ECE 341: Junior Design

Fast Fourier Transform Tone Detector

Kyle Barton, Miles Drake, Samuel Barton

Fall 2020

Instructor: Matt Shuman
Grading TA: Shane Witsell

1 Introduction and Requirements

As a culminating term project, a task was assigned to create a system that detected musical notes from sampled audio signals. This was to be done with an Arduino Nano, an electret microphone, and the accompanying filtering, amplifying, and output circuits. The agreed upon implementation used the Arduino output pins to power one of eight LED lights, indicating which note was detected.

To calculate which note is detected, a Fast Fourier Transform (FFT) algorithm is used within the arduino. An FFT algorithm estimates the discrete Fourier transform which decomposes a given signal into its corresponding frequency components. Where the discrete Fourier transform is slow and complex (O(N^2) complexity), an FFT can be performed rapidly and accurately with significantly less resources.

The requirements of this assignment were given explicitly as follows:

1. Each of the 8 notes, reasonably in tune (+/- 5%) , must turn on only one LED. Computation can be performed on the Arduino or in MATLAB, but the LEDs must be controlled by the Arduino.
2. The audio source must be more than 10 feet away from the audio amplifier microphone supplied in the 341 kit.
3. Audio amplifier must have a signal with more than 1 volt of amplitude when stimulated with an audio source.
4. Audio amplifier must have less than .1 .2 volts of amplitude when not stimulated with an audio source.
5. More than 20 samples must be acquired for the detected period.
6. At least 3 periods of acquired audio data must be graphed in MATLAB. This is not required to be done during a live session. CSV data can be recorded and then post processed for this requirement.
7. Compute the SNR (Signal to Noise Ratio) of the sampled audio signal in MATLAB. The SNR must be at least 20.
8. Plot the spectrum of Power vs Frequency in MATLAB. Plot at least 3 harmonics of the signal on the frequency graph.

This project required a wide array of both technical and non-technical engineering skills. The project management and organization aspects were of even more difficulty due to COVID19 and remote instruction. Along with these soft skills, this project tested our electrical engineering technical skills by taking the work from assignments to a hands on design.

Some of the tools used to complete the project were:

- Circuit design and analysis
- LTSpice simulation and analysis
- Physical prototyping with breadboards and protoboards
- MATLAB and Arduino coding
- Digital measurement and debugging
- Arduino firmware programming and serial communication

2 Block Design

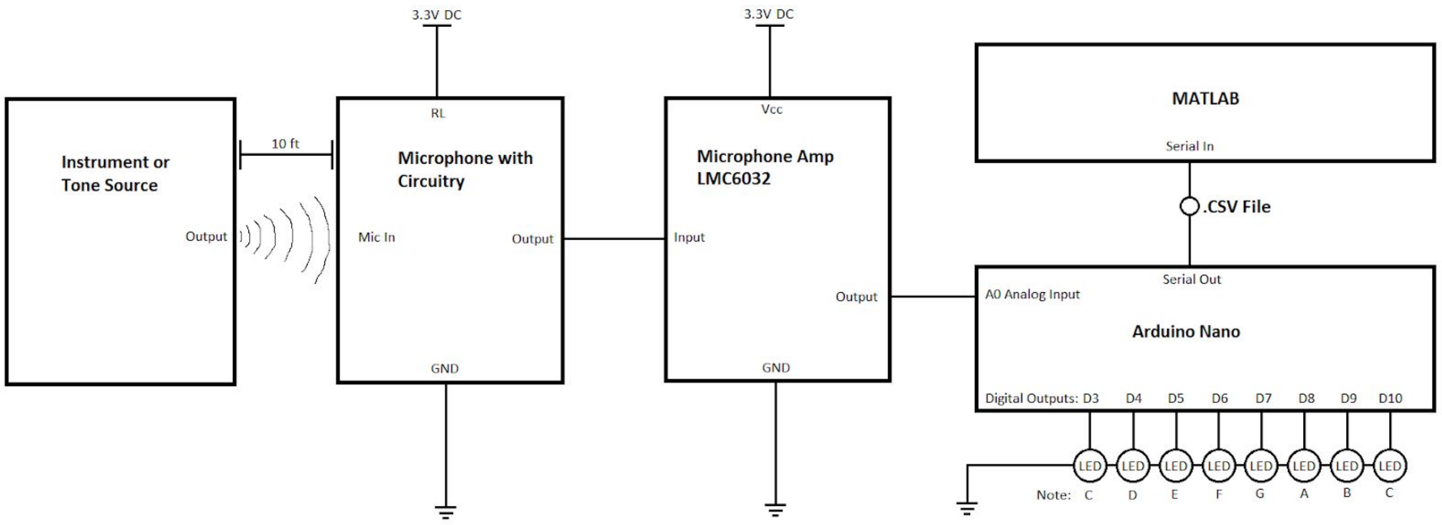


Figure 1: Top Level Block Diagram

From the requirements it was a relatively straightforward block diagram to create and only one main decision needed to be made: would the MATLAB processing take place live in the system to identify the tone or would it be done as post processing? The assignment requirements did not call for MATLAB to process the data live, so it was decided to instead: take in the raw voltage data, write this data as a csv file, process the signal onboard the Arduino to calculate the frequency, then power the LED's from the arduino.

Performing the FFT calculation onboard the Arduino avoided a potential problem with serial communication. If the FFT calculation was going to be done on MATLAB, this would require the signal data to be sent to MATLAB via serial, the data processed, and then the signal DATA sent back to the Arduino via serial again. Although this is definitely possible, previous problems had been encountered sending serial data from MATLAB to the Arduino and doing the calculation onboard removed the need to deal with it at all and greatly streamlined the process.

As mentioned before, the method used to further process the data with MATLAB was via CSV file. From the Arduino code, the raw ADC data from the signal can be written directly to a .CSV file as soon as it is collected. For each required note (c4 to c5) there is a single recorded session to create a separate .CSV file for each tone. For each of these sessions a sampling frequency of 20 times the expected frequency of the signal would be used, with a total of 256 samples (accomplished by manipulating the registers of the ATMEGA328p processor). This allows for the requirements of 20 samples per recorded period, 3 periods of data to be plotted, as well as keeping a fairly low resolution to ensure an accurate FFT. Then at a different time, the data from each file can be read into MATLAB and a separate fourier transform can be performed along with other further assessments such as the voltage vs time, harmonics, power vs frequency, and signal to noise ratio (SNR).

3 Implementation

Block 1: Instrument/Tone Source

For the purpose of this project, the tone source was fairly arbitrary. For testing purposes, a frequency generator phone application was used. Frequency generators are very simple to use and allow the designer to test many different frequencies with ease. The downside of these applications is that they are typically fairly quiet unless connected to an external speaker. For the final test of the tone detector, an unamplified acoustic guitar was used to tie the project to real-world application. The guitar was typically louder than the frequencies being output from the phone speakers which allowed for the tone detector to analyze frequencies from a further distance. However, the guitar had its flaws including being able to error check the tone being played and the note the tone detector was finding.

Block 2: Microphone with Circuitry

The microphone that was used to start the analysis of the tone from the source was an electret condenser microphone - the CMC-5042PF. In a very basic sense, an electret microphone uses a thin diaphragm made of a polarized material called an electret and a metal back plate as a capacitor. When the audio signal reaches the diaphragm, the vibrations of the audio signal move the diaphragm back and forth - changing the voltage across the diaphragm and the back plate. This changing voltage results in the audio signal that is then amplified through the rest of the circuit.

In Figure 2 below, the schematic is given for the microphone block of the project. For the purpose of this project, the microphone will be powered with a 3.3V source from the Arduino Nano, as the datasheet for the CMC-5042PF recommends an operating voltage of 2V with a maximum operating voltage of 10V. The 2.2kΩ resistor is used to limit the amount of current flow out of the microphone block - as the data sheet states the current consumption of the CMC-5042PF is 0.5 mA. The capacitor acts as a high-pass filter, getting rid of the DC bias and noise that is presented by the 3.3V source used to power the microphone block.

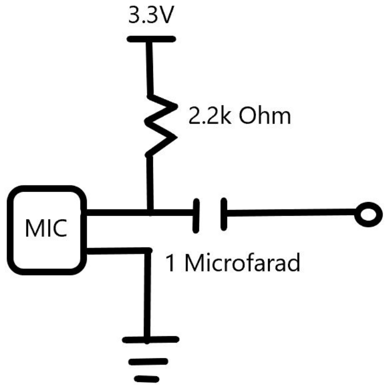


Figure 2: Schematic of the microphone circuit.

Block 3: Microphone Amp LMC6032

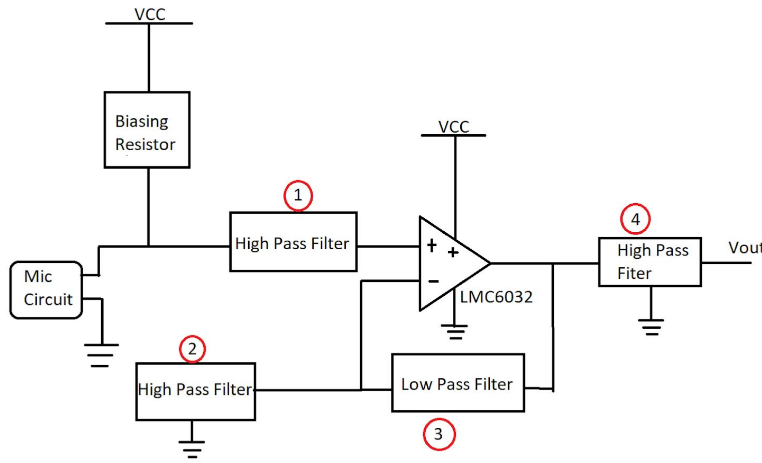


Figure 3: Block diagram of the microphone amplifier circuit with mic circuit included.

Debatably the most important portion of the project, the microphone amplifier circuit takes the input signal from the microphone and increases the amplitude to a viable range so that the signal can be analyzed. When the microphone circuit block was created in hardware, the signal from the output had a maximum voltage of 0.01V - barely high enough for the DMM to accurately read. For the purpose of this project, the signal at the output had to have a minimum of 1V amplitude during excitation. Taking an input signal from 0.01V maximum amplitude to 1V requires a gain of 100. This gain is set by the microphone amplifier circuit by manipulating resistor values surrounding the LM6032 op-amp. Through this section, resistor values will be noted by their label R1-R6 and capacitors C1-C4 which can be found in the full annotated schematic in Figure 4 below.

Figure 3 above shows the block diagram of the microphone amplifier. The purpose of this block is to both increase the input signal from the microphone and filter out some of the unnecessary noise from the signal. This amplifier circuit was given in the documentation for this project and consists of 6 main parts: a biasing resistor, 3 high pass filters (blocks 1, 2, and 4), a low pass filter (block 3), and an op-amp (the LMC6032).

