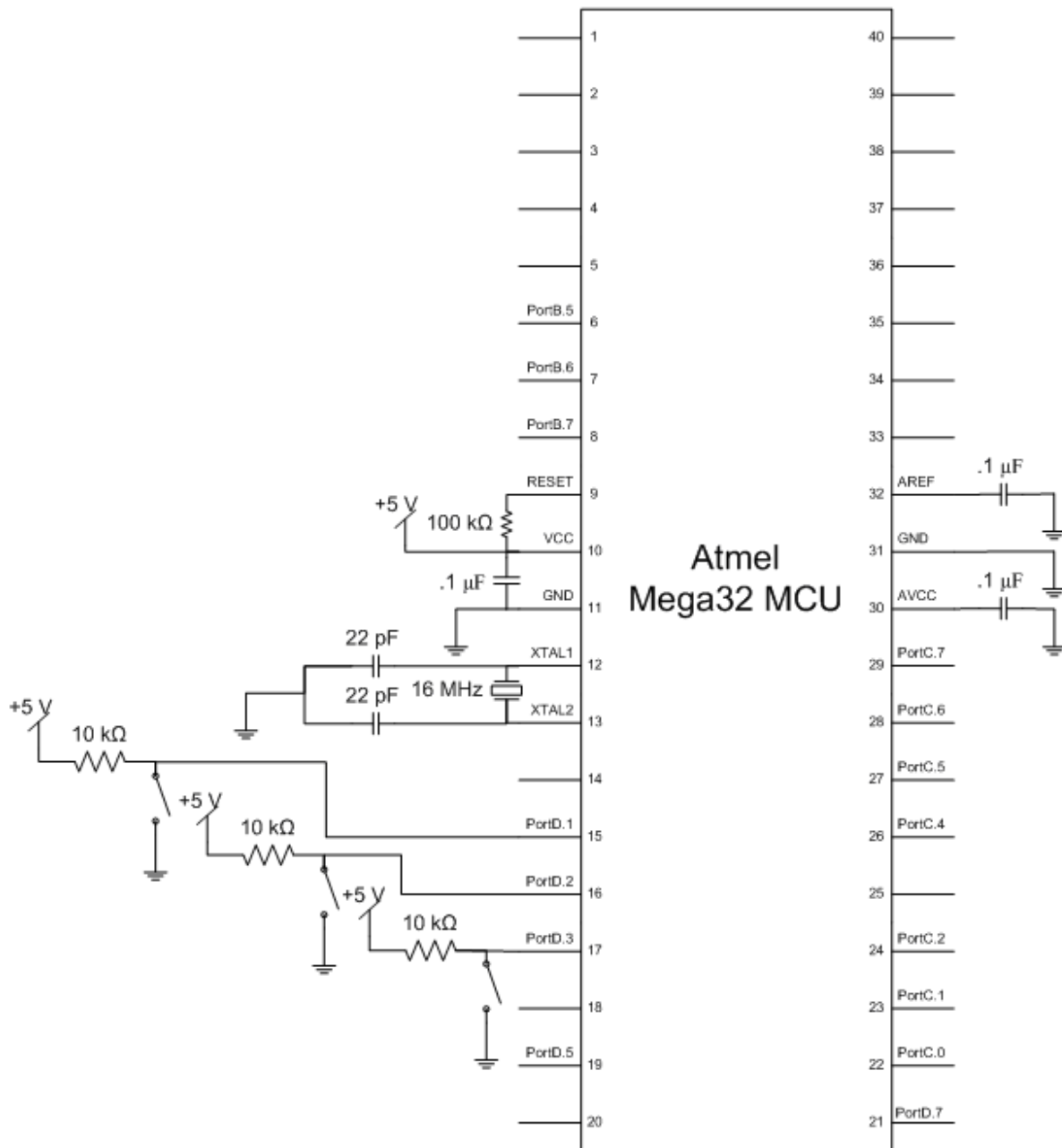
Programming port

LCD Circuit

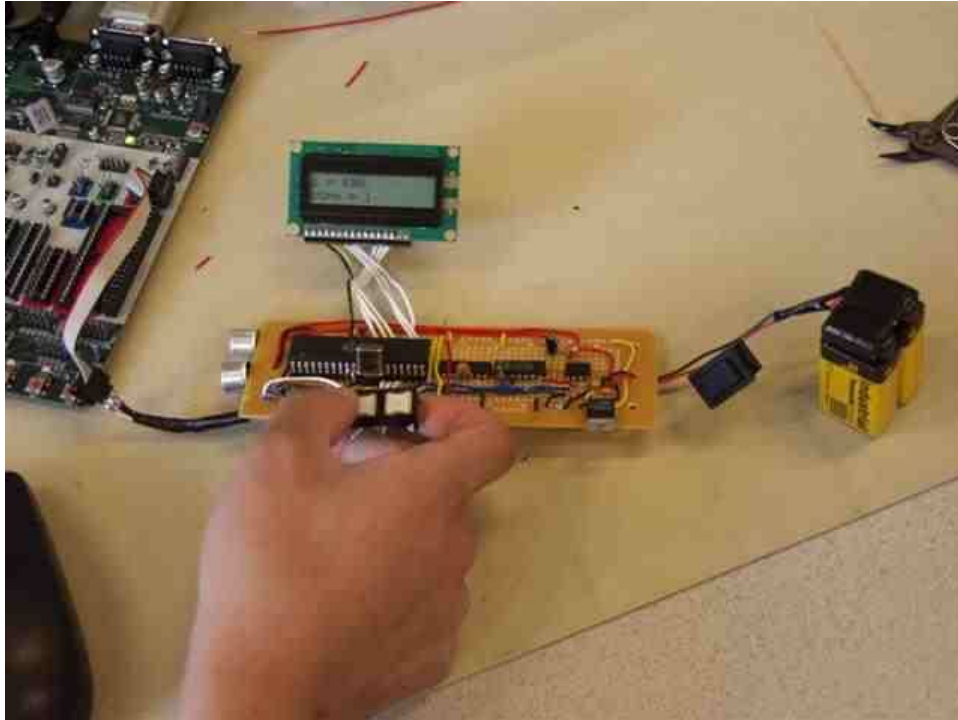


MCU circuit

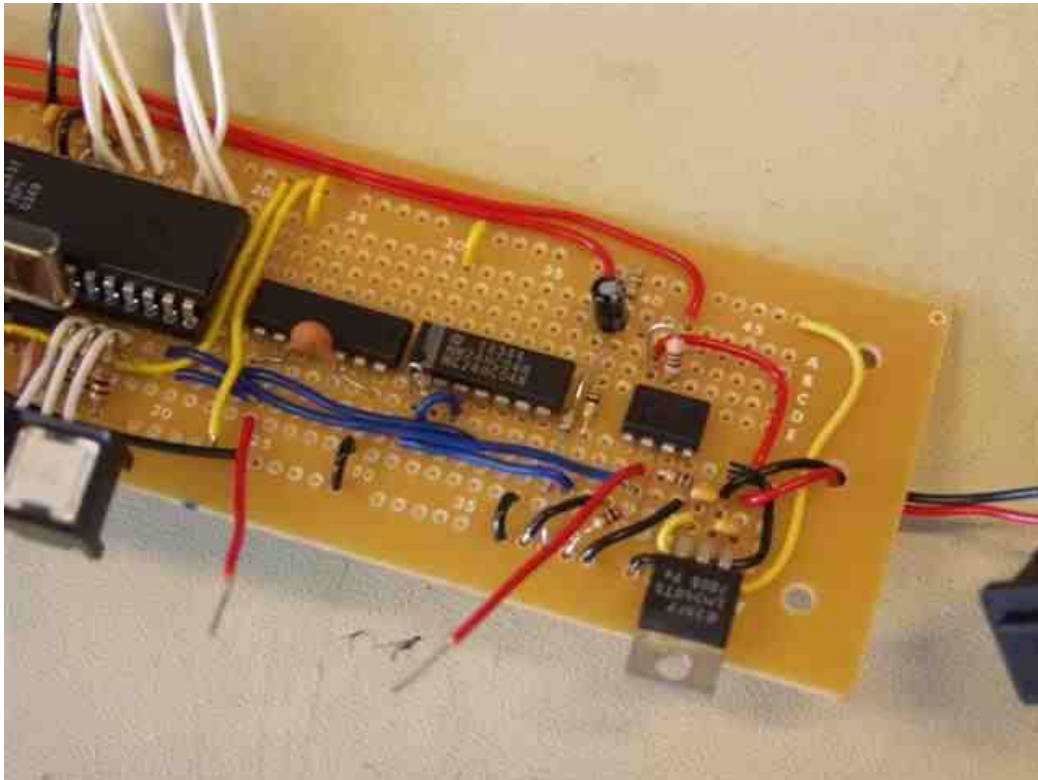
MCU Circuit



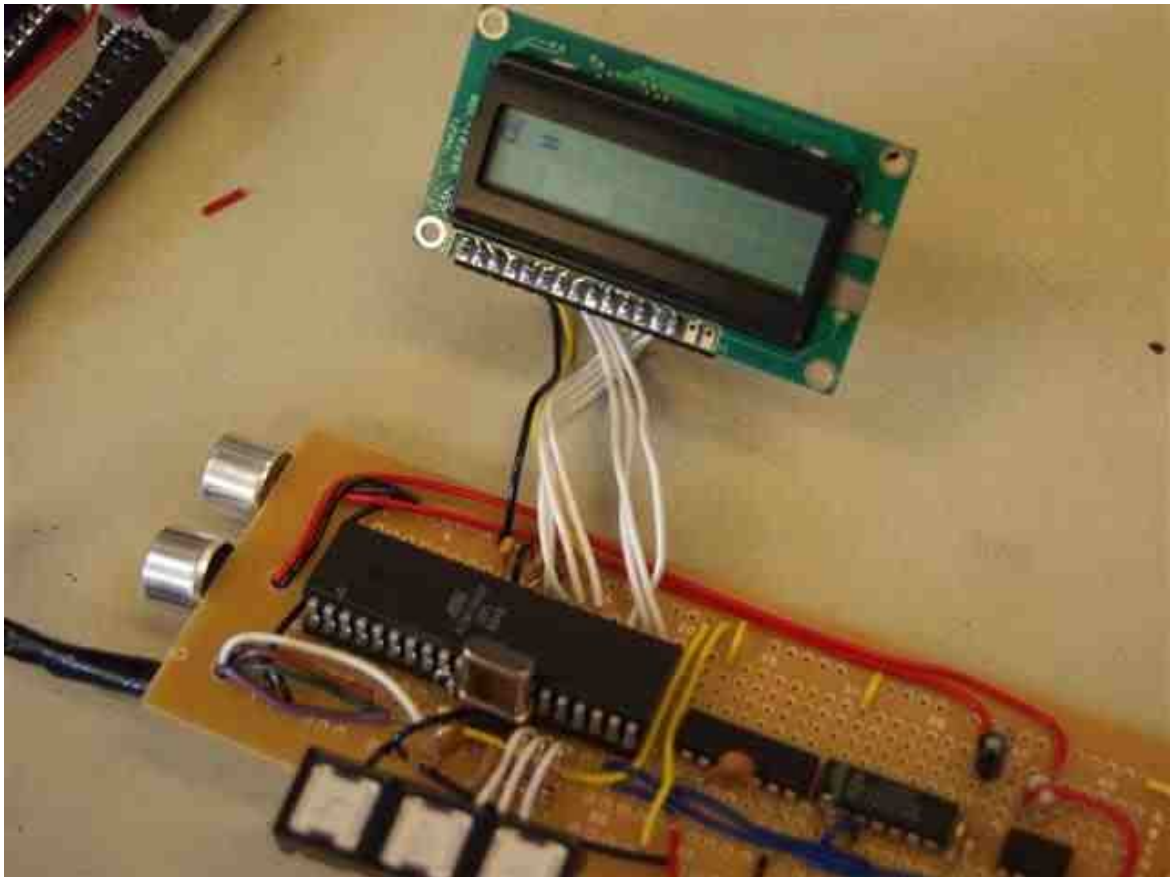
Testing the software



Close-up of the Tx and Rx circuits



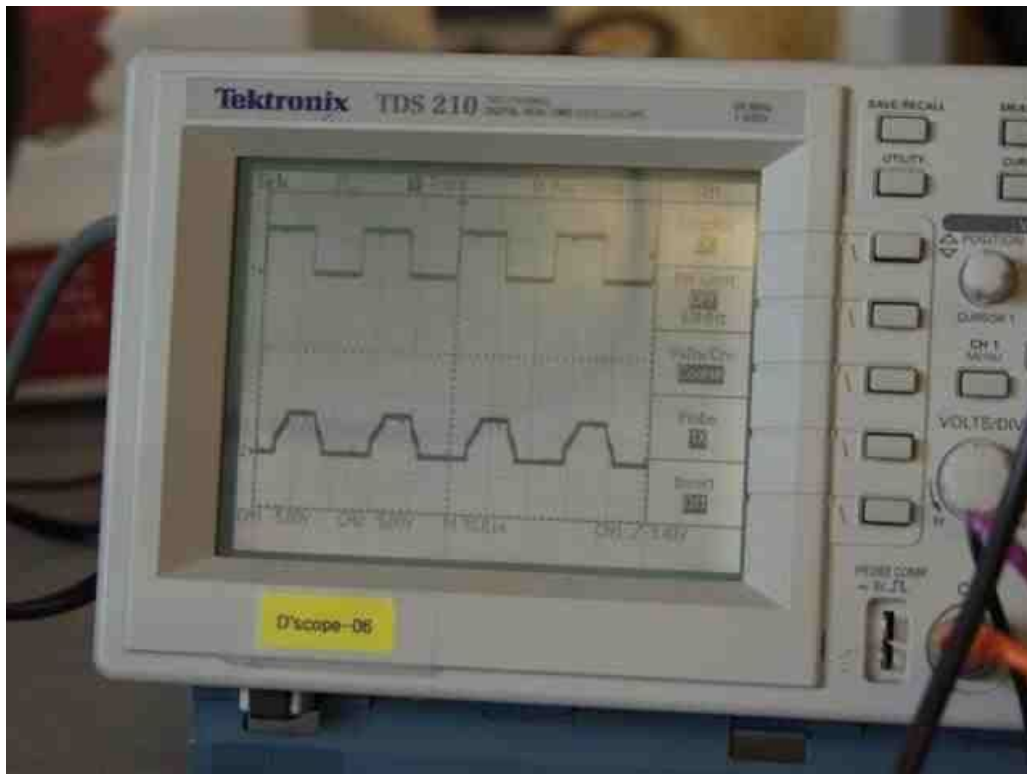
Close-up of the MCU and LCD



Testing the hardware



Sample input and output



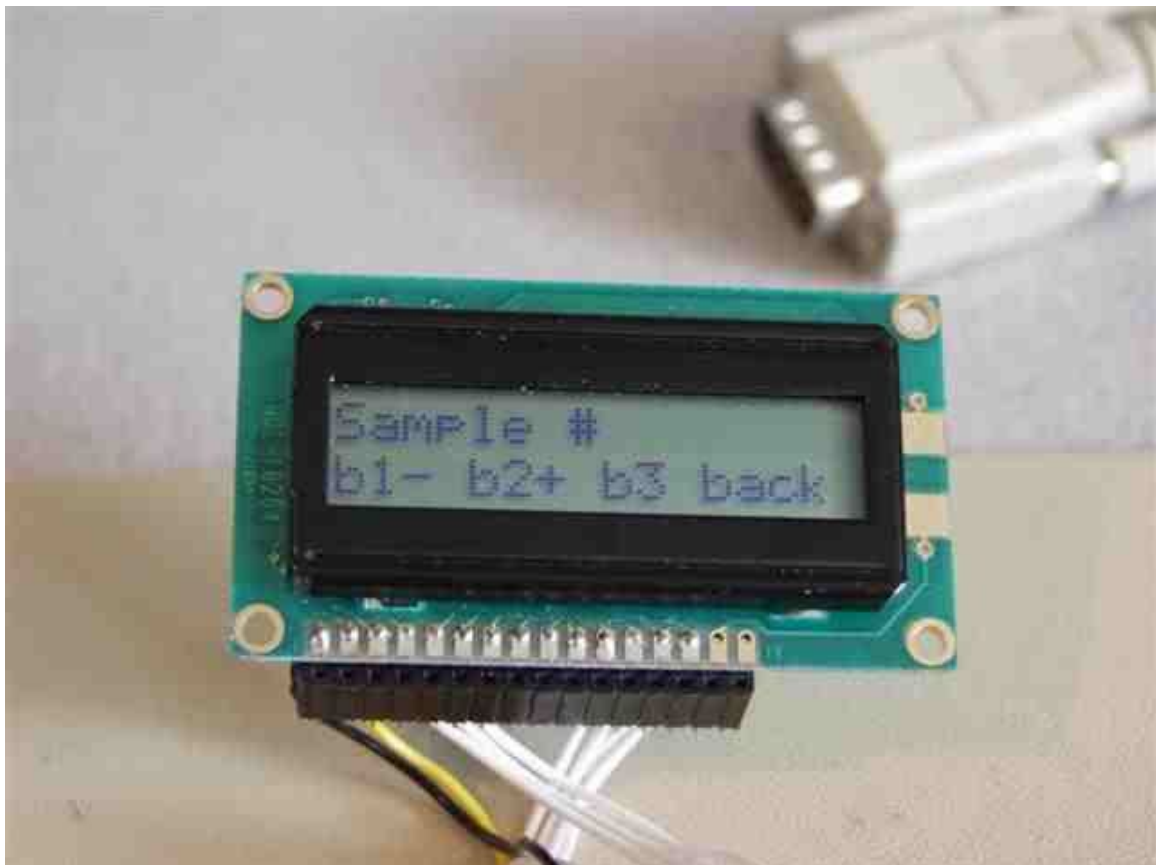
Mode menu



Sample distance and speed measurement



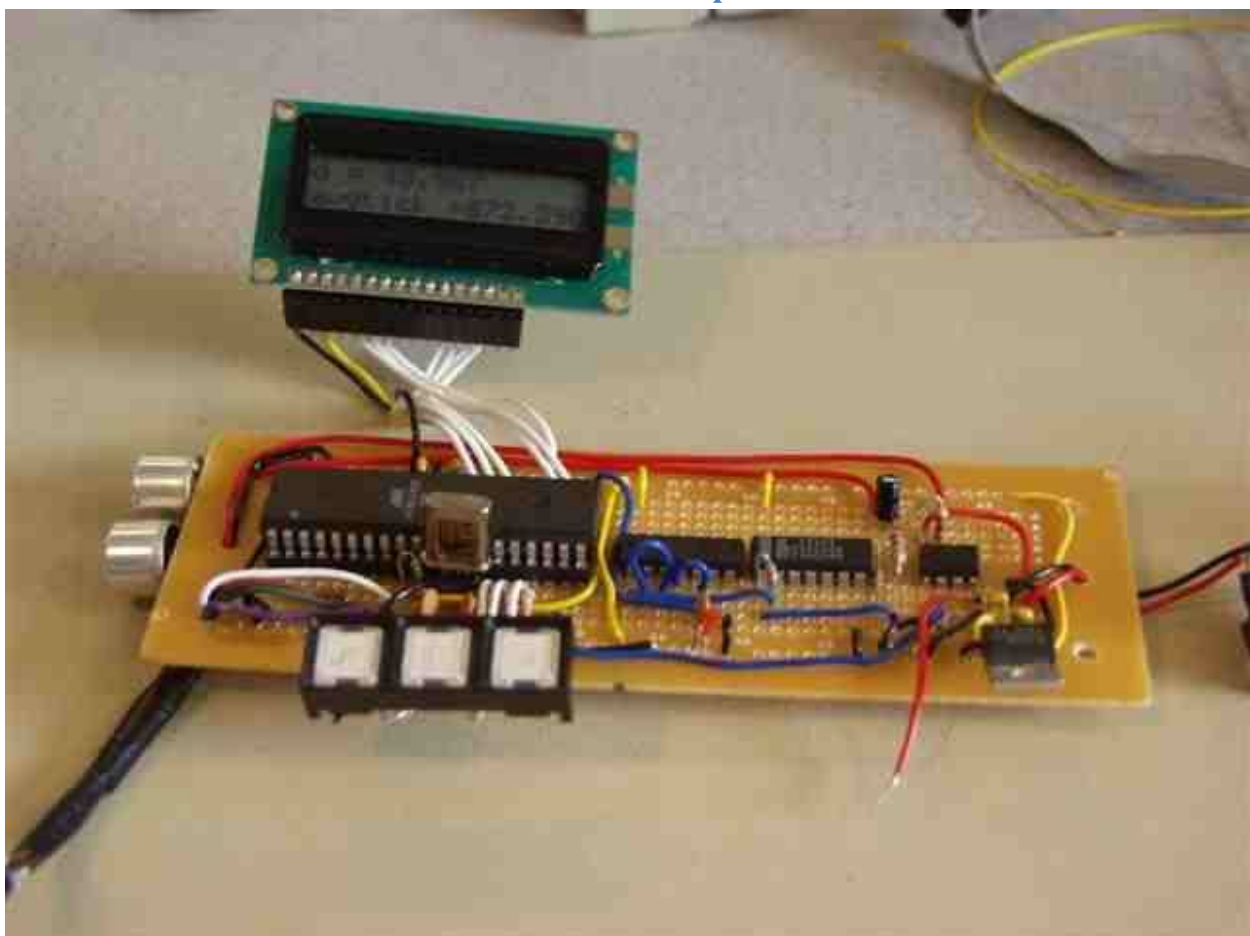
User selectable measurements



Calibration mode



The final product



Testing issues

When testing the MCU program with the hardware, we noticed that the receiver would instantly register a pong whenever the button was