# Ultrasonic Listener

## Microcontroller Based Frequency Shifter

A Senior Project

presented to

the Faculty of the Electrical Engineering

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Troy Fredriks

7 June 2010

# Contents

# Tables and Figures

# Introduction

Ever wonder what kind of sounds are going on outside of the human audible range? Perhaps the warning signs of a leaky high pressure pipe or some interesting nature sounds are just outside what the human ear is tuned to pick up. This project addresses this very issue. By shifting higher frequency sounds down to the range where human ears can hear, the user is able to get an idea of what kind of sounds are being made that humans cannot detect.

# Background

The human ear is designed to pick up sounds in frequencies ranging from 20 Hz to 20 kHz. However, many interesting sounds are being made all the time with frequencies that are much higher; bats, dolphins, and orcas typically communicate in the 20 kHz – 100 kHz range. It is desirable to ‘listen in’ on these higher frequency sounds, and thus, needs to be shifted down to the human audible range.

## Theory

The basic idea of how to down shift a frequency is to multiply it with another frequency. Consider the basic trigonometric identity [1]:

$$\sin(A) \cdot \sin(B) \equiv \frac{1}{2}[\cos(A-B) - \cos(A+B)] \quad 1.1$$

Therefore, down shifting an ultrasonic frequency of 25 kHz to an audible 5 kHz would require multiplying the 25 kHz signal with a 20 kHz signal:

25 kHz – 20 kHz = 5 kHz

In addition to the downshifted signal of 5 kHz, the multiply will also generate an up shifted signal at:

25 kHz + 20 kHz = 45 kHz

This signal is undesired and should be filtered out for high fidelity applications or simply ignored in low fidelity applications (most consumer audio electronics would not be able to actually physically produce this frequency anyhow).

## Understanding the analog circuit approach

The basic approach for shifting the frequencies into the audible region involves the use of a double balanced mixer. Recall that the basic function of the double balanced mixer is to output a sum and difference of the two input frequencies (via multiplication):

$$\omega_O = (\omega_1 + \omega_2) + (\omega_1 - \omega_2) \qquad 1.2$$

We can use this property to our advantage by focusing on the difference portion of the mixer (the sum portion can simply be filtered out). For example, to shift the frequencies over the range of 20 kHz to 40 kHz down to the audible range, a subtraction of 20 kHz is needed. Substituting $\omega_1$, in equation 1.2 above, with the range of 20 kHz to 40 kHz and $\omega_2$ with 20 kHz, the following output is observed:

$\omega_O$ = (40-60 kHz) + (0 -20 kHz)

Note that now we have a range from 0 – 20 kHz (desired) and another range from 40 kHz – 60 kHz (undesired). To eliminate the unwanted range from 40 kHz – 60 kHz, a lowpass filter with a 3 dB roll off at 20 kHz is added after the output stage. Finally we arrive at a solution that will shift frequencies from 20-40 kHz range down to the audible range. One major limitation with this approach is that in order to change the range, the 20 kHz oscillator will need to be changed accordingly (40 kHz to shift the range 40 kHz -60 kHz , etc.). This limitation can be easily overcome, along with other added benefits, by implementing a digital design using a microcontroller.

## Why choose a Microcontroller?

The microcontroller approach resembles the basic analog approach very closely. In the microcontroller design, the double balanced mixer will be implemented in the digital realm by the processor as a simple multiply. In addition, the oscillator frequency can be generated by the processor, and thus, can easily be changed for quick range changes.

Moving from the analog realm (where the signal is generated) to the digital realm (where the signal processing takes place) requires the use of an analog to digital converter (ADC). Once the signal has been processed, the signal passes through a digital to analog converter (DAC) to be amplified through a speaker.

Additional benefits of using a microcontroller include: use of an LCD display to monitor ranges and create an easier user interface and limits use of outboard hardware because many filters can be implemented in the digital world.

Because the microcontroller operates at a voltage range of 0 to +3.3 volts, an additional consideration needs to be made to ensure accuracy of the multiply: will offsetting the signals by a positive DC bias negatively affect multiply?

To help visualize this dilemma, plots have been created demonstrating both cases.

Figure 1, below, shows a plot with zero DC bias; series 1 is multiplied by series 2 and the output is series 3. This is idealized case and contains no parasitic leak thru of the original signals.

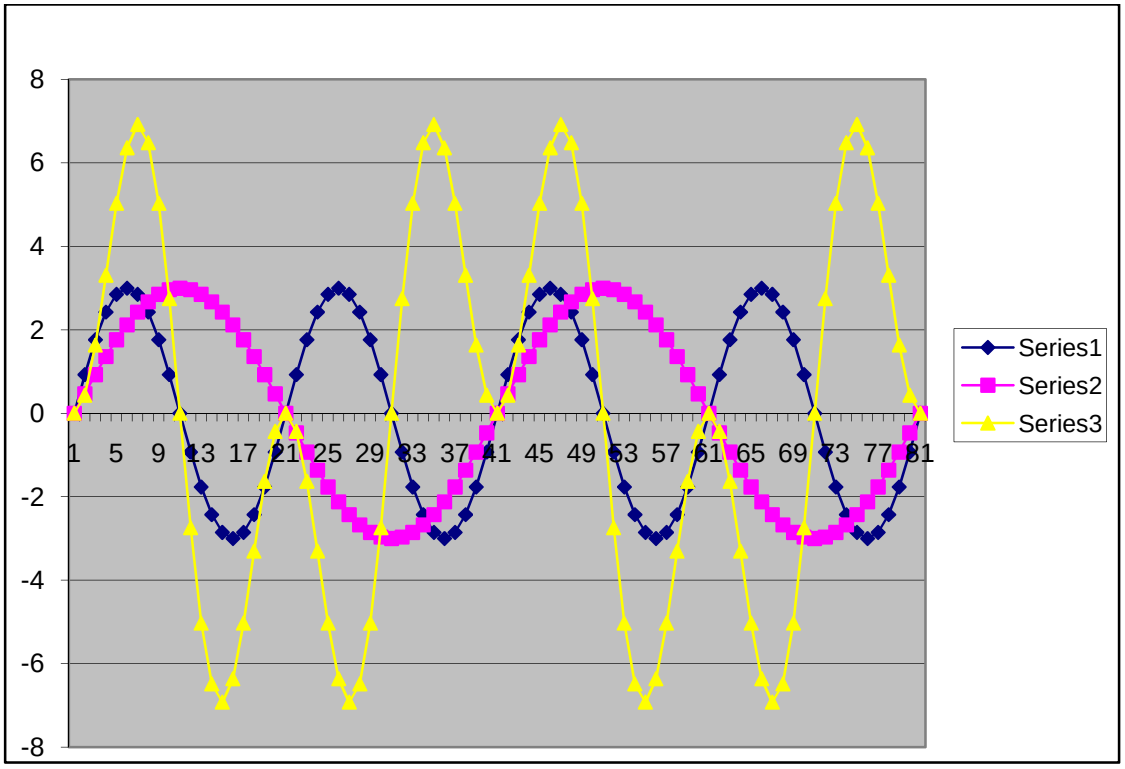


**Figure 1: Theoretical output of multiplied waves with zero offset**

Figure 2 shows the case with a positive DC bias; it concludes a non-ideal case where extreme peaking occurs in certain locations as well as leak-thru of the original signals.

This non-ideality can be compensated by using *signed* numbers in the program. In this way we can treat the midpoint of the microcontrollers operating voltage (1.65V) as a 0 to make the multiply function correctly.