The biasing resistor is used to ensure the minimum amount of current (specified by the microphone) is met, as well as ensuring the ratio of output and input voltage meets the desired gain value. This resistor is chosen such that the majority of current being output by the microphone flows through it, implying it must be of a lower resistance than the resistors used for the high pass filter (filter 1). The calculations for finding the desired value for the biasing resistors goes as follows:

$$Desired\ Gain\ \approx 130,\ V_{out(max)}\ =\ 1.228$$
$$Gain\ =\ \frac{V_{out}}{V_{in}}\ \rightarrow 130\ =\ \frac{1.228}{V_{in}}\ \Rightarrow V_{in} =\ 0.0095$$
$$Sensitivity\ =\ -42\ dB,\ Sensitivity\ \rightarrow \frac{V}{Pa}\ =\ 10^{-\frac{42}{20}} = 7.943\ \frac{mV}{Pa}$$
$$\frac{V}{Pa} \rightarrow \frac{A}{Pa}\ =\ \frac{Sensitivity\ (\frac{V}{Pa})}{Microphone\ Impedance},\ Microphone\ Impedance\ =\ 2.2k\Omega\ \Rightarrow \frac{A}{Pa}\ =\ \frac{7.493\frac{mV}{Pa}}{2.2k}\ =\ 3.6105\mu A$$
$$I_{max} = 2Pa \times Sensitivity(\frac{A}{pa}) = 7.221\mu A$$
$$V_{in}\ =\ I_{max}(R_{in}) \rightarrow 0.0095 = 7.221\mu A(R_{in})\ \Rightarrow R_{in}\ =\ 1598.125$$
$$R_{in} = R1||100k\ (100k\ from\ equivalent\ resistance\ of\ high\ pass\ filter)\ \Rightarrow \frac{R1(100k)}{R1+100k} = 1598.125\Omega$$
$$\Rightarrow R1\ =\ 1624.1k\Omega$$

For simplicity of implementation, the R1 value found above was changed to 2.2k Ω.

Filter 1 in figure 3(a) above was used without changes from the schematic given in the lab documentation. The filter consists of two 200k Ω resistors, one from Vcc and one to ground, as well as a 330nF capacitor. Figure 4 below shows the schematic for the filter described. This filter acts as a high-pass filter, with a cutoff frequency of 3.956Hz.

The op-amp used for the physical implementation of the microphone amplifier differs from that given in the documentation for this project. The LMC6032 was used, which has a recommended supply voltage range of 4.75V to 15.5V. During physical testing, it was found that supplying the LMC6032 with a lower voltage of 3.3V resulted in less noise at the output, allowing for better analysis. This lower supply voltage also resulted in an overall lower maximum output voltage amplitude, with a maximum of approximately 1.5V in implementation rather than the 2.3V maximum that was found when the op-amp was powered with 5V. With this in mind, we found that using the 3.3V supply for gathering data for the analysis to be done in MATLAB was more suitable, while the 5V voltage source was better suited for the distance requirement. Both voltage sources were used depending on the desired application of the amplifier.

The gain of the entire amplifier was set simply by two resistors (R4 and R5) found in filters 2 and 3, respectively. The LMC6032 op-amp was set in a non-inverting fashion. The gain for a non-inverting amplifier is given as $Gain\ =\ 1 + \frac{R_F}{R2}$, in which RF is the feedback resistor (R5) and R2 is resistor R4 in filter 2. For an amplifier with low noise, it is good practice to set the feedback resistance to be higher (rather than lower) and have a low filtering resistance. For the initial gain of approximately 110, an R5 value of 110k Ω amd R4 value of 1k Ω were chosen. With this in mind, during the physical application of this resistor network the output voltage resided much lower than the desired value of 1V. To reach the limit, the R4 value was decreased to a value of 220 Ω to implement a much larger gain of 501. After this switch, the voltage surpassed the required voltage of 1V with ease.

Filter 2 consists of the 220 Ω resistor in series with a 47 μF capacitor, acting as another high pass filter. Using the equation $f_c = \frac{1}{2\pi RC}$, the cutoff frequency was calculated to approximately 15Hz, letting all frequencies higher than 15 Hz pass through to the output of the amplifier.

Filter 3, on the other hand, acts as a low pass filter, with the R5 resistor of 110k Ω in parallel with the 120pF capacitor, with a cutoff frequency of approximately 12kHz in order to eliminate some of the high frequency noise created by the microphone at the output node.

Filter 4 also acts as a high pass filter that removes a majority of the DC bias found at the input terminal of the LMC6032. This allows for a voltage plot consisting of purely AC values to be analyzed. During periods in which the microphone was resting (no external excitation from the tone source), the requirement of the amplifier was to keep an AC voltage below 0.2V. The use of filter 4 allowed for this requirement to be met, while also filtering out some of the extraneous low-frequency noise that resided below the 33Hz cutoff frequency that this filter portrays.

Table 1: List of filters with corresponding resistor and capacitor values, followed by the cutoff frequencies of each of the filters.

Filter Number	**Equivalent Resistance (Ω)**	**Capacitor Value (F)**	**Cutoff Frequency (Hz)**
1	100k	330 n	3.956
2	220	47 µ	15
3	110k	120 p	12075
4	10k	3 µ	33

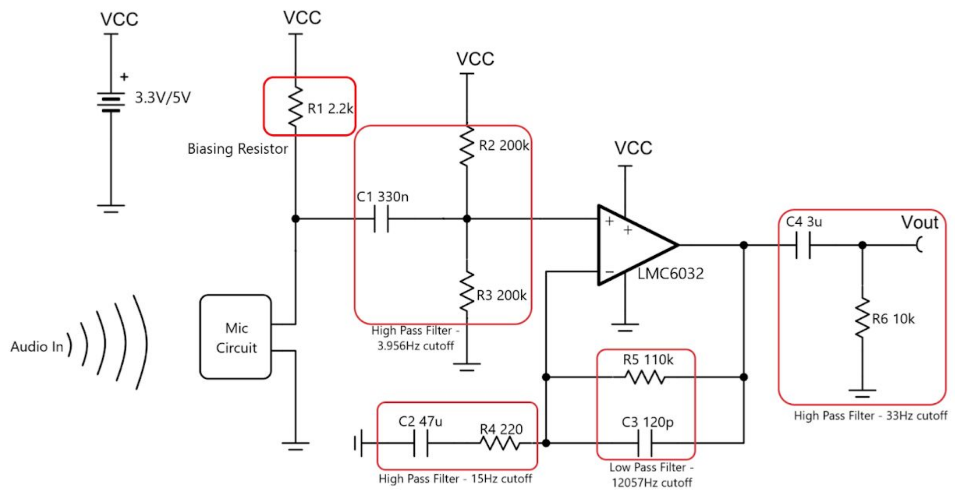


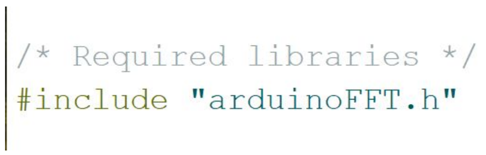
Figure 4: Schematic of the microphone amplifier with filters highlighted in red.

Block 4: Arduino Nano

The Arduino Nano microcontroller was used in this project with the following four main functions:

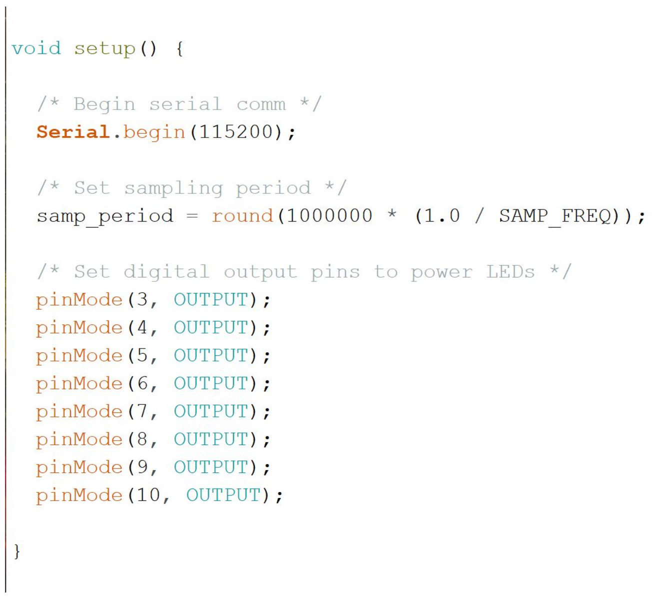
- Take in audio signal raw voltage data with the ADC
- Write raw ADC data to .CSV file
- Compute an FFT calculation of the signal and determine the frequency of the tone
- Determine which LED to power based on the calculated frequency

The detailing here will explain parts of the code but for reference the full Arduino sketch code is located in APPENDIX 1.



```
/* Required libraries */
#include "arduinoFFT.h"
```

Knowing that an FFT computation will be done, the first step of the Arduino code was to find an appropriate FFT library. A library expands on the basic Arduino sketch functionality by providing many extra functions. The FFT library that this project used was created by Enrique Condes and can be retrieved from this Github link: FFT Library. Looking at the API, the library has many functions of different computations relating the FFT. This library was retrieved from the Arduino project found here. This project's code is based around the example code given in that project by *lbf20012001*.



```
void setup() {

  /* Begin serial comm */
  Serial.begin(115200);

  /* Set sampling period */
  samp_period = round(1000000 * (1.0 / SAMP_FREQ));

  /* Set digital output pins to power LEDs */
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);

}
```

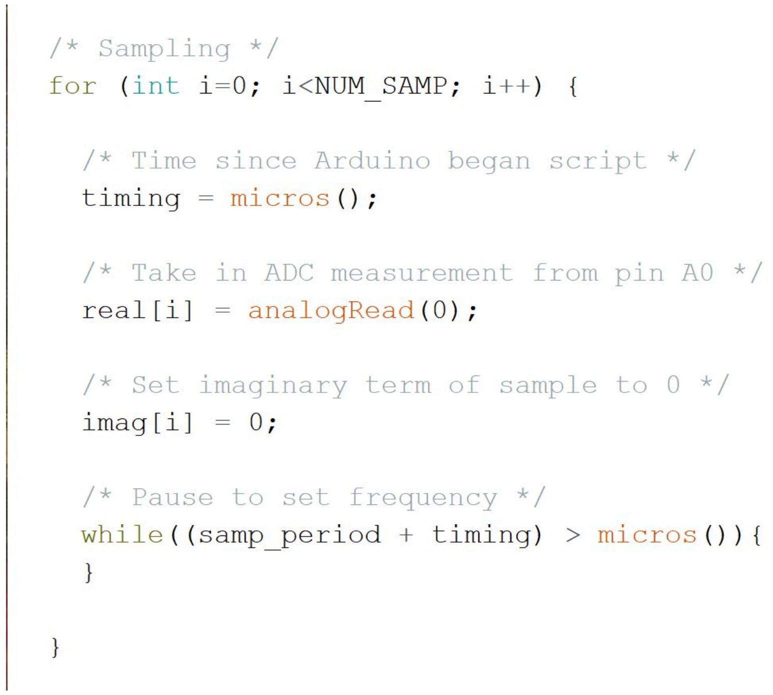
Figure 5: Example clip of the setup of the Arduino script for tone detection.