pushed to send the ping. As a result, distances of insanely small values were being reported. It ended up being a hardware problem when we wired the two transducers. The feed lines to the transducers were very close physically and it ended up that they were interfering with each other. This problem was not initially noticed when testing with the signal generator. In order to fix this, we rerouted some of the wires on the solder board and moved the transmitting source from PortD.6 to PortD.7 (on the other side of the MCU from the receiving pin). This improved the interference issue and we were no longer seeing spikes on the receiver at the same time as the transmitted ping.

Even though we were able to get rid of some of the noise by rerouting some of the wires on the board, we still had some inexplicable noise. We had thought that if we increased the gain of the signal from the receiving transducer and fed it through multiple Schmitt triggers to clean up the signal, we would be better off. Unfortunately, the gain also increased the amplitude of the noise to the point where it was giving us erroneous measurements. Since we were using a variable power supply as the power source of our board during testing, we noticed that if we decreased the supply voltage from 18 volts down to 9 volts, the noise was reduced. It so happened that the extra gain in the transmitter (from the 18 volts) caused the noise that was being registered by the receiver. By reducing the gain, the noise was reduced and we were able to get more accurate measurements.

By doing this, however, our effective range decreased. This was not really that much concern because a large range with incorrect measurements is useless. A small range with correct measurements is much more desirable.

Accuracy

Initially, we were hoping for a range of a few meters. This would have been possible using a higher voltage supplied by the batteries. The effective range is also directly proportional to the voltage of the batteries. If the batteries start losing power, the gain of the ping will decrease, leading to shorter ranges that can be measured. However, during testing, we noticed that higher voltage supplies caused more noise and interference with the receiver. By using only one 9 volt battery as the power source, our device only has a maximum range of about 20 centimeters.

Orientation of the device with the object that is reflecting the pulse also affects accuracy. Since ultrasound is very directional, aiming the device at an object or wall at an angle will result in the reflection not being picked up by the receiving transducer. Optimal operation requires that the device be pointed perpendicular to an object with a flat surface.

Testing the device led to measurements that were quite accurate. Measurements were only a few millimeters off from actual distances. Calibrating the device incorrectly led to proportionately incorrect measurements. The accuracy of the speed calculations were a little harder to judge since we did not have an accurate alternative mean of measuring speed.