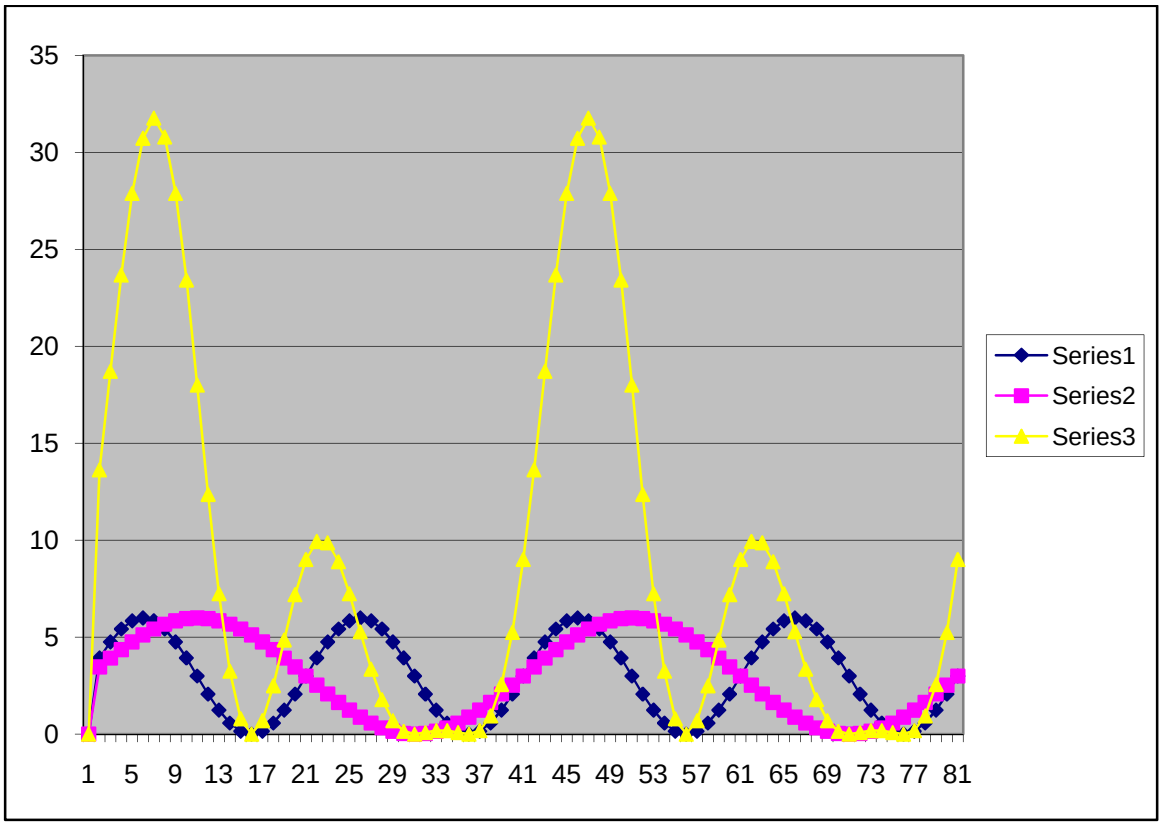


**Figure 2: Theoretical output of multiplied waves with positive offset**

# Requirements

Functionalities:

- Shift frequencies from 20 kHz – 60 kHz down to 200 – 10200 kHz
- Filter unwanted frequencies
- Establish appropriate line levels

Data Requirements:

- Microcontroller based signal processing
- High-speed A/D and D/A conversion

User Interface:

- LCD display
- Input for frequency selection
- 1/4” output jack

System Requirements:

- Ultrasonic Transducer (at least 20 kHz – 60 kHz bandwidth)
- Microcontroller with A/D and D/A conversion
- Audio Amplifier

Other Requirements:

- Ultrasonic frequency generator (for testing)

# Equipment

This section details the necessary equipment needed to design and test the project. Microchip's MPLAB IDE V8.36, MPLAB ICD 3 Debugger, and the Explorer 16 development board all integrate to allow full functionality of the MCU.

## Design Tools

### MPLAB IDE V8.36

MPLAB IDE V8.36 is Microchip's development environment for any of its many different MCUs. MPLAB IDE allows the user to create programs in C, assembly language, or other with the appropriate compiling software. MPLAB IDE integrates with, and controls the MPLAB ICD 3 to allow for programming and debugging of programs. MPLAB IDE also has an emulator that enables the user to test their programs on a simulated version of any MCU Microchip offers.

### MPLAB ICD 3 Debugger

MPLAB ICD 3 is Microchips newest edition of the ICD series for programming and debugging. With this tool, I was able to easily debug the errors in my program code. This tool allows insertion of 'break points' in the code; these break points halt the program where the break point is inserted, allowing inspection of

all registers for verification of correct values. The debugging tool also allows for instructions to be processed step by step, further allowing me to investigate errors in the code.

### Microchip Explorer 16 Development Board

The Explorer 16 is a demo board designed to provide easily integrated peripherals to many of Microchips 16 bit and 32 bit processors, and allows for quick exchange of MCUs with the intelligent Plug-in Module (PIM) design. The LCD display, buttons, LEDs and power connections are of particular interest to the design process of this project. Using the Explorer 16 demo board reduced development time considerably and allowed me to focus on the program and other peripherals.

## Lab Equipment

In addition to the software development tools, two pieces of lab equipment are necessary in the development process:

- Oscilloscope
- Function Generator

# Hardware

## Block Diagram

The block diagram helps to get the overall big picture of what is going on at the hardware level. Figure 6 below shows the main blocks of the project, with arrows indicating the direction of signal flow.

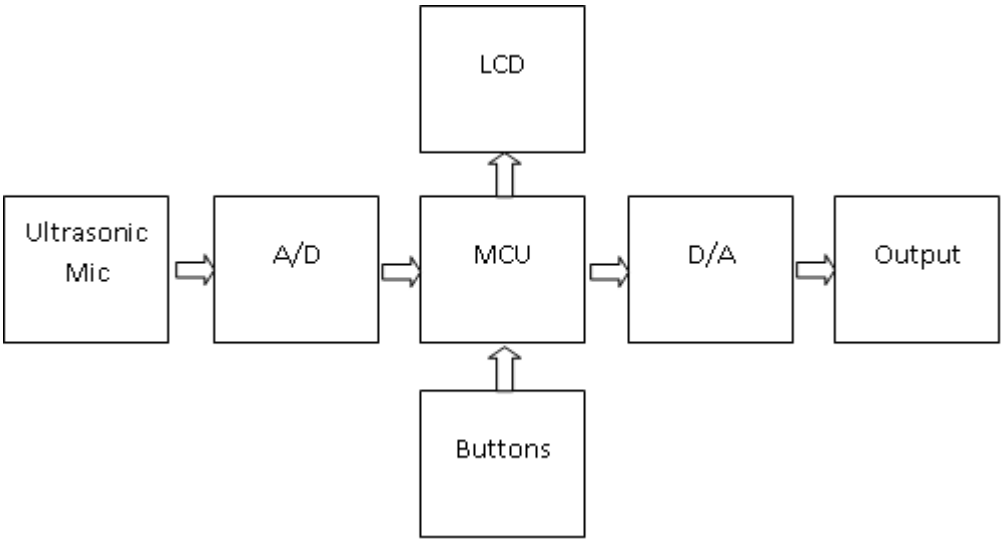


**Figure 3: High Level Hardware Block Diagram**

## Ultrasonic Transducer

The first piece of hardware in the overall chain is the ultrasonic transducer. The chosen device is **Knowles Acoustics' SPM0404UD5 : Mini SiSonic Ultrasonic Acoustic Sensor.** This device was chosen for its wide and relatively flat pass band response over 10 kHz to 65 kHz.

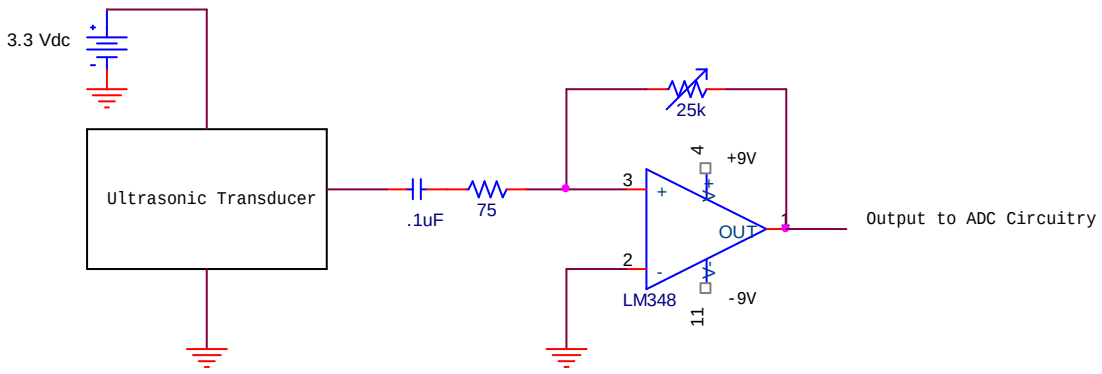


**Figure 4: Ultrasonic Transducer Circuitry**

Figure 7 shows a detailed schematic of the ultrasonic transducer and the appropriate gain stage that follows. The .1uF capacitor couples the transducer output to the gain input; acting as a DC blocker isolating the transducer output from the gain stage so only an unbiased AC signal feeds thru.