The next main aspect of the Arduino code is the setup. Knowing that the Arduino is going to be communicating via serial (either to the serial monitor or MATLAB), the Serial.begin() function

Initiates the serial communication with the chosen baud rate of 115200 which would be fast enough to export the high frequency sampling data.

In the setup, the sampling period is also set based on the inputted sampling frequency global variable. This is done by taking 1 over the given frequency and then multiplying it by 10^6 to convert the period to microseconds.This number is then rounded for simplicity as precision to the decimal number of microseconds is not needed for this application.

The last part of the setup sets the used digital pins as outputs. Configuring these pins as output with the pinMode() function allows the pin to be powered with up to 40mA of current at 5V which is more than enough to turn on the LED note indicator lights. Setting the pins to outputs can also be done via firmware programming if more control is needed (EG a pullup resistor). This was not necessary for this project so the pinMode() function was adequate.



```
/* Sampling */
for (int i=0; i<NUM_SAMP; i++) {

  /* Time since Arduino began script */
  timing = micros();

  /* Take in ADC measurement from pin A0 */
  real[i] = analogRead(0);

  /* Set imaginary term of sample to 0 */
  imag[i] = 0;

  /* Pause to set frequency */
  while((samp_period + timing) > micros()){
  }

}
```

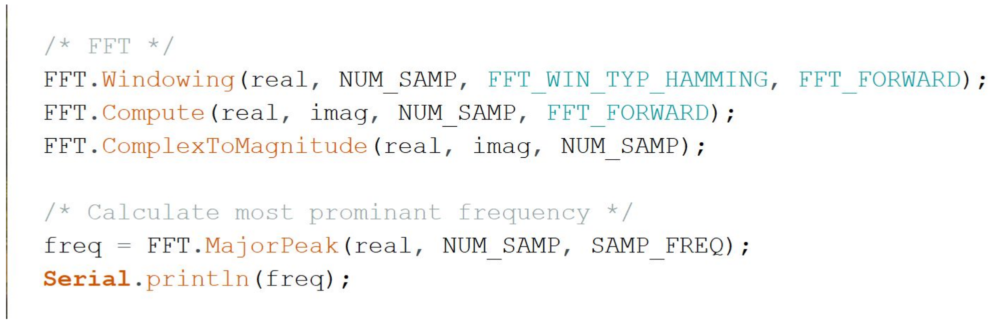
Figure 6: Main loop for taking in ADC values of the output of the microphone amplifier.

From here the main loop of the code begins. The first step is sampling the output signal from the microphone amplifier circuit. This is done in a for loop that runs until the desired number of samples are collected which is once again set as a global variable. In the loop itself it simply reads the ADC value of the analog input pin 0 and sets that to the index of the sample number in the array. Note that the FFT function used takes in complex numbers so for every sample the imaginary term of the number is set to 0, making it fully real.

The timing of the loop had to be considered to achieve the correct sampling frequency which was implemented by using pauses (in the form of a while loop). At the beginning of each iteration of the for loop, a ‘timing’ variable is set using the function micros(). This returns the number of microseconds since the program began running on the Arduino. Then after the sampling occurs for that iteration, a while loop pauses the program according to the frequency. This is done by checking if it has been the entire sampling period since the ‘timing’ variable was set earlier. If the entire sampling period has not passed, the empty while loop pauses until it has. This ensures that only one sample is recorded every sampling period.

This is not the most robust method of programming as it is assumed that the sampling itself does not take the entire sampling period. If it did and the entire sampling period had already passed once it got to the timing while loop in the code, there would be no pause and the sampling frequency would be incorrect. For this reason, the code is only truly accurate for when the sampling period is longer than how long it takes to actually sample.

Some quick research and calculation shows that the analogRead() function takes 100 microseconds to read an analog input. Therefore the maximum sampling frequency is 10KHz and anything faster the timing is off. The code still works, but the sampling period data is inaccurate.

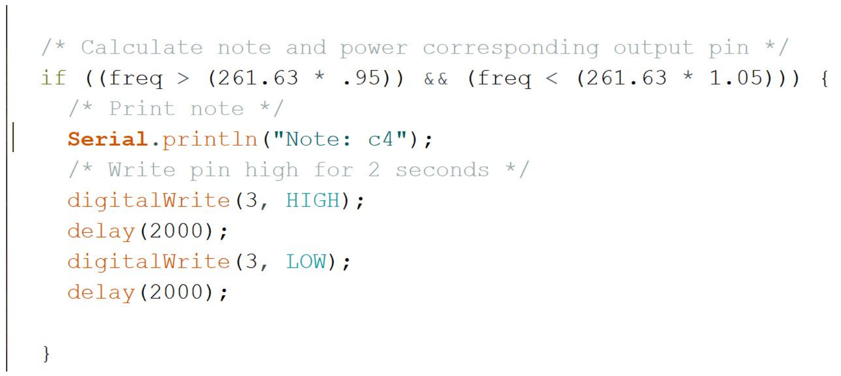


```
/* FFT */
FFT.Windowing(real, NUM_SAMP, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
FFT.Compute(real, imag, NUM_SAMP, FFT_FORWARD);
FFT.ComplexToMagnitude(real, imag, NUM_SAMP);

/* Calculate most prominant frequency */
freq = FFT.MajorPeak(real, NUM_SAMP, SAMP_FREQ);
Serial.println(freq);
```

Figure 7: Arduino script surrounding the FFT portion of the tone detection.

The next block of code in the main loop is the FFT calculation. The windowing is set automatically to a ‘hamming’ window which is a type of FFT windowing that leaves a slight discontinuity and gives a more accurate peak reading. The compute() function actually computes the FFT and the ComplexToMagnitude() calculates the magnitude of the complex results. Once the FFT is computed, a frequency variable ‘freq’ is set to the major peak (largest spike in analyzed signal) which is the dominant frequency of the signal. This value is then serial printed so that it can read from the serial monitor.



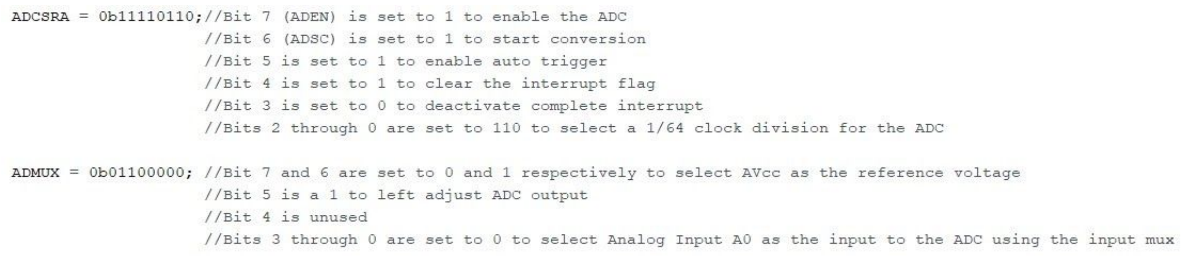
```
/* Calculate note and power corresponding output pin */
if ((freq > (261.63 * .95)) && (freq < (261.63 * 1.05))) {
  /* Print note */
  Serial.println("Note: c4");
  /* Write pin high for 2 seconds */
  digitalWrite(3, HIGH);
  delay(2000);
  digitalWrite(3, LOW);
  delay(2000);

}
```

Figure 8: Example of the classification of the note found by the FFT and illuminating corresponding LEDs.

The final block of code in the main Arduino loop is the note detection and powering the LED circuit. The if statement above is repeated for each note to detect if it is within the frequency range. If the note is detected to be within the range, the name of the note detected is serial printed and then the corresponding output pin is powered for 2 seconds.

A separate but related Arduino sketch was used to sample tones and print the data to .CSV files to be used in MATLAB. This sketch was much simpler than the one used for lighting the LEDs. It needed to have a higher sampling frequency so that it could get 20 samples per second for the project requirement. The code, shown in Appendix 2, takes in the same data as the other arduino code, but it manipulates the ADC using registers to maximize speed. The code used to set the registers of the ADC is shown below.



```
ADCSRA = 0b11110110;//Bit 7 (ADEN) is set to 1 to enable the ADC
                    //Bit 6 (ADSC) is set to 1 to start conversion
                    //Bit 5 is set to 1 to enable auto trigger
                    //Bit 4 is set to 1 to clear the interrupt flag
                    //Bit 3 is set to 0 to deactivate complete interrupt
                    //Bits 2 through 0 are set to 110 to select a 1/64 clock division for the ADC

ADMUX = 0b01100000; //Bit 7 and 6 are set to 0 and 1 respectively to select AVcc as the reference voltage
                    //Bit 5 is a 1 to left adjust ADC output
                    //Bit 4 is unused
                    //Bits 3 through 0 are set to 0 to select Analog Input A0 as the input to the ADC using the input mux
```

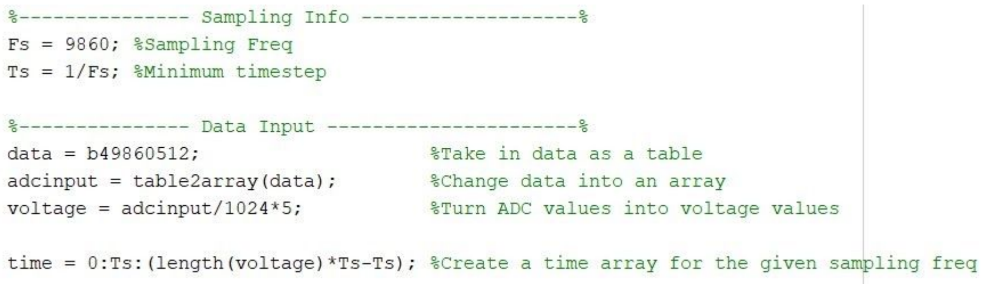
Figure 9: Register adjustments made for taking in voltage data to be output into a .CSV file.

Most of the bits of both ADCSRA and ADMUX were set to their default values, but bits 2 through 0 of the ADCSRA register were set to 110 to set the ADC clock speed to 1/64 of its original speed. This is to increase the ability to sample at higher frequencies. Also, rather than serial printing the voltage value during every sampling period, the data is gathered in an array and serial printed all at once at the end of the code. The serial monitor can then be used to copy voltage data into a CSV file for plotting in MATLAB.