Safety considerations and usability

Because the frequency of the transducers is beyond the range of hearing for humans, there is no safety concern regarding the transducers. The 40 kHz frequency may, however, affect the hearing of other animals.

Another safety concern has to do with the usability of the device in dark conditions. Initially, we had purchased a backlit LCD so the device could be used in the dark. Even though the LCD had a controller that was in the same family as the LCDs that we used in the past (along with an almost identical spec sheet), the LCD did not work. Instead of spending time trying to debug our LCD, we just used one provided by the lab. Alternatively, we could have implemented an audible sound that would indicate to the user how far away he or she was from an object.

Impact on other people's designs

Because the rangefinder does not radiate any RF frequencies and only radiates inaudible pulses, our design did not affect other people's designs in the lab (as far as we know). It is possible that our 40 kHz pulses could have caused interference with any other group using similar transducers, but we received no complaints.

Improvements

Overall, we are quite happy with the results of our rangefinder. This is the one of the first classes we've taken at Cornell that allowed us to build something from scratch that we can actually take home and have pride in accomplishing. Of course, only having a month to do the entire thing means there are some improvements that still can be done.

First off is the LCD. We were quite disappointed that the backlit LCD we had originally purchased did not work. We plan on purchasing the other LCD that was recommended to us so we can have a backlight that works. Being able to use the rangefinder in the dark adds a whole new dimension to its usability. A new pushbutton would also have to be implemented so the user can turn on and off the backlight at his or her convenience.

Mounting the rangefinder in some type of project box or enclosure is another improvement that we'd do had we the time. This would make the rangefinder seem more like a finished product that could be marketable. It'd also make using it simpler and single handed.

Lastly, some of the wiring could be improved. As mentioned earlier, the wiring for the receiving and transmitting transducers is very close to each other and causes some interference. The solder board was laid out left to right without any forethought of where certain circuits should be placed in order to reduce noise. Had the receiving circuit been on one side of the board with the transmitting circuit on the other side, our noise and interference issues would most likely have been greatly reduced.

Standards and intellectual property

As mentioned earlier, our rangefinder was built from the ground up and does not interact with any other device. Thus, there were no considerations for standards or intellectual property that were made during its design and construction.

Ethical considerations (based on the IEEE Code of Ethics)

1) to accept responsibility in making engineering decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment

When originally coming up with the idea of the rangefinder, we considered possible applications that might be beneficial to the public, yet safe for use. We also are aware of the fact that depending on input voltage, the

LM340 voltage regulator may possibly burn the user. Providing means necessary to circumvent the risk by packaging the device in some sort of enclosure would prevent any harm to the user.

3) to be honest and realistic in stating claims or estimates based on available data

When programming the MCU, we had decided to make the rangefinder accurate to the millimeter. Testing proves that this is true. We had also initially intended the rangefinder to work at large distances (a few meters), but due to a hardware wiring issue noise and interference prevented this. This was acknowledged earlier and stated to the user.

7) to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others

When we had run into difficulties in the hardware and software design of the rangefinder, we sought help from Prof. Land and some of the many course TAs. They offered assistance which was taken into consideration and is very appreciative. Without their more advanced knowledge in MCUs, programming, and circuit theory, our rangefinder would have never been a success.

8) to treat fairly all persons regardless of such factors as race, religion, gender, disability, age, or national origin

There is no possible way that a single device can treat fairly every single person of every walk of life. Inherent in any device are limitations that will rule out usage by at least one group of people. Our rangefinder can be used by any person who is able push a button and read an LCD display in English using West Arabic (European) numerals. This encompasses the large majority of people on this planet. To accommodate those who cannot see, an audible alarm that emits a periodic tone proportional the measured distance could be implemented.

9) to avoid injuring others, their property, reputation, or employment by false or malicious action

The only output that our rangefinder creates is a short burst of sound at 40 kHz. This is far beyond the range of human hearing, so our device is not capable of harming others. The sound level is also not of significant power to destroy property through shockwaves.