Gain is accomplished via simple inverting operational amplifier configuration with a variable feedback resistor (potentiometer) for appropriate gain selection. Maximum gain is calculated (ignoring phase inversion):

$$A = \frac{25k}{75} = 333.3\frac{V}{V}$$

In dB:

$$A = 20\log 333.3 = 50.45\ dB$$

Thus, the gain can vary from -∞dB to +50dB, providing sufficient flexibility for the input to the AD converter for maximum resolution.

## A/D converter

The 10-bit analog to digital converter is built on-board the microcontroller making integration much easier. A DC bias network is created from two 100k ohm resistors in series from +3.3V to ground. This voltage divider ensures the ‘zero point’ into the A/D is right in the middle of its maximum voltage swing. Two voltage protection diodes are added, one each, in parallel with the biasing resistors to protect the A/D from over, or under, voltage inputs. A coupling capacitor links the output of the ultrasonic transducer stage to the DC bias network to block DC and only allow AC to pass. Figure 8 shows the schematic.

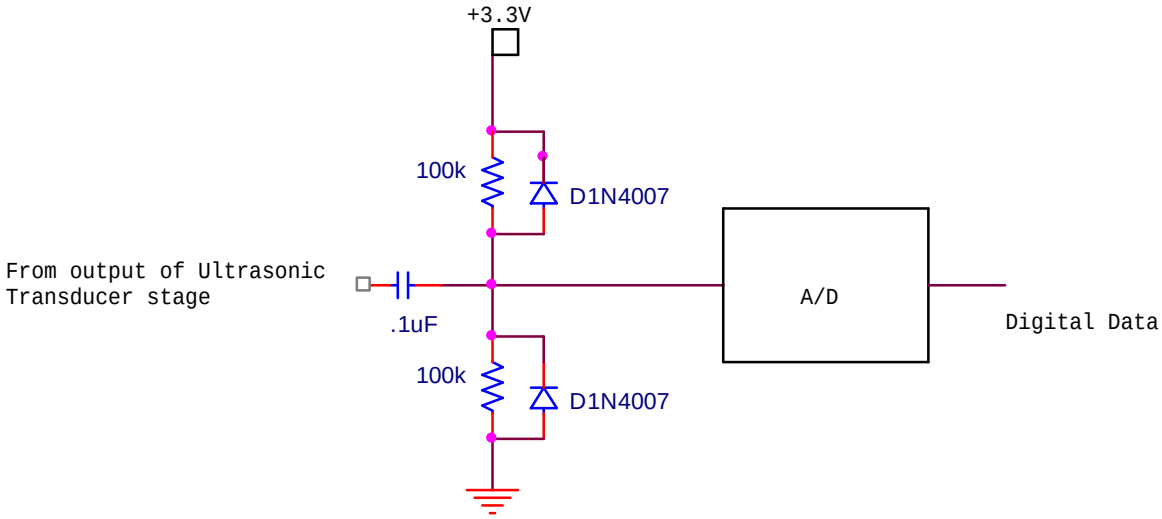


**Figure 5: A/D converter circuitry**

## MCU – Microchip dsPICFJ256GP710A

The processor chosen for this project was Microchip's dsPIC33FJ256GP710A. This is one of Microchip's fastest and feature rich processors and includes many special features with digital signal processing in mind. Special considerations pertaining to this project include:

- Up to 40 Million Instructions per second (MIPS)
- On board A/D with sampling rates as high as 1.1 MHz
- DSP Engine provides single cycle multiply and barrel shifts

Detailed information on the programming of this block will be discussed in the **Program Code** section of this document.

## LCD display and buttons

The LCD display and the buttons are integrated on the Explorer 16 development board for easy integration into the project. The LCD is a two line 16 character display which guides the user thru the program flow. The buttons allow the user to navigate thru the different program states. Further information on the use of the LCD and the buttons will be discussed in the **Program Code** section of this document.

## Digital to analog converter

The 12-bit digital to analog converter is external to the processor and converts the processed signal back to an analog form that can be amplified and played through a speaker. The DAC chosen is Analog Devices' DAC312 for its parallel input, to save clock cycles, and for its extremely low settling time (250ns). The DAC is a 20 pin DIP package and has been configured to operate on 2's compliment logic due to the nature of the signed logic that was implemented.

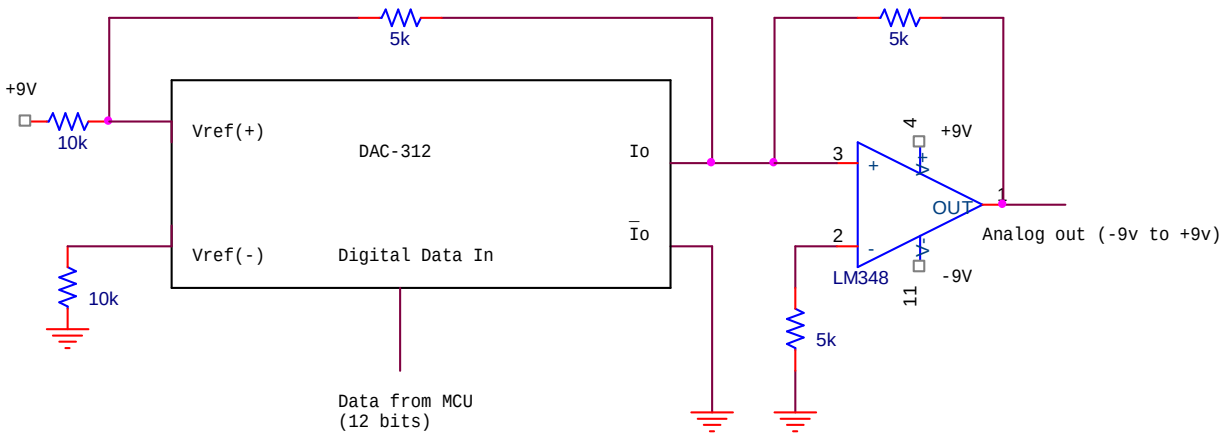


**Figure 6: DAC-312 wiring Schematic (set up for 2's Compliment output)**

Figure 9 shows the complete schematic with the required op amp current to voltage converter (recommended setup connections and values from datasheet). The DAC reads a 12 bit 2's compliment number and outputs a voltage from -9v to +9v. This signal is then passed to the final circuit for preparation as output at standard consumer audio levels.

## Output circuitry

Once the signal has been produced in the analog realm via the DAC, the signal finally needs to be processed to properly integrate with consumer electronic amplifiers.

First the signal needs to pass thru a coupling capacitor to remove any possible DC offset errors produced by the DAC.

Consumer electronics are designed for an input of -10 dBV or .316$V_{RMS}$. Considering the maximum peak voltage from the DAC is 9v the max RMS voltage would be:

$$\frac{9V}{\sqrt{2}} = 6.36\ VRMS$$

Dividing the 6.36 $V_{RMS}$ by the target .316 $V_{RMS}$ results in the resistor ratio of the voltage divider:

$$\frac{6.36}{.316} = 20.12\ (unitless)$$

Thus, the appropriate resistor values for the voltage divider network are approximately 20:1. The chosen resistors of 47k and 2k were chosen based on availability and have a ratio of 23.5:1.

Lastly the output has been terminated with a 1/4 “output jack for easy interfacing with many consumer grade electronic amplifiers.