Block 5: MATLAB

The MATLAB script for this project is used to plot voltage over time data read from a CSV file generated by the Arduino taking input from the microphone amp. After the voltage is plotted, the power vs frequency plot is generated using MATLAB’s fft() function. Finally, the signal to noise ratio (SNR) is calculated and plotted using MATLAB’s snr() function.

For each note, the CSV file generated is loaded into MATLAB in the form of a table. That table is then turned into an array. The sampling frequency of the given CSV file is also taken in from the Arduino code and loaded into MATLAB. The code for the above mentioned processes along with the process of creating the time scale is shown in the figure below.



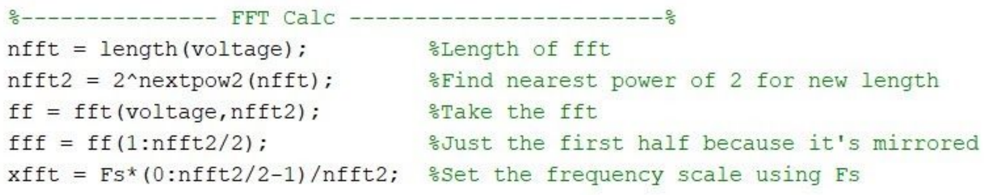
```
%--------------- Sampling Info -------------------%
Fs = 9860; %Sampling Freq
Ts = 1/Fs; %Minimum timestep

%--------------- Data Input ----------------------%
data = b49860512;                   %Take in data as a table
adcinput = table2array(data);       %Change data into an array
voltage = adcinput/1024*5;          %Turn ADC values into voltage values

time = 0:Ts:(length(voltage)*Ts-Ts); %Create a time array for the given sampling freq
```

Figure 10: MATLAB code for taking in data from CSV file

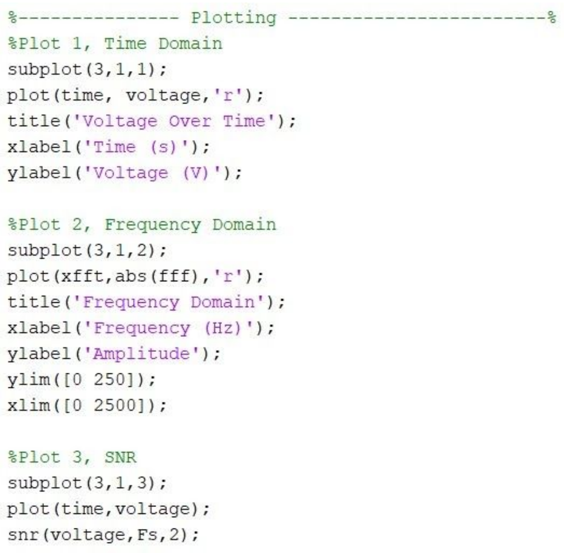
When taking in data, the data variable has to be changed to the name of the CSV file and Fs has to be changed to the sampling frequency of the data in the CSV file. The code will automatically turn the ADC values in the CSV file into voltage values. The code also calculates the sampling period and creates a time vector against which the voltage values can be plotted. Once the data has been transferred to MATLAB, the fft() function is used to find the power vs frequency distribution of the voltage data. The code for the FFT section is shown in the figure below.



```
%--------------- FFT Calc ------------------------%
nfft = length(voltage);          %Length of fft
nfft2 = 2^nextpow2(nfft);        %Find nearest power of 2 for new length
ff = fft(voltage,nfft2);         %Take the fft
fff = ff(1:nfft2/2);             %Just the first half because it's mirrored
xfft = Fs*(0:nfft2/2-1)/nfft2;   %Set the frequency scale using Fs
```

Figure 11: MATLAB code for taking the FFT of voltage data

After the FFT has been taken, the SNR of the voltage data is taken and all of the information is plotted. The voltage over time data is plotted, the power vs frequency data is plotted, and the SNR plot is plotted. All three of these subplots are placed into one figure. The code for this section is shown in the figure below.



```
%--------------- Plotting ------------------------%
%Plot 1, Time Domain
subplot(3,1,1);
plot(time, voltage,'r');
title('Voltage Over Time');
xlabel('Time (s)');
ylabel('Voltage (V)');

%Plot 2, Frequency Domain
subplot(3,1,2);
plot(xfft,abs(fff),'r');
title('Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
ylim([0 250]);
xlim([0 2500]);

%Plot 3, SNR
subplot(3,1,3);
plot(time,voltage);
snr(voltage,Fs,2);
```

Figure 12: MATLAB code for plotting data and taking SNR

4 Testing

For this section of the report, each block of the block diagram was tested or simulated with mock data to see how that particular block functions on its own. Then, all the blocks can be brought together to create the full circuit.

Block 1: Instrument/Tone Source

To test the tone source, all of the necessary notes were played with the signal generator app on a cellphone. They all worked.

Block 2: Microphone with Circuitry

When the microphone amplifier was first built with the original gain of 110, the functionality of the microphone with attached circuitry was tested using a DMM to measure the microphone's output current. In this stage of the design, it was found that the microphone was not working properly, with a much lower current flow than the calculated maximum current flow in Section 3 above. This lack of current flow was taken into account as a malfunction by the microphone, giving reason behind the lack of output voltage the microphone amplifier with a gain of 110 was producing. After manipulating the gain to a much higher value of 501, the microphone was able to be used to output a voltage greater than 1 volt in amplitude. This circumvention of the microphone problem by raising the overall gain of the amplifier (though not ideal) allowed for sufficient analysis for the purposes of this project.

Block 3: Microphone Amplifier

To simulate the workings of the designed microphone amplifier circuit, LTSpice was used. Two different simulations were run; one pertaining to finding the desired gain of the amplifier, and one to represent the filter design of the circuit.

For the gain simulation, an AC current source with 7.221 μA current at 440Hz was simulated. This AC current source is used to mimic the workings of the microphone circuit block working at its maximum output current with a frequency that's in the range of notes to be analyzed (A4). Both the amplifier and the circuitry in this simulation is powered by a 5V source to correspond to the 5V source from the Arduino Nano.

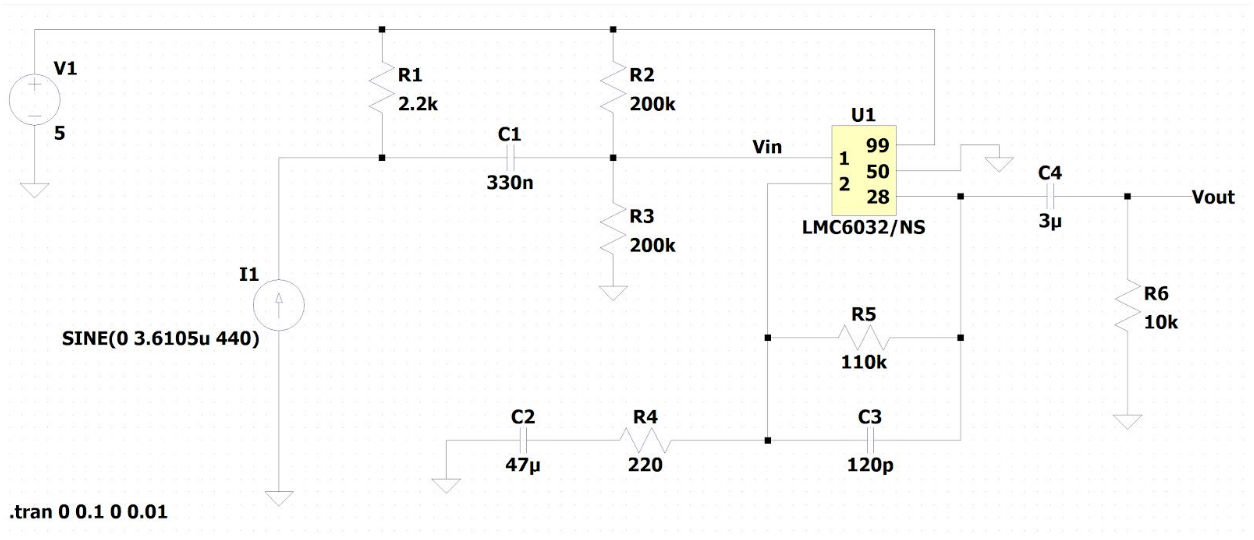


Figure 13: Schematic in LTSpice of the microphone amplifier design.

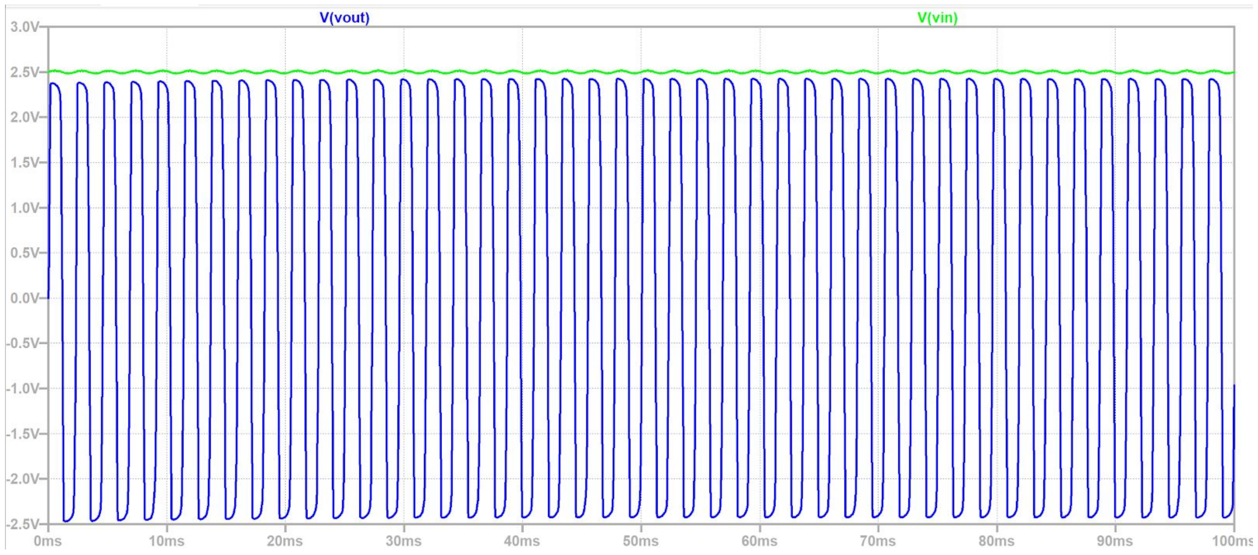


Figure 14: Plot of input and output voltage over time for circuit gain analysis.