Programme Code:

```
/*          Vincent Ng (vsn2) / Jerry Wang (jw293)
           ECE 476: Wednesday 1:30 lab
           Final Project
           Handheld Ultrasonic Rangefinder
*/

#include <Mega32.h>
#include <stdio.h>          // sprintf
#include <stdlib.h>         // ftoa

#define LCDwidth 16        //characters
#define t1 1              //timer vars
#define t2 2400
#define nopush 1
#define maybepush 2
#define pushed 3
#define maybenopush 4
#define pulsecount 6      //# of pulses to send
#define waittime 0        //wait time which isn't neccessary
#define dp 3              //decimal points

#asm
    .equ __lcd_port=0x15
#endasm
#include <lcd.h> // LCD driver routines

void initialize(void);
void pulse(void);
void debounce(void);
void checkpin7(void);

unsigned char time1, time2, delayedsend, wait;

char sendpulse, calibrate, displayspeed,received, speedmode, normal, pushstate, D1,D2,
D3, mode;
char lcd_buffer[17]; // LCD display buffer
int count, rcount, samples;
unsigned long ticks, loopcount, ovrflwcount, stoptime, stopcount;
double distance, currdist, prevdist,speed, distsum, speedsum, tsum, ctpm, tpm;

interrupt [TIM0_OVF] void timer0_ovrflw(void){
    ovrflwcount++;
}

void main(void)
{
    initialize();
    while (1)
    {
        if(delayedsend == 0){                                //a delay
before sending the next pulse                                if(time1 == 0){
                                                                pulse();
                                                                time1 = t1;
                                                                }
                                                                }
        if(time2 == 0){
//timer for 30 ms
            debounce();
        }
    }
}
```

```

        time2 = t2;
    }
    checkpin7();
}

}

void checkpin7(void){
    if((ovrflwcount>500)&&(received==0)){
        lcd_gotoxy(0,0);
        //position to upper left on display
        lcd_putsf("d =");
        //another string from flash
        received = 1;
        lcd_gotoxy(4,0);
        lcd_putsf("out of range");
    }
    if((~PIND.7)&&(wait == 0)&&(received==0)){                //if the pin goes high we
received
        stoptime = TCNT0;
        //count the # of ticks
        stopcount = ovrflwcount;                                //count # of
times counter over flowed before receiving
        received = 1;
        //received the pulse
        lcd_clear();
        lcd_gotoxy(0,0);
        //position to upper left on display
        lcd_putsf("d =");
        //another string from flash
        ticks = ((stopcount*256)+stoptime);                    //calculate total ticks
        distance = (((double)ticks)*1.36*(tpm/ctpm))/2;
        //# of ticks * (1.36 mm pertick)*(ticks per 5 cm/calibrated)
        //all over 2 to compensate for sending and receiving
        prevdist = currdist;                                    //store
previous dist
        currdist = distance;                                    //store
current dist
        ftoa(distance,dp,lcd_buffer);                            //display distance
        lcd_gotoxy(4,0);
        lcd_putsf("                ");
        lcd_gotoxy(4,0);
        lcd_puts(lcd_buffer);
        if(normal){
            if(rcount>0){
                delayedsend = 60000;
                //delay enough before sending next pulse
                sendpulse = 1;
                count = 2*pulsecount;
                wait = waittime;
                distsum = distsum + distance;                    //add up the
samples
                if(rcount == (samples/2)){                        //take
the middle and last samples
                    speedsum = distance;                            //for
speed calculation
                }else if(rcount == 1){
                    speedsum = speedsum - distance;
                }
                rcount--;
            }else{

```

```

        distsum = distsum/(samples); //take the
average
        speedsum = (speedsum/(samples/100)); //calculate speed
between first and last
        ftoa(distsum,dp,lcd_buffer);
        //displaystuff
        lcd_gotoxy(4,0);
        lcd_putsf(" ");
        lcd_gotoxy(4,0);
        lcd_puts(lcd_buffer);
        lcd_gotoxy(0,1);
        lcd_putsf("spd = ");
        lcd_gotoxy(9,1);
        ftoa(speedsum,dp,lcd_buffer);
        lcd_puts(lcd_buffer); //end displaying stuff
        distsum = 0; //reset variables
        speedsum = 0;
        normal = 0;
    }
} else if(calibrate){
/*takes 200 samples of how many ticks for 5cm
*averages it and sets it as the calibrated ticks per 5 cm
*/
    if(rcount>0){
        delayedsend = 60000;
        sendpulse = 1;
        count = 2*pulsecount;
        wait = waittime;
        distsum = distsum + distance; //sum up the distances
        tsum = tsum + ticks; //sum up the ticks
        rcount--;
    } else{
        distsum = distsum/((float)samples); //average the distances
        tsum = tsum/((float)samples); //average the ticks
        ftoa(distsum,dp,lcd_buffer); //display all this stuff to lcd
        lcd_gotoxy(4,0);
        lcd_putsf(" ");
        lcd_gotoxy(4,0);
        lcd_puts(lcd_buffer);
        lcd_gotoxy(0,1);
        lcd_putsf("avgtick = ");
        lcd_gotoxy(10,1);
        ftoa(tsum,dp,lcd_buffer);
        lcd_puts(lcd_buffer); //end displaying stuff
        distsum = 0;
        //reset variables
        ctpm = tsum;
        tsum = 0;
        calibrate = 0;
    }
}
}

```

```

void debounce(void){//standard debounce state machine as given in lecture
    switch(pushstate){
        case nopush:
            if((~PIND.1) || (~PIND.2) || (~PIND.3)) // button pushed?
                pushstate = maybepush;
            else pushstate = nopush;
        break;
        case maybepush:
            if((~PIND.1) || (~PIND.2) || (~PIND.3))