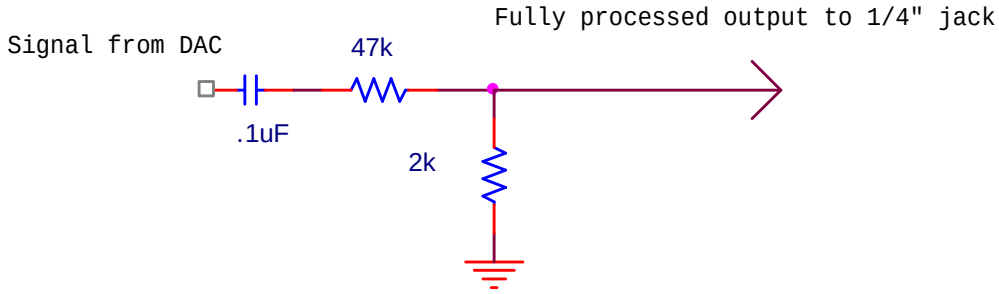


**Figure 7: Final output circuitry consists of coupling capacitor and voltage divider**

# Program Code

The program code is broken up into two main parts: the main infinite loop and the interrupt service routine (ISR). Coding is done in C programming language and has been optimized for fewest clock cycles.

## Main Loop

The main loop can be characterized by having two functions: 1) setting up and modifying the all the variables that control which frequency band is to be shifted and 2) to act as a system idle service when no other instructions are being executed.

The main loop can change between ten different ‘states’, where each state handles the information necessary for updating the LCD display with the correct frequency band and for setting up the frequency of the self generated sine wave (this is done by setting up the timer that controls the ISR). The table below shows each state and the corresponding LCD display and generated sine frequency data.

| State Number | LCD Display | Sine Frequency |
|---|---|---|
| 0 | 10 kHz – 20 kHz | 10 kHz |
| 1 | 15 kHz – 25 kHz | 15 kHz |
| 2 | 20 kHz – 30 kHz | 20 kHz |
| 3 | 25 kHz – 35 kHz | 25 kHz |
| 4 | 30 kHz – 40 kHz | 30 kHz |
| 5 | 35 kHz – 45 kHz | 35 kHz |
| 6 | 40 kHz – 50 kHz | 40 kHz |
| 7 | 45 kHz – 55 kHz | 45 kHz |
| 8 | 50 kHz – 60 kHz | 50 kHz |
| 9 | 55 kHz – 65 kHz | 55 kHz |

**Table 1: Main loop state data**

With the use of the buttons, one and two, the program can change states and update the correct information. Button one lowers the frequency band with each press unless it is already at the minimum, in which case, it does nothing more. Button two does the opposite, increasing the state number one by one until it reaches the maximum. The program is initialized in state 0.

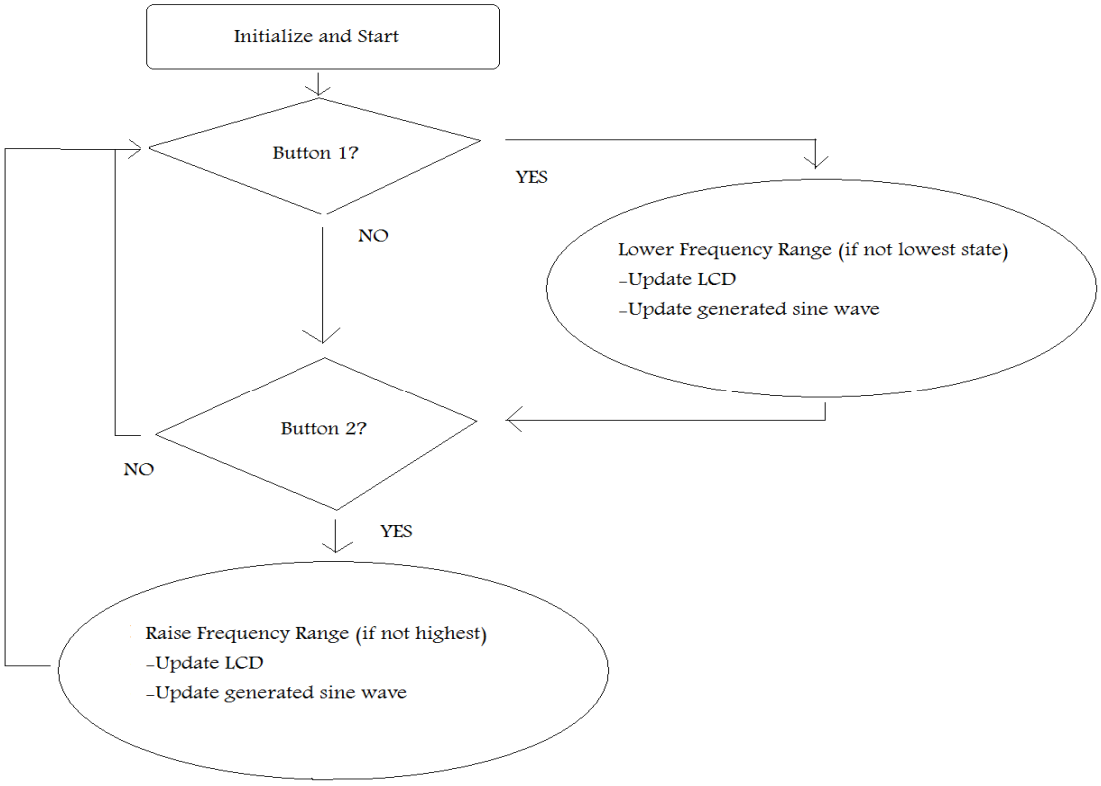


**Figure 8: Program flow of main loop**

Figure 11, above, shows the program flow of the main loop. If no buttons are pressed, the program simply loops around until either: 1) a button gets pressed or 2) an interrupt is generated. The code for the main loop is shown below, each line with a descriptive comment following the C code. The variable *CtrFreq* acts as the state number.

```
/* Infinite Loop */
while ( 1 )
{
    if(PORTDbits.RD6 == 0)           //Enter if button 1 is pressed
    {
        IEC0bits.T1IE = 0;           //Disable interrupts
        if(CtrFreq > 0)CtrFreq -= 1; //Lower frequency range if it is not at lowest range
        PR2 = OSC1[CtrFreq];         //Send frequency information to sine generating timer
        Update_LCD_Ctr_Freq();       //Update LCD information
        Delay(Delay_15mS_Cnt);       //Delay
        IEC0bits.T1IE = 1;           //Enable interrupts
    }
    if(PORTDbits.RD7 == 0)           //Enter if button 2 is pressed
    {
        IEC0bits.T1IE = 0;           //Disable interrupts
        if(CtrFreq < 9)CtrFreq += 1; //Increase frequency range if it is not at highest range
        PR2 = OSC1[CtrFreq];         //Send frequency information to sine generating timer
        Update_LCD_Ctr_Freq();       //Update LCD information
        Delay(Delay_15mS_Cnt);       //Delay
        IEC0bits.T1IE = 0;           //Enable interrupts
    }
   Nop();                            //No Operation (single cycle delay)
   Nop();                            //No Operation (single cycle delay)

}
```

## Interrupt Service Routine

The interrupt service routine is a simple routine that performs the multiply and outputs the data to the port pins of the MCU. When the ISR first gets called, it fetches a data point from a table of values that correspond to the sine function. For the sine wave generation, a table consists of 20 points, equally spaced, throughout one full period of the wave. The ISR timer is configured in the main loop (with state changes) to generate an interrupt corresponding to the desired frequency of the sine wave. To generate the 10 kHz sine wave, the ISR timer needs to be configured to generate an interrupt, then, at 20 times (because there are 20 ‘sample ‘points per period) 10 kHz or 200 kHz. During each ISR the data is also read from the ADC, thus, the sampling rate for the ADC is also 200 kHz at the lowest 10 kHz – 20 kHz state and increases as the state increases.

Once the ISR has loaded the data from the sine table and the ADC, it then follows by performing the multiply and outputting the data to the correct ports. Figure 12 below shows the program flow of the ISR.