Analyzing Figure 13, it can be found that the biasing resistor (R1) biases the input of the amplifier to a voltage of 2.5V. The amplitude of the input signal can be found to be 0.0303V, which is far too small to be analyzed by the Arduino Nano accurately. The output voltage was taken after the 3 µ F capacitor C4 to bias the output back to 0. Had the analysis taken place on the left side of C4, the signal would be biased at the input voltage of 2.5V. The voltage plot for the output, however, has a somewhat peculiar shape which is much more aggressive than that of the input signal. This may be attributed to limitations to the LMC6032, implying that the output voltage is "clipping" before it can hit its maximum amplitude. With the fault found in the microphone used, it was believed that this clipping would not be experienced during the physical implementation and was thoughtfully overlooked for the time being. The amplitude of the simulated output signal was calculated to be 4.796V, as the maximum and minimum voltage values were 2.429V and -2.429V, respectively. Taking the gain to be the ratio of output voltage over input voltage, it was found that the gain of the simulation was 158.3 $\frac{V}{V}$.

$$Gain = \frac{Output\ Voltage}{Input\ Voltage} = \frac{4.796}{0.0303} \approx 158.3$$

By using the frequency sweep command in LTSpice, the effects of the four filters in the microphone amplifier could be visualized. The command used in LTSpice tested frequencies from 0.1Hz to 16kHz to fully encompass the expected cutoff frequencies each of the filters has. Figures 15 and 16 below show both the schematic used in LTSpice as well as the frequency analysis.

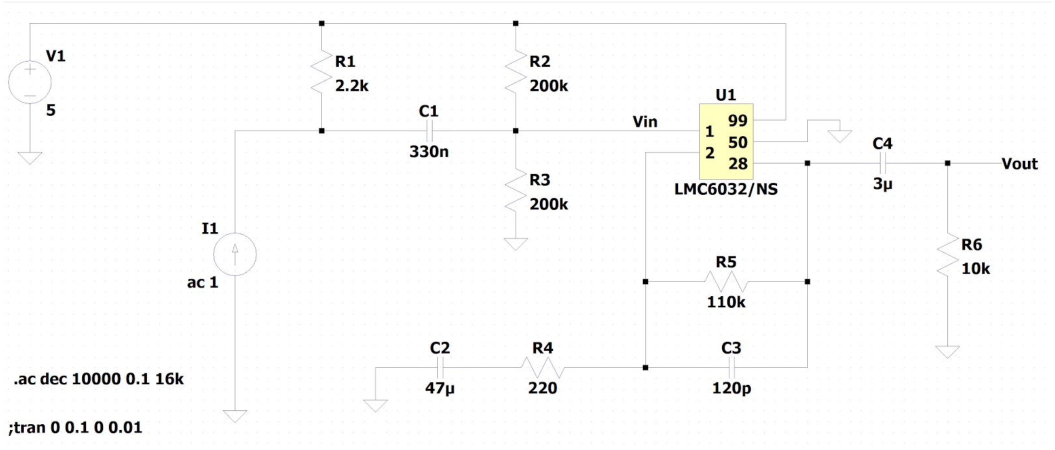


Figure 15: Schematic used for the frequency analysis of the microphone amplifier.

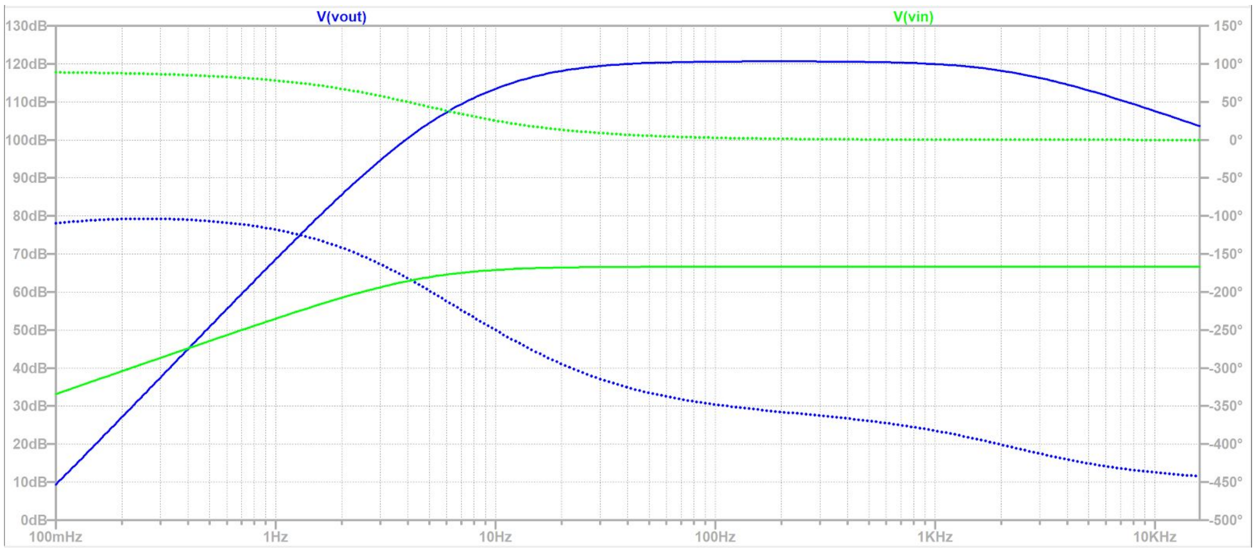


Figure 16: Filter analysis of the microphone amplifier.

Figure 16 shows the frequency response of both the input and output of the amplifier design. As apparent in the frequency response of the input node, filter 1's cutoff frequency limits the effect of frequencies below approximately 4 Hz, while the rest of the frequencies can pass through. Analyzing the output frequency response, it is apparent that filter 2's cutoff frequency of 15 Hz allows for signal frequencies above the cutoff to pass to the output, while all frequencies above 1kHz begin to taper off.

Block 4: Arduino Nano

The Arduino code was tested to have the proper sampling frequency by recording the time of micros() before the sampling begins and then checking how long it took to sample by using micros() after sampling is complete. This time is divided by the number of measurements and checked against the sampling frequency. Using this method, it was found that the Arduino code had the ability to sample up to 18kHz.

Block 5: MATLAB

To test the MATLAB code, a simple signal containing two sine waves and 0.2 volts of random noise was put through the FFT and SNR calculations. The frequencies of the two sine waves were 261 and 440 Hz, respectively, and their amplitudes were each 0.5. The signal was biased at 2.5V. The following figure shows the results of the FFT and SNR calculations for that signal.

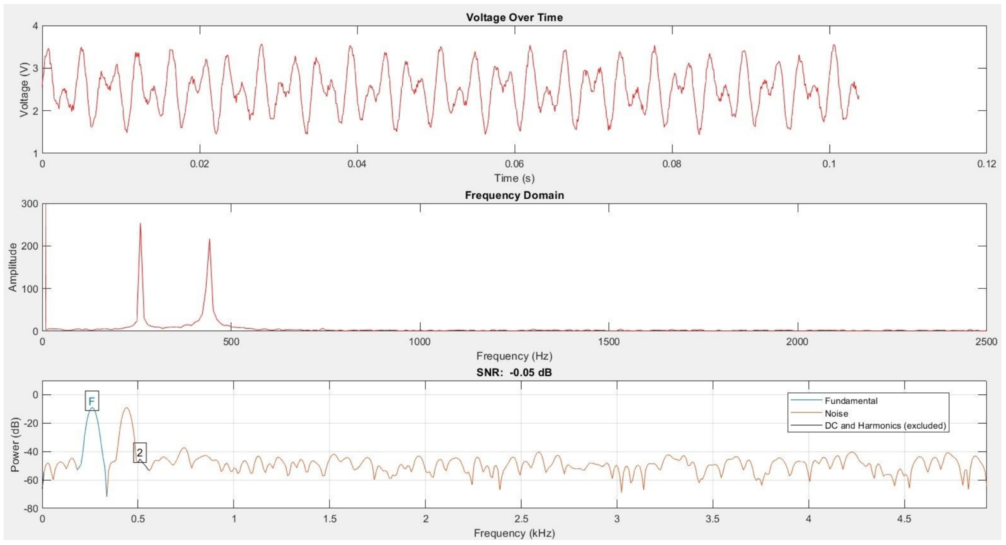


Figure 17: MATLAB FFT and SNR results for Test Signal

As can be seen from the figure above, the FFT analyzes the signal and picks out very close to the proper frequencies. It got 259 Hz instead of 261 and 442 Hz instead of 440. This is within the 5% interval that it needs to be, so this FFT code checks out. And the SNR is non zero, showing that there is noise.

5 Results

The microphone circuit was built in a small protoboard according to the schematic in figure 2. A picture of the finished circuit can be seen in the figure below. The inputs, outputs, and components are labeled.

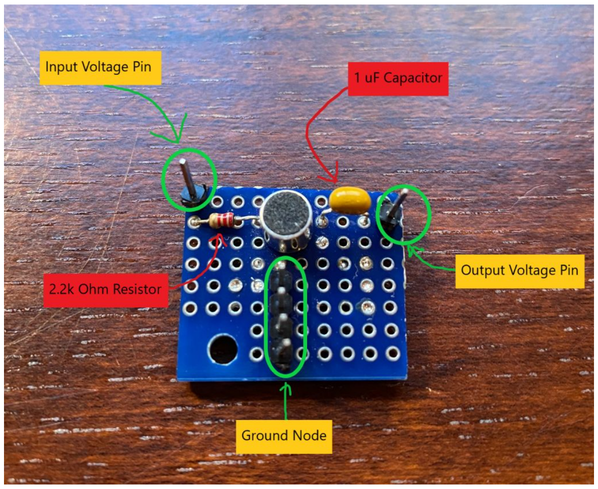


Figure 18: Microphone Circuit

The microphone amplifier circuit was also assembled according to the schematic, but in a larger protoboard. The completed circuit, with inputs, outputs, components, and filters labeled.

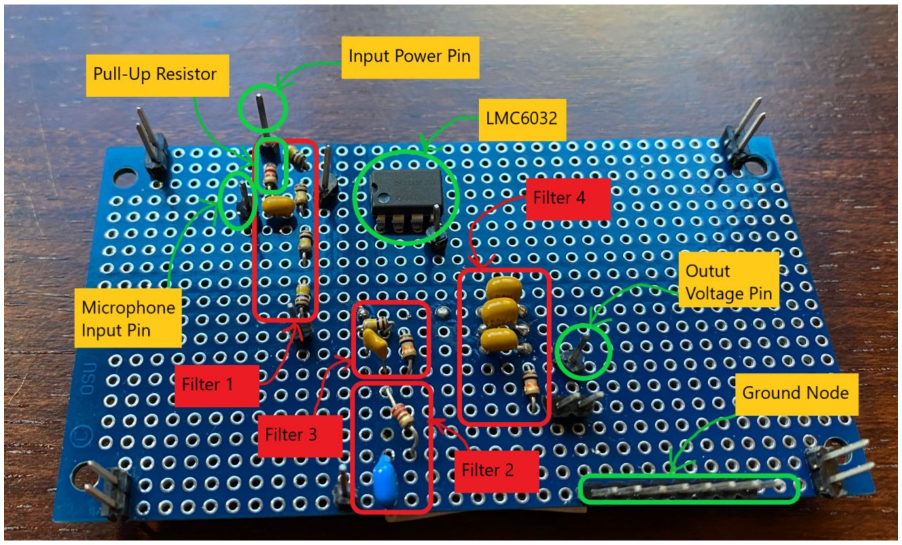


Figure 19: Microphone Amplifier Circuit

The microphone, amplifier, Arduino, and LEDs were set up according to the block diagram. The figure below shows the microphone circuit and the microphone amplifier in one circuit together.