```

```

        pushstate = pushed;
    else pushstate = nopush;
break;
case pushed:
    if((~PIND.1) || (~PIND.2) || (~PIND.3)){ // button pushed?
        pushstate = pushed;
        if(~PIND.1) D1 = 1;
        else if(~PIND.2) D2 = 1;
        else if(~PIND.3) D3 = 1;
    }else pushstate = maybenopush;
break;
case maybenopush:
    if((~PIND.1) || (~PIND.2) || (~PIND.3)) // button pushed?
        pushstate = pushed;
    else{
        pushstate = nopush;
        /*
        D1 is normal operation
        D2 is speed detection mode
        D3 is calibration mode
        This checks which button was pressed and sets
        the appropriate flags for each mode
        */
        if(received){
            if(!mode){
                if(D1){
                    sendpulse = 1;
                    count = 2*pulsecount;
                    wait = waittime;
                    rcount = samples;
                    D1 = 0;
                    normal = 1;
                }else if(D2){
                    ctpm = 73;
                    mode = 1;
                    lcd_clear();
                    lcd_putsf("Sample #");
                    lcd_gotoxy(0,1);
                    lcd_putsf("b1- ");
                    lcd_putsf("b2+ b3 back");
                    D2 = 0;
                }else if(D3){
                    sendpulse = 1;
                    count = 2*pulsecount;
                    wait = waittime;
                    calibrate = 1;
                    D3 = 0;
                    rcount = samples;
                }
            }else if(mode){
                if(D1){
                    if(samples>50){
                        samples = samples - 50;
                    }
                    D1 = 0;
                    lcd_gotoxy(0,0);
                    lcd_putsf(" ");
                    lcd_gotoxy(0,0);
                    sprintf (lcd_buffer, "n = %i", samples);
                    lcd_puts(lcd_buffer);
                }else if(D2){
                    samples = samples+50;
                    lcd_gotoxy(0,0);

```

```

        samples);

        lcd_putsf("          ");
        lcd_gotoxy(0,0);
        sprintf (lcd_buffer, "n = %i",

        lcd_puts(lcd_buffer);
        D2 = 0;
    }else if(D3){
        ctpm = 73;
        mode = 0;
        lcd_clear();
        lcd_gotoxy(0,0);
        lcd_putsf("b1 sendpulse");
        lcd_gotoxy(0,1);
        lcd_putsf("b2 mode2 ");
        lcd_putsf("b3 cal");
        D3 = 0;
    }

    }
    } //end received if
    }
    break;
    } // switch
}

void pulse(void){
    //sends a certain number of pulses depending on what the count is
    if(sendpulse){
        received = 0;
        if(count>0){
            TCCR1A = 0b01000011;
            count--;
        }else{
            sendpulse = 0;
            TCCR1A = 0b00000011;
            TCNT0 = 0;
            ovrflwcount = 0;
        }
    }
}

```

```

void initialize(void)
{
    //initialize pushflags for buttons

    D1 = 0;
    D2 = 0;
    D3 = 0;
    normal = 0;
    prevdist = 0;
    currdist = 0;
    rcount =0;
    distsum = 0;
    speedsum = 0;
    tsum = 0;
    // initialize the LCD for 16 char wide
    lcd_init(LCDwidth);
    lcd_clear();
    lcd_putsf("b1 sendpulse");
    lcd_gotoxy(0,1);
    lcd_putsf("b2 mode2 ");
    lcd_putsf("b3 cal");
    sendpulse = 0;
    //set up the ports

    //which pin is pushed

    //initialize the display
    //clear the display
}

```

```

        DDRD=0b00100000;                // PORT D.5 is output rest
is inputs
        PORTD=0;
        //initialize the variables to their proper values
        stoptime = 0;
        displayspeed = 0;
        speedmode = 0;
        loopcount = 0;
        speed = 0;
        mode = 0;
        received = 1;
        samples = 200;
        calibrate = 0;
        pushstate = nopush;
        ticks = 0;
        ctpm= 73.0;                        //init ticks per 5 cm
        tpm = 73.0;
        //set up timer 0 and timer 1
        TCCR0=0b000000011;                //prescalar to 64 and turn
on clear-on-match                          //timer1 compA and TIMER0
        TIMSK = 0x11;
COMP interrupts on                          //timer1 OCR to count 25
        OCR1A = 24;
times and toggle OC1A at 40kHz
        TCCR1A = 0b00000011;              //divide the clock by 8 PWM mode on
        TCCR1B = 0b00011010;
        //init the timers
        time1=t1;
        time2=t2;
        wait = 0;
        delayedsend = 0;
        #asm
        sei
        #endasm
}

interrupt [TIM1_COMPA] void timer1_compare(void)
{
        if (time1>0)                        --time1;                //40khz
        if (time2>0)                        --time2;                //debounce machine
        if (wait>0)                          --wait;                //wait
to receive
        if (delayedsend>0)                  --delayedsend;        //wait to send another pulse
}

```