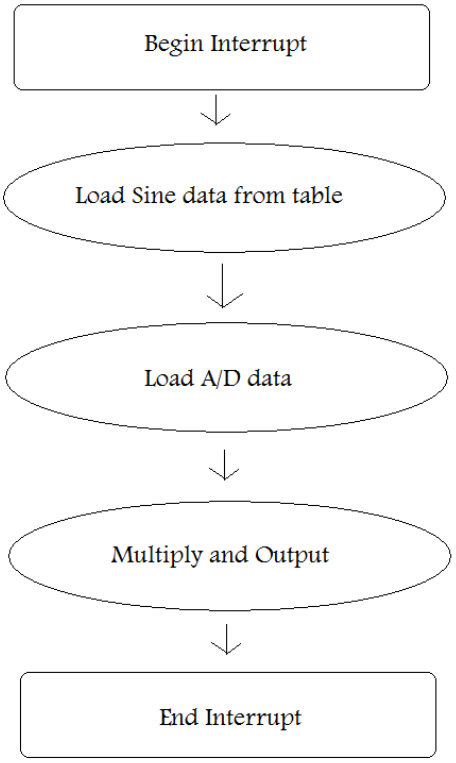


**Figure 9: Program flow of ISR**

The code for the ISR is slightly more complicated mainly due to the fact that it must compensate for moving the data to fit into the correct data output port registers. The code for the ISR is shown below, each line with a descriptive comment following the C code.

```
void __attribute__((__interrupt__)) _T2Interrupt( void )
{
    SinPtr += 1;                //Incriment the sine table pointer
    if(SinPtr > 19) SinPtr = 0; //Reset the sine pointer to 0 at the end of the table
    SinValue = OSC1Val[SinPtr]; //Assign the variable SinValue with new value from table
    ABufRead = ADC1BUF0;        //Read ADC
    ABufRead -= 500;            //Subtract 500 to compensate for offset of ADC
    AD1CON1bits.SAMP = 0;       //Start next conversion
    Prod = SinValue*ABufRead;   //Multiply
    DACData = Prod >> 8;        //Shift most significant data to the lowest 12 bits
    Portdat = DACData;          //Copy data to Portdat
    PORTA = Portdat;            //Send lower 8 bits to port A
    Portdat = Portdat>>8;       //Shift bits 8 - 11 downto 0 - 3
    PORTG = Portdat ^ 0x0008;   //Invert the most significant bit and assign MSBs to port G
    IFS0bits.T2IF = 0;          // reset Timer 2 interrupt flag
}
```

# Analysis

This section uses analysis tools such as the oscilloscope and function generator to confirm and demonstrate correct operation. Most of the work involved in confirming operation is done with the aid of the ICD 3 debugging unit which allows the programmer to perform step by step instructions while simultaneously confirming the correct values for all registers of interest.

## Data

The first thing that can be evaluated is that the LCD and the buttons are functioning correctly. Upon start up, during the initialization stage, the LCD correctly displays my name, Troy Fredriks, along with Cal Poly as seen below in figure 13 below.

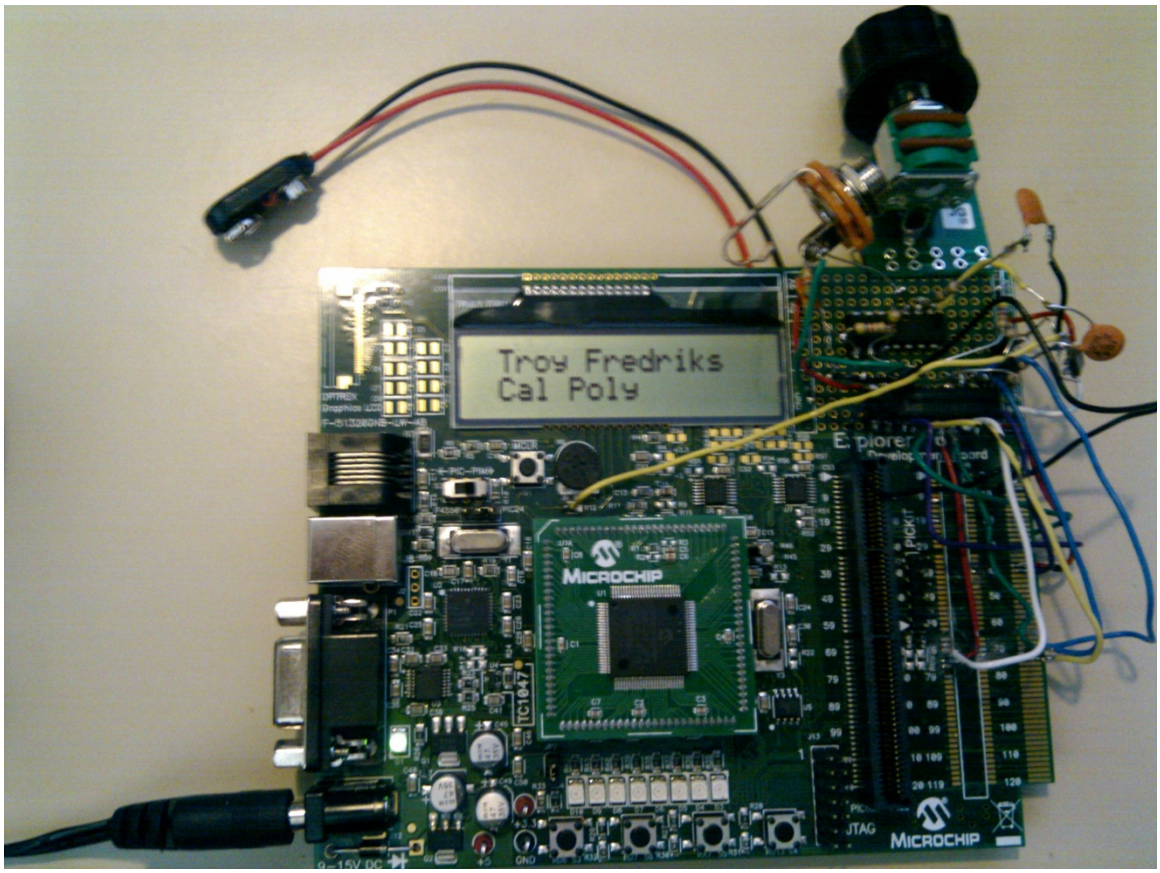


**Figure 10: LCD initialize test**

The LCD also displays the frequency range of interest accordingly in the display. The buttons allow proper state changes from the lowest state to the highest state and back again.