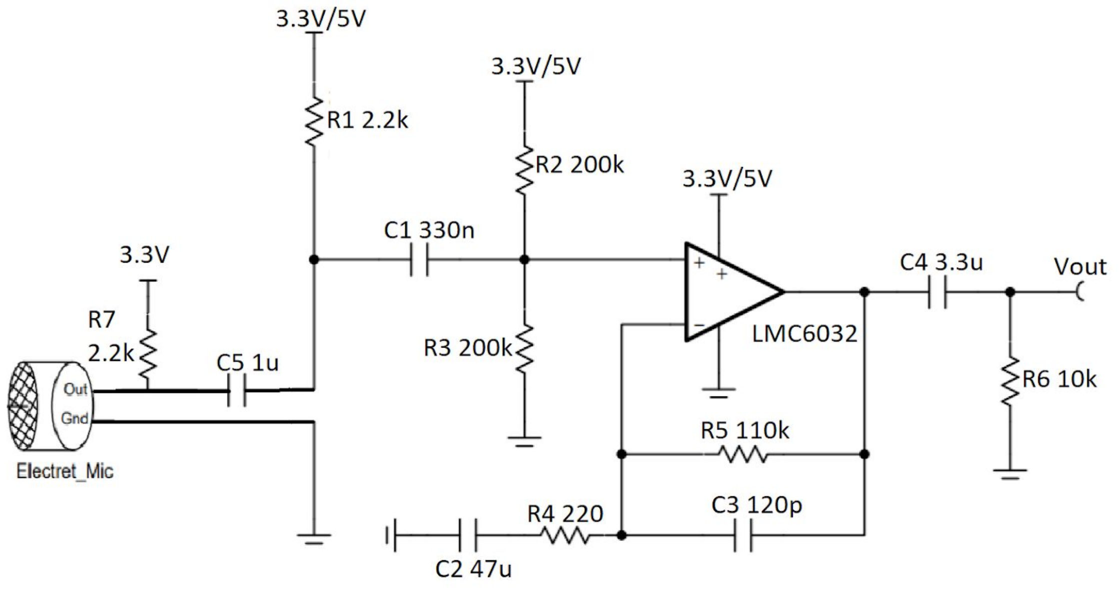
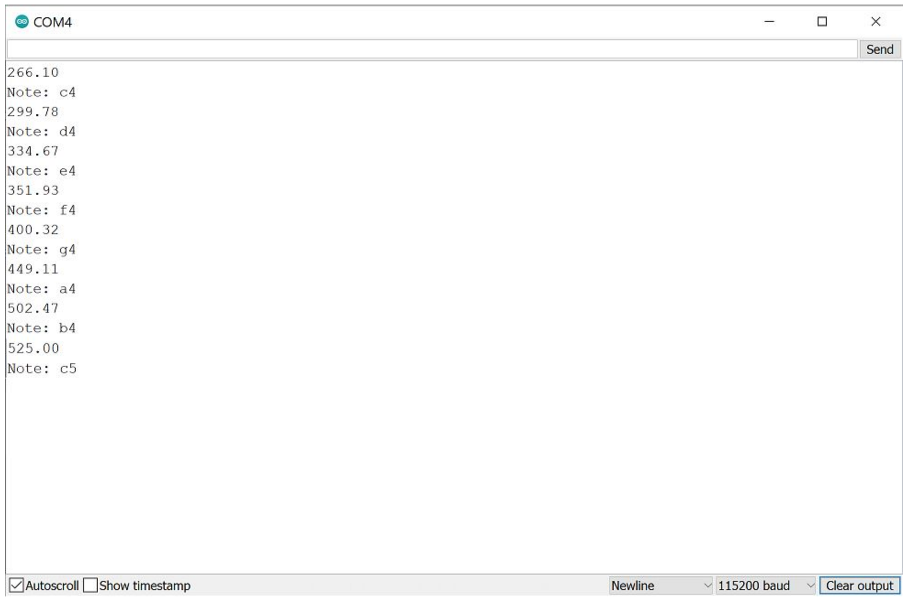


Figure 20: Finished Microphone Circuit Schematic

Once everything was set up, the Arduino code was loaded onto the Arduino board and the circuit started receiving information. The tone source (cellphone with a signal generator) was held 10ft away from the microphone and set to play a C4 (261 Hz). After the Arduino gave its output to the serial monitor, the button on the Arduino was pressed to start the code again, and the next note was played. This process was repeated for all of the notes up to C5. The serial output of the Arduino for this test of all notes is shown in the figure below. The following figure shows that the LED corresponding to C4 lights up when a C4 note is played by the signal generator.



```
266.10
Note: c4
299.78
Note: d4
334.67
Note: e4
351.93
Note: f4
400.32
Note: g4
449.11
Note: a4
502.47
Note: b4
525.00
Note: c5
```

Figure 21: Serial Monitor Output for Circuit

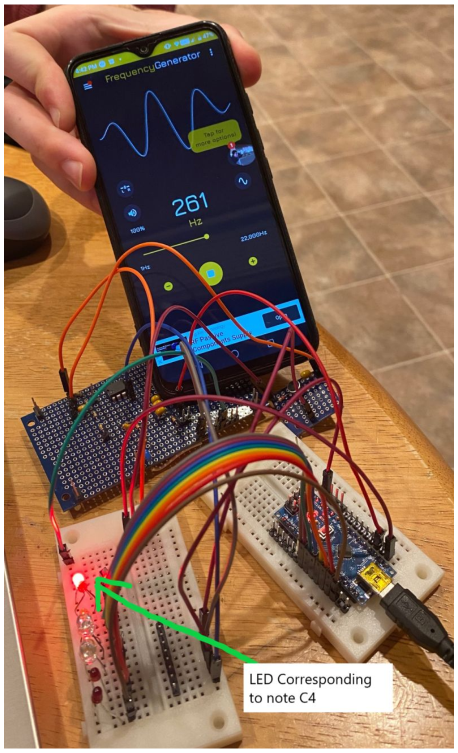


Figure 22: Validation of the LED circuit function with a note of C4.

After the circuit was proven to detect the proper notes from 10 ft away and light up the correct LED, the circuit was used to take in some data purely for the purpose of data analysis. The Arduino code with the faster sampling rate (shown in Appendix 2) was loaded onto the Arduino and the same test was done as above with all 8 notes. Except this time the phone was held at only 1 inch from the microphone in order to be able to see what note was being played. And the sampling frequency for each note was set to 20 times the frequency of the note (to reach the 20 samples per period requirement).

The CSV files containing voltage data for each note were loaded into MATLAB and then the following 8 figures were created, one for each note. Each figure contains 3 plots. The first plot is the voltage over time data for the note being played into the microphone. The second plot shows the power vs frequency plot generated by an FFT. The third plot also shows the frequency data that it uses to calculate the SNR.

Important data points are labeled on these figures, like the frequencies of harmonics and the SNR. For convenience sake, there is also a table showing all of this important data. The table is shown following the 8 figures.

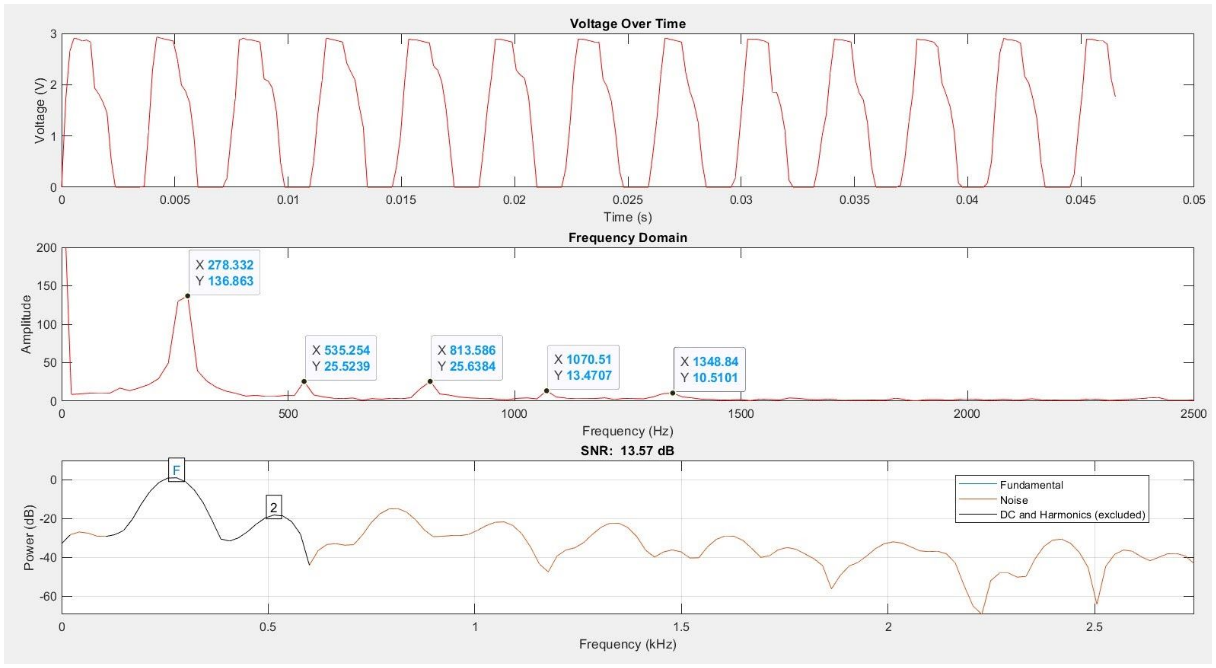


Figure 23: C4 Note Voltage, FFT, and SNR plots

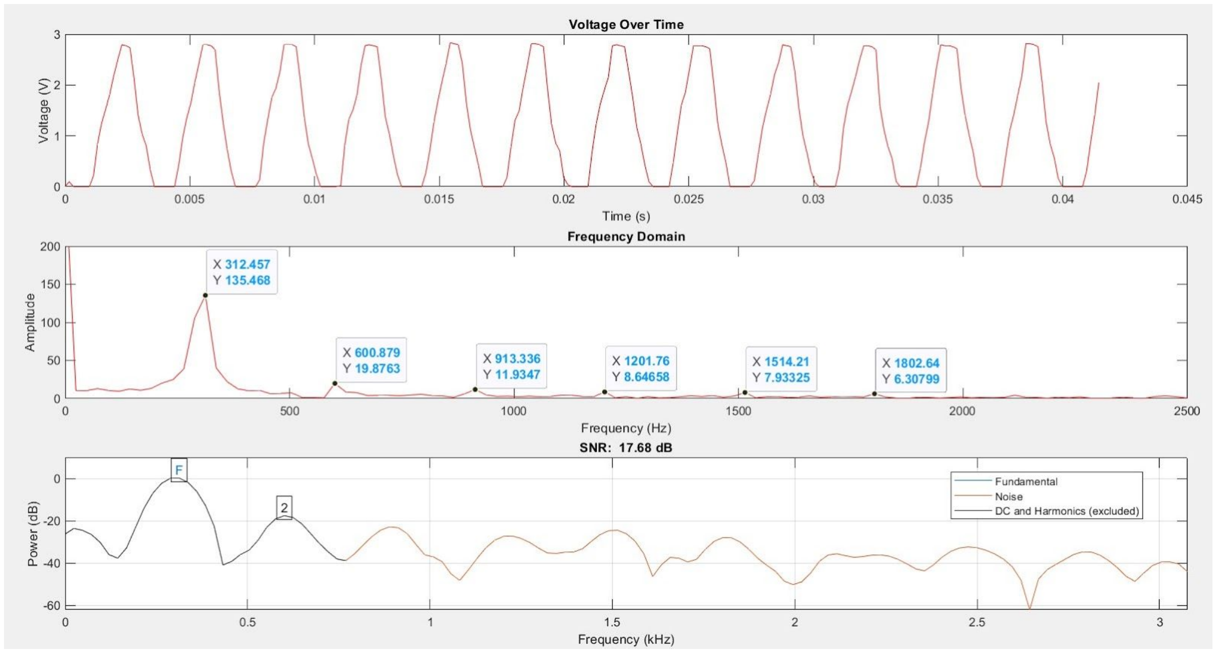


Figure 24: D4 Voltage, FFT, and SNR plots

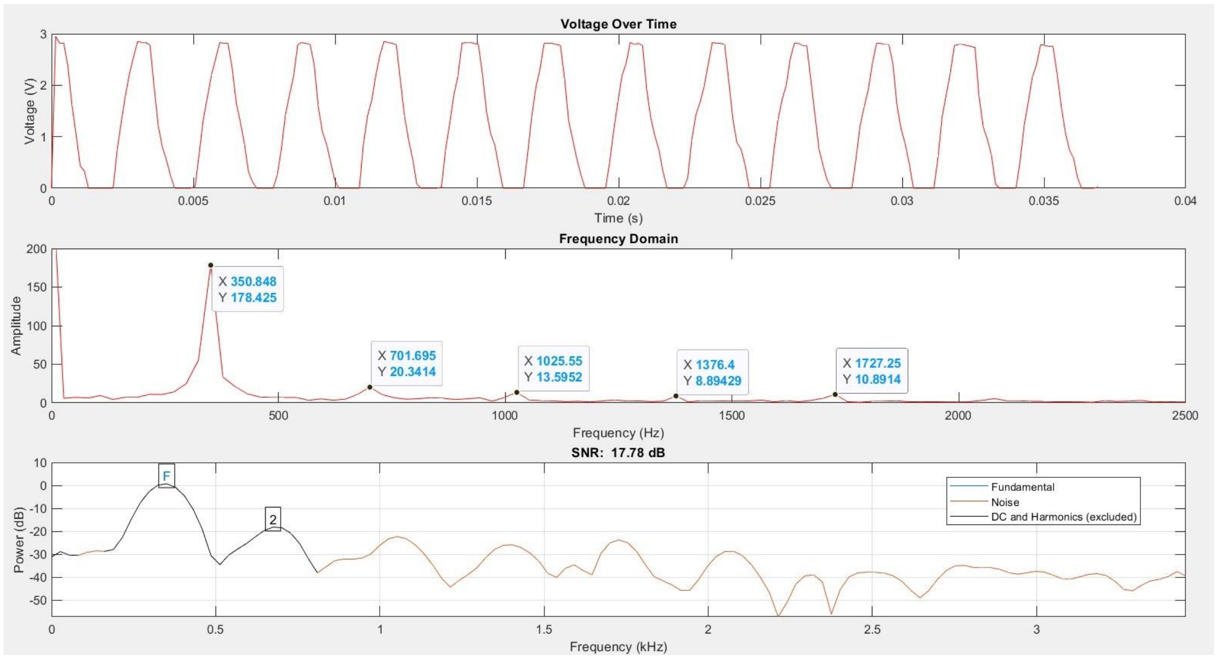


Figure 25: E4 Voltage, FFT, and SNR plots

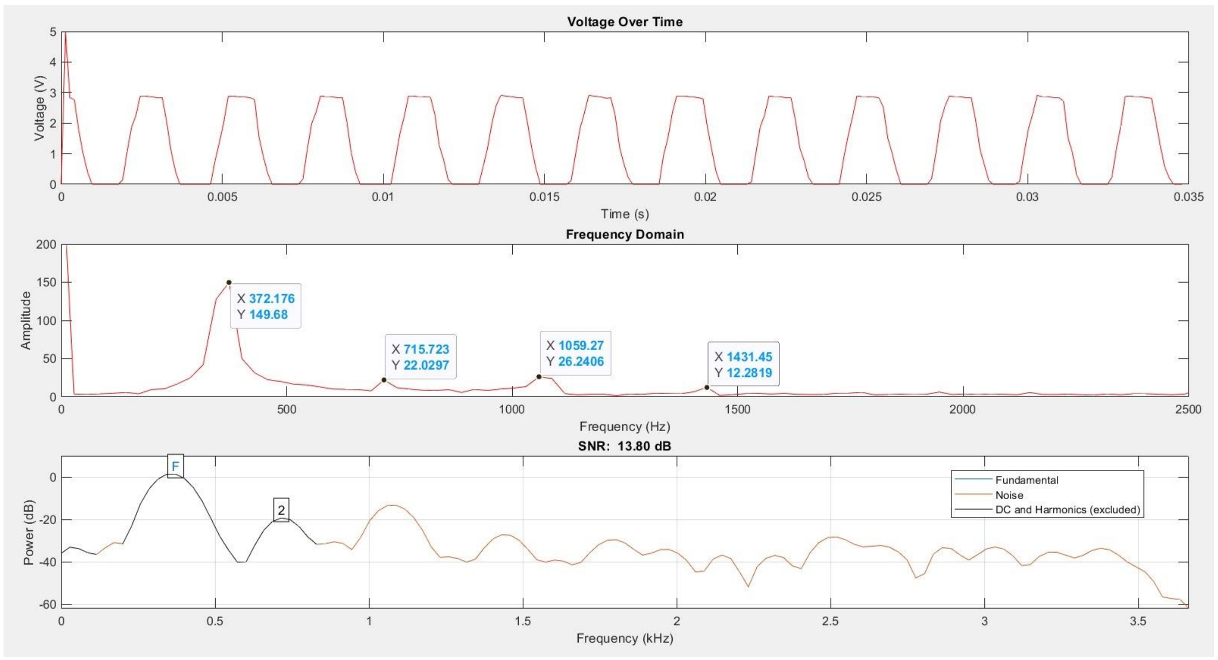


Figure 26: F4 Voltage, FFT, and SNR plots

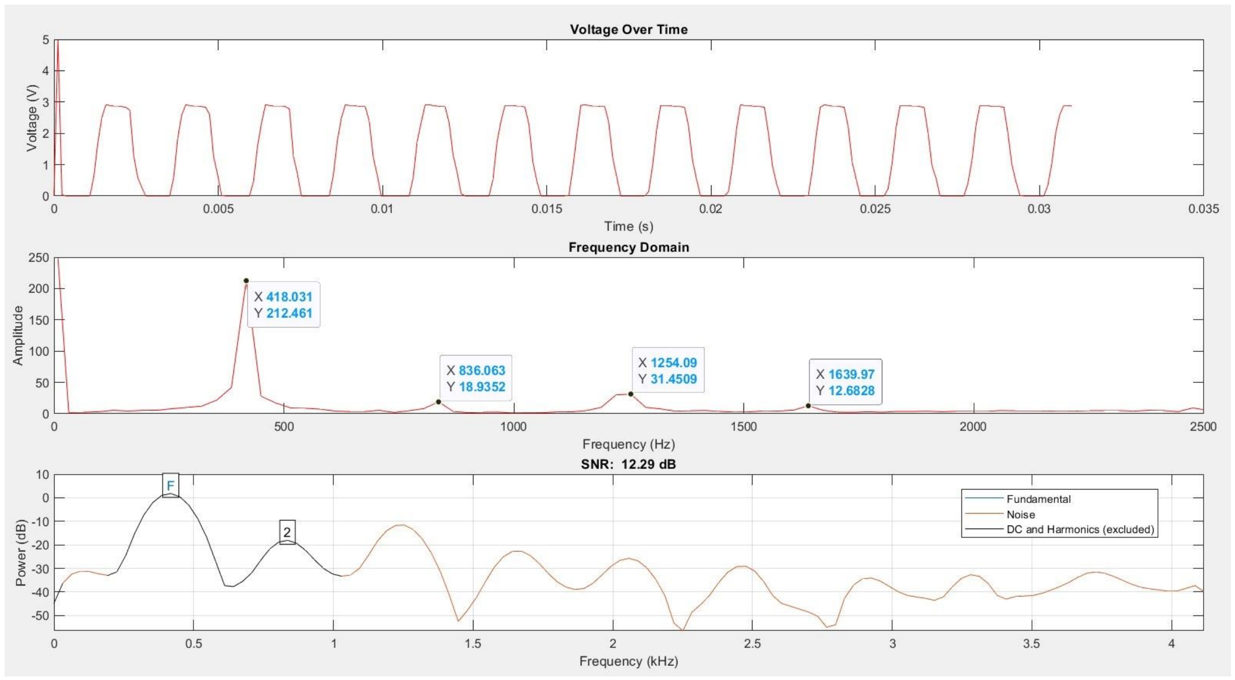


Figure 27: G4 Voltage, FFT, and SNR plots

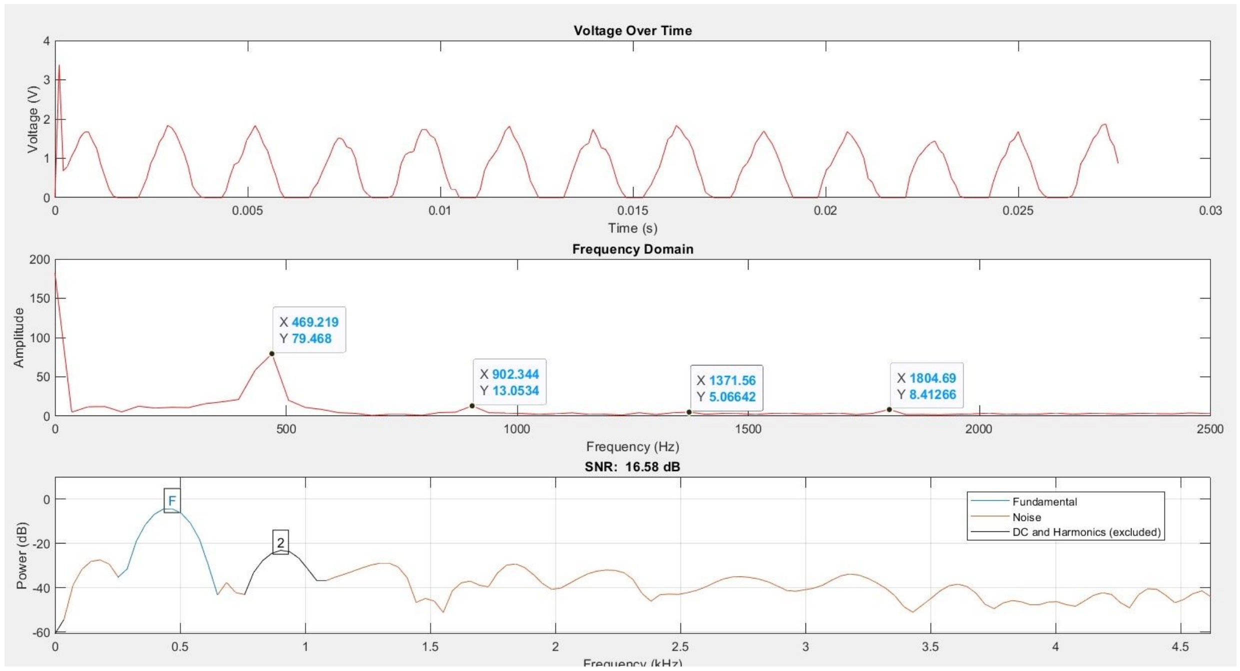


Figure 28: A4 Voltage, FFT, and SNR plots

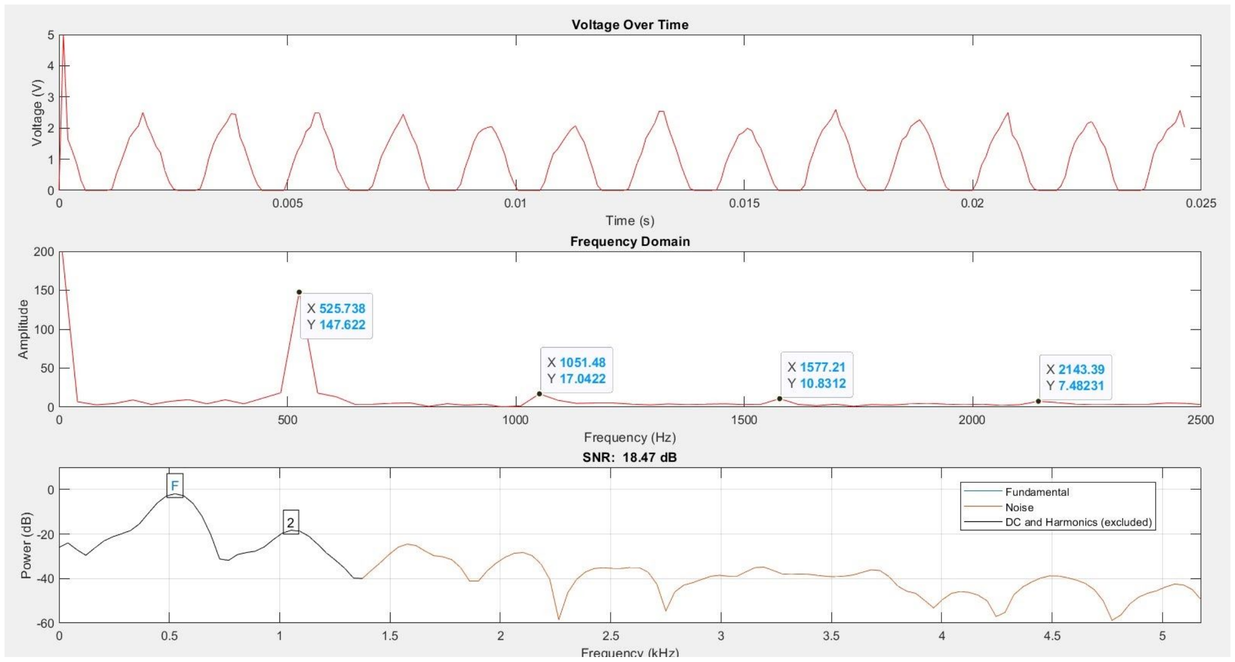


Figure 29: B4 Voltage, FFT, and SNR plots

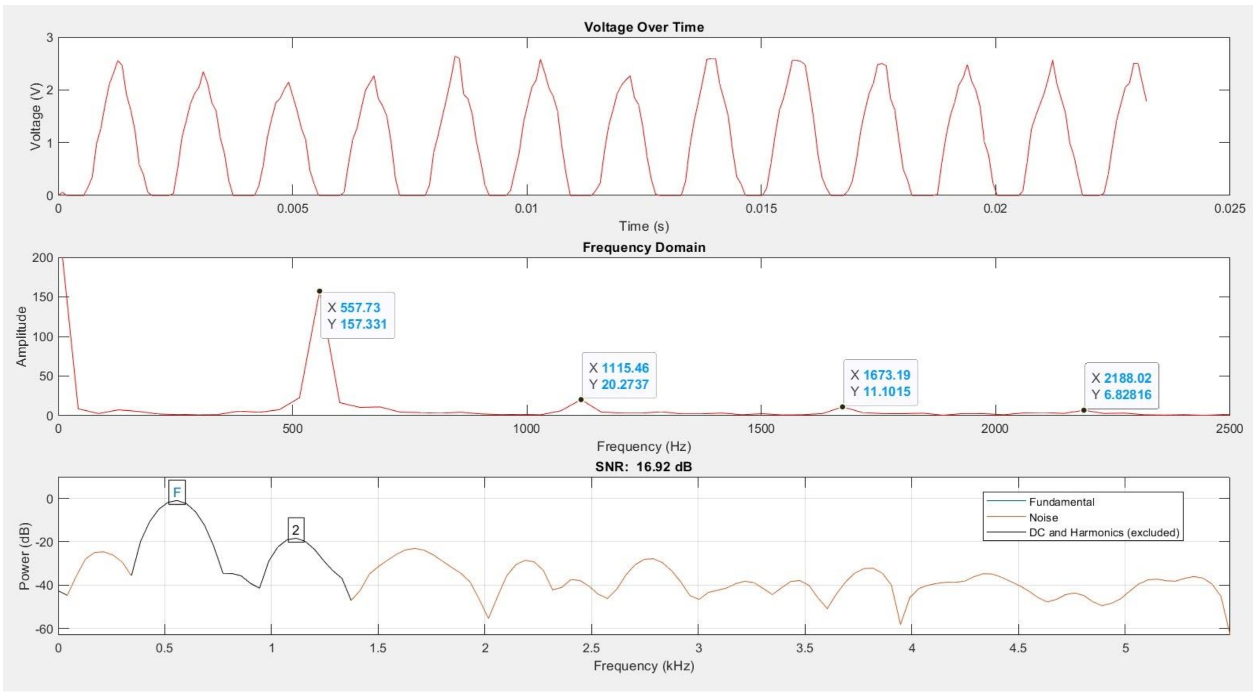


Figure 30: C5 Voltage, FFT, and SNR plots

Table 2: Final Results for All Notes

Note	Actual Frequency (Hz)	Measured Frequency (Hz)	2nd Harmonic (Hz)	3rd Harmonic (Hz)	4th Harmonic (Hz)	Max Voltage Amplitude (V)	SNR (dB)
C4	261.63	278.33	535.25	813.59	1070.51	3.00	13.57
D4	293.66	312.46	600.88	913.34	1201.76	2.80	17.68
E4	329.63	350.85	701.70	1025.55	1376.40	2.90	17.78
F4	349.23	372.18	715.72	1059.27	1431.45	2.90	13.8
G4	392.00	418.03	836.06	1254.09	1639.97	3.00	12.29
A4	440.00	469.22	902.344	1371.56	1804.69	2.00	16.58
B4	493.88	525.74	1051.48	1577.21	2143.39	2.60	18.47
C5	523.25	557.73	1115.46	1673.19	2188.02	2.70	16.92

Many of the frequencies calculated by the FFT do not match up perfectly with the actual frequency of the note. In most cases, it was found that this was merely an issue with binning. For example, D4 was measured at 312 Hz when the actual is 293 Hz. But the 2nd harmonic, which would be twice the original frequency was 600. Which puts the original at 300, much closer to the actual. The Arduino was at its maximum number of samples, so the resolution of the FFT could not go any lower without sacrificing the 20 samples per period requirement.