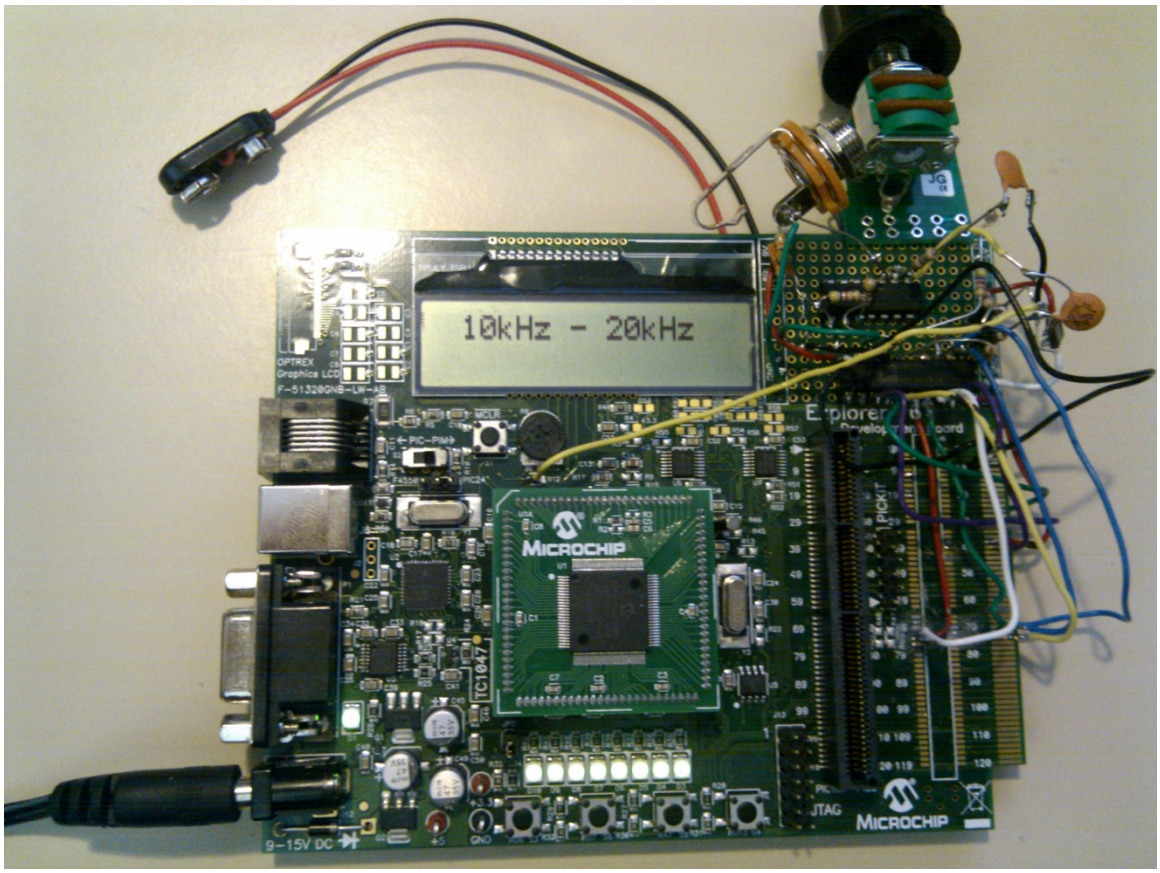


**Figure 11: LCD display with state 0 loaded**

With the LCD and buttons working properly, testing of the self generated sine waves can now be performed. In state 0, the generated sine should be approximately 10 kHz. By tying the ADC to a fixed extreme value, the sine will be multiplied by a constant and will show up at the output as a sine wave of 10 kHz.

Figures 15, 16 and 17 show properly constructed sine waves at 10 kHz, 15 kHz, and 20 kHz, respectively. These plots correspond to increasing state values of 0, 1, and 2.

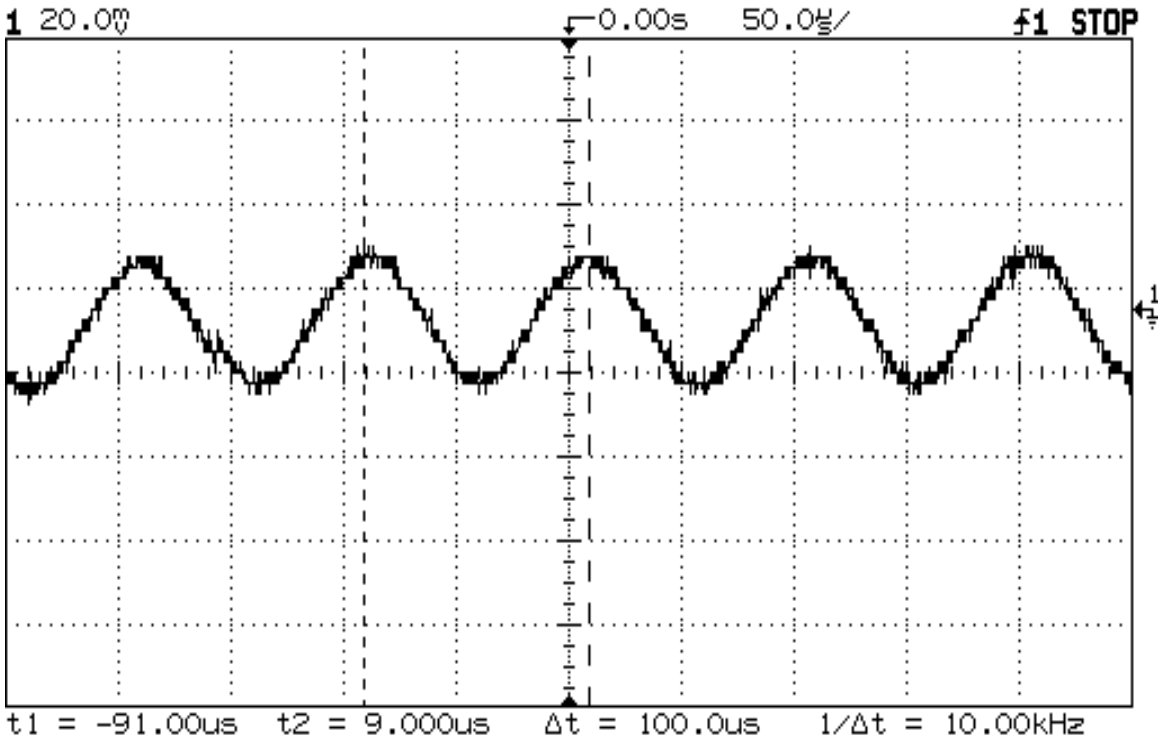


**Figure 12: MCU generated sinusoid @ 10 kHz**

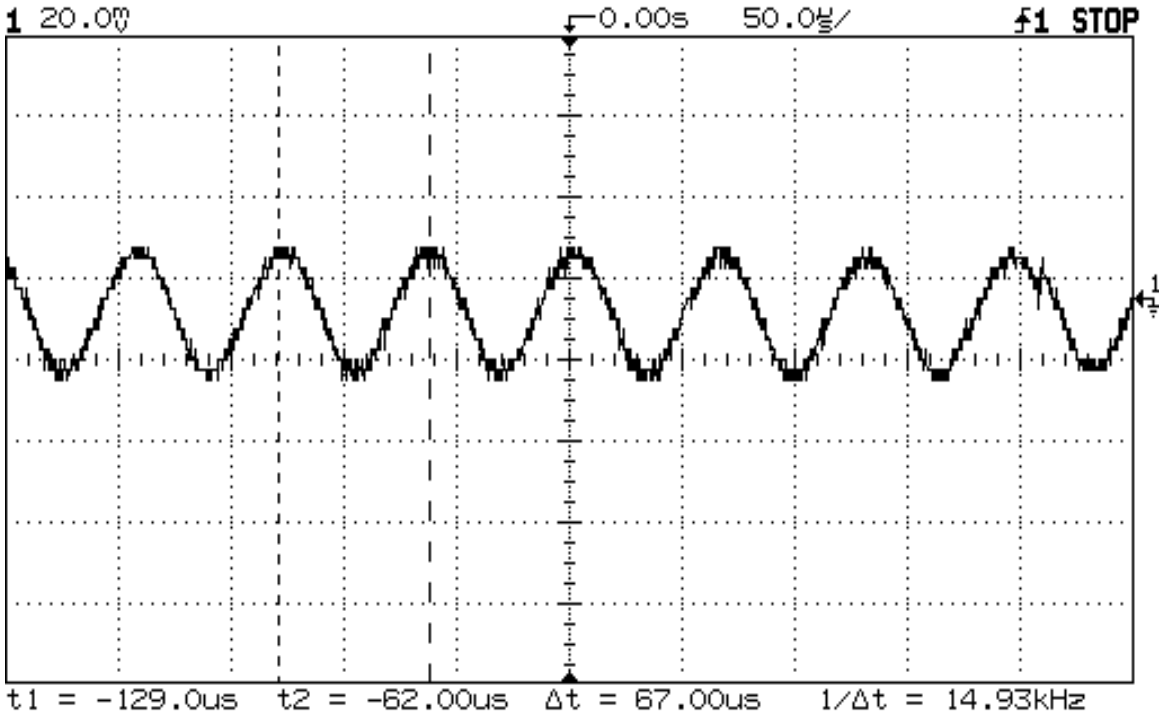


**Figure 13: MCU generated sinusoid @ 14.93 kHz**

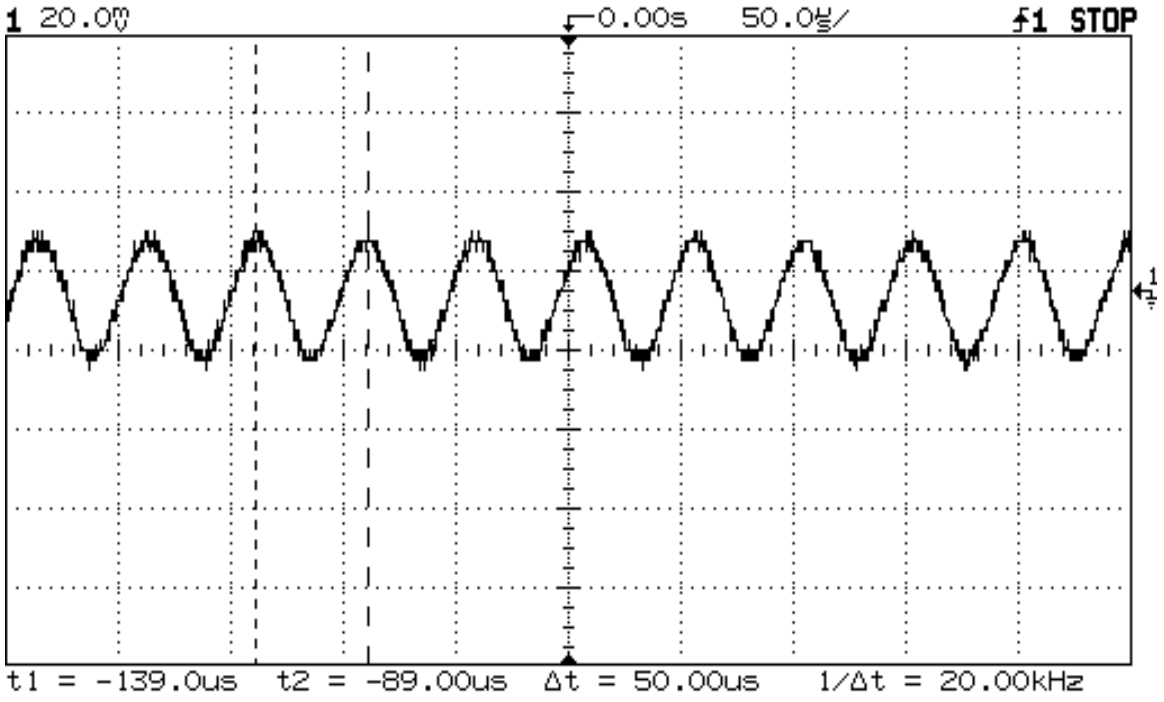


**Figure 14: MCU generated sinusoid @ 20 kHz**

Figure 18 below shows how the MCU fails to properly produce the next state frequency of 25 kHz. This is due to the fact that the processor has run out of clock cycles to perform all of the program code fast enough.

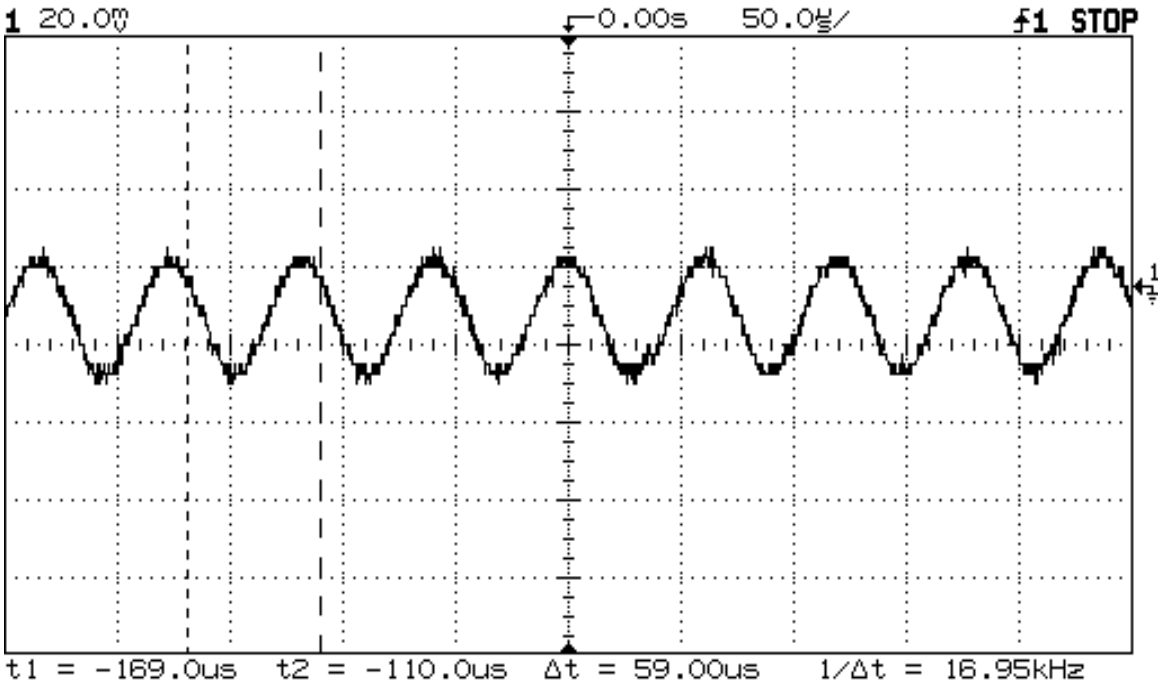


**Figure 15: MCU fails to properly produce 25 kHz**

With confirmation of the DAC and sine wave generation from the previous testing step, analysis of the multiply function is now ready to be performed. Testing is first performed by bypassing the ultrasonic transducer and inputting a pure sine wave from the function generator. Figure 19 below shows the output results of mixing a 14 kHz test signal with the MCU's self generated 10 kHz signal. The output shows a downshifted signal of 4 kHz and an upshifted signal of 24 kHz. The multiply has succeeded.

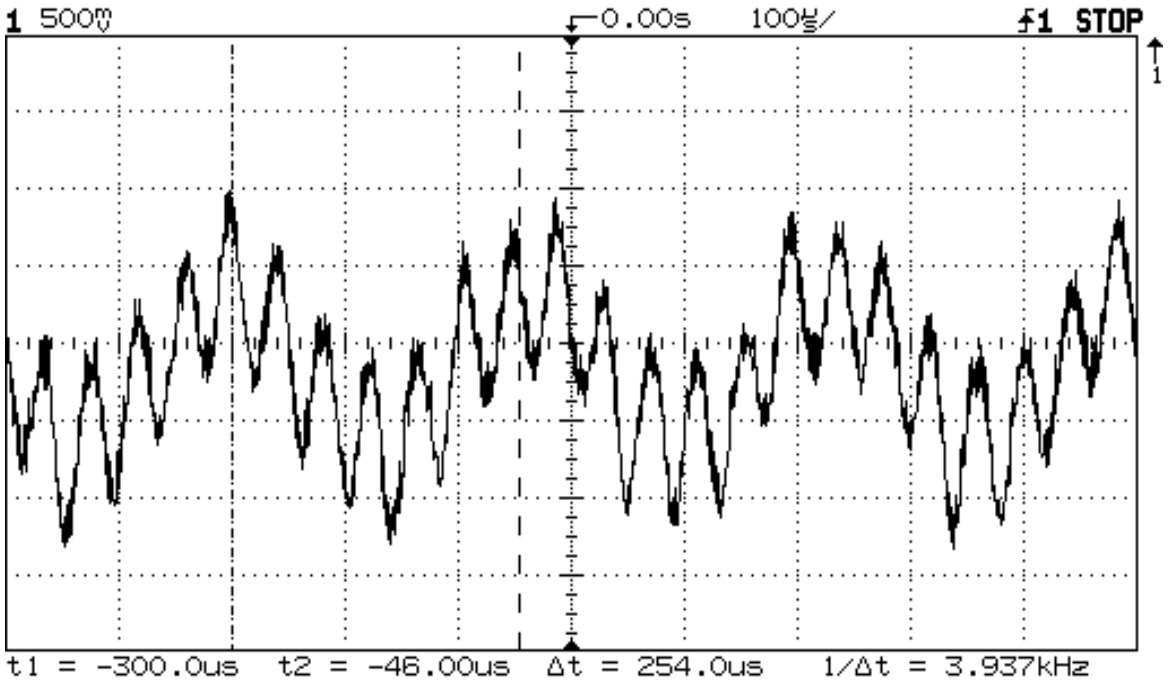


**Figure 16: 10 kHz generated by MCU mixed with 14 kHz provided by function generator**

Next we mix a 25 kHz signal from the function generator with the 20 kHz signal generated by the MCU. The results can be seen in figure 20 below; 5 kHz signal and 45 kHz signals are produced.

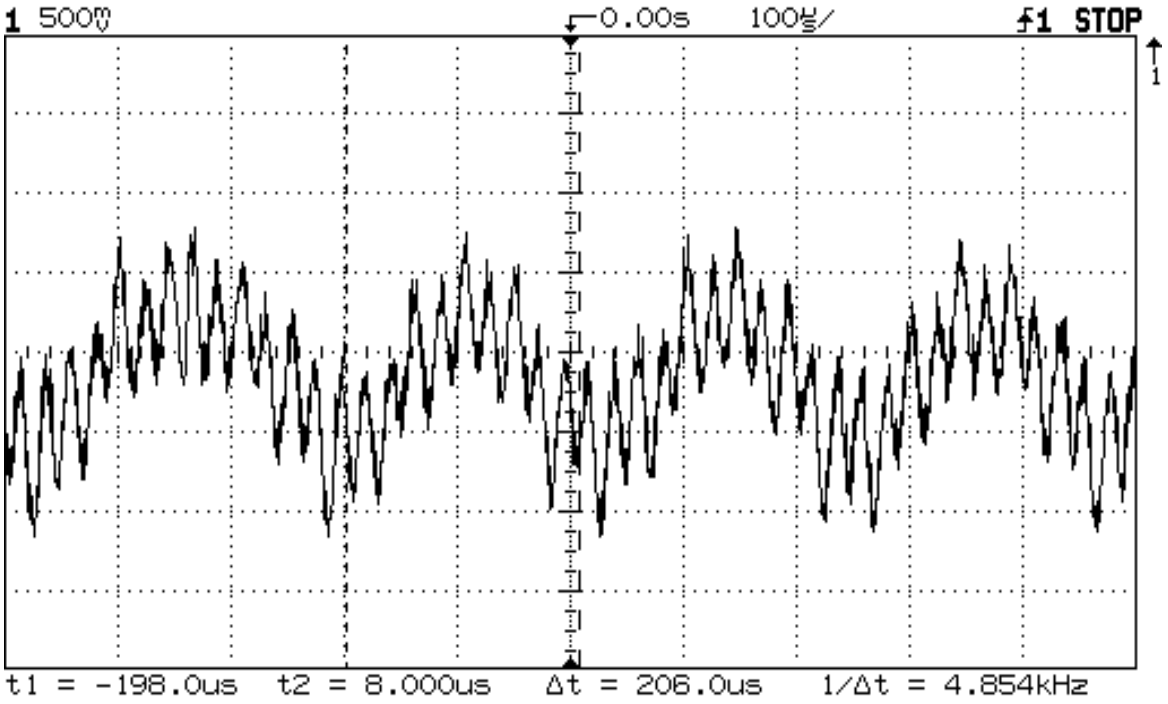


**Figure 17: 20 kHz generated by MCU mixed with 25 kHz provided by function generator**

The final test is to hook up the transducer and use a real acoustic source as input. A dog whistle was chosen as an adequate test source due to its high frequency acoustic properties. Figure 21 below shows the results, indicating the dog whistles' fundamental frequency is on the order of approximately 11.2 kHz.

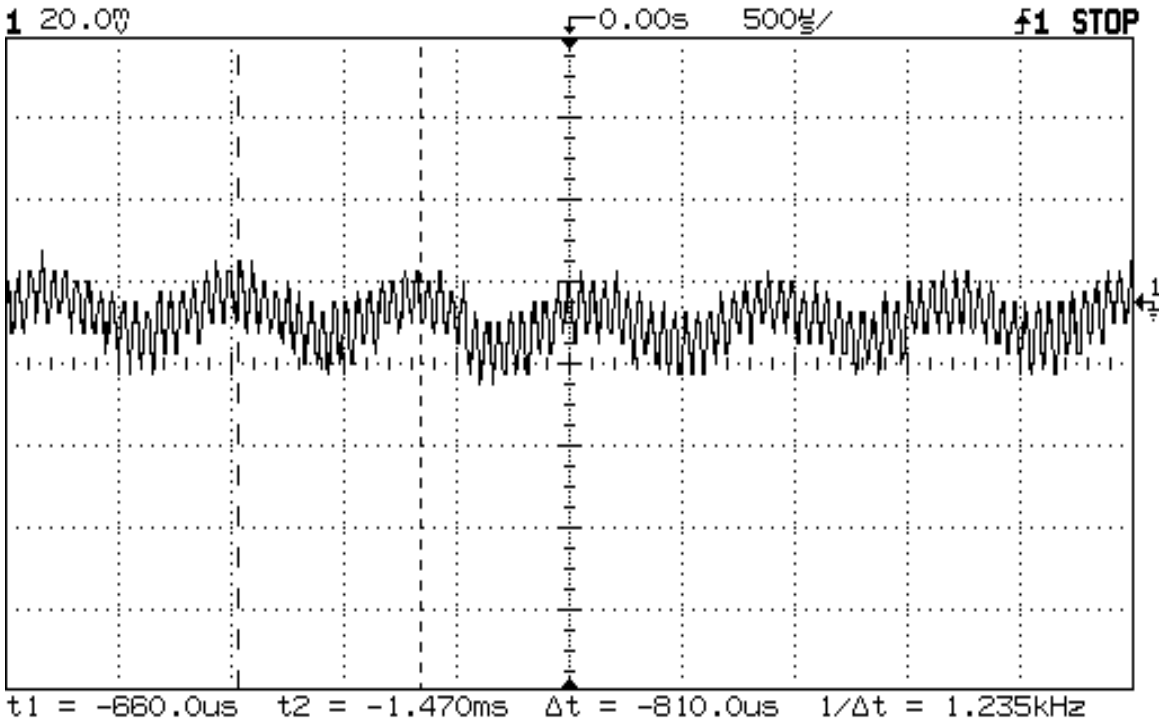


**Figure 18: Dog whistle mixed with 10 kHz results in 1.2 kHz plus 22 kHz**

# Cost Considerations

Cost is a very important consideration when creating a new design. Costs need to be minimized to maximize profits in a business environment. Very rarely is cost of little concern. What follows is a cost analysis of the entire design process.

The total hours spent developing the project, from start to finish, is on the order of approximately 120 hours. Considering an estimated typical salary of $50,000 per year, labor costs are approximately $3,125.00 assuming a 40 hour work week.

The design tools are the next largest contributor. The ICD3 debugger, Explorer 16 development board, and the dsPIC33 MCU are sold bundled together for $300.00

The last consideration is the other hardware components including: the ultrasonic transducer ($17) the DAC ($9) and various resistors and capacitors ($4).

This results in a total design cost of approximately $3500.

# Sustainability/Societal Impacts

The importance of considering sustainability and societal impacts has never been greater in our society. Sustainability considerations are ones that promote endurance of our species, habitat and interactions of our surroundings. Using microcontrollers to program specific functions as opposed to implementing the same task in analog hardware components provides significant benefits to sustainability. Not only is it possible to upgrade the device with a simple software change, as opposed to swapping out physical hardware; if the device becomes obsolete, it can be totally reprogrammed with an entirely different function in mind. This greatly serves to limit the undesirable impacts concerning sustainability.

# Bibliography

[1]

http://en.wikipedia.org/wiki/Frequency_mixer

[2]

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

[3]

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en537580

[4]

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2519&param=en025319&page=wwwdevdsPICBoardKits

# Appendix

## Schematic