The maximum voltages measured were all much higher than expected. It might have to do with the gain of 500 in the microphone amp circuit or the fact that the tone source was held directly in front of the microphone. But it did not affect the data much. Just a larger signal. The SNR values calculated were all below the required SNR of 20, but many of them were very close. It might have something to do with the fact that the voltage input was not given a DC bias before being input into the ADC. The ADC cannot read negative voltages, so the signal was clipped at zero, possibly causing more noise.

6 Conclusion

Overall this project was successful. The tone detector worked correctly, detecting frequencies from 10ft and outputting above a 1V signal.

Each block of the circuit was designed, tested, prototyped, then implemented. The electret microphone and amplifier circuit were drawn out as schematics, with each block testing and verified individually on LTSpice before testing and verifying the entire circuit again on LTSpice. From here the circuits were built physically on a breadboard and tested with a DMM and once verified, they were soldered into protoboards.

Code for the Arduino was written bit by bit debugging along the way until it reached the desired functionality of detecting the tone with an FFT calculation and powering the correct LED. Once the code was verified working for test data, the circuit and the Arduino were connected together and the whole system was confirmed working together.

The MATLAB code was written to take in data from the Arduino code, and then the SNR and FFT functions were implemented based on their documentation. It was tested with simulated signals, then given some data from an actual test to confirm that it worked correctly.

After the entire design was verified and the analysis completed it was time to compile the documentation done along the way into this final report.

Project Timesheet Data

Throughout the project the team recorded the approximate amount of time spent working per day. Our team's overall communication and collaboration worked extremely well, with no problems to report. Much of the time spent working on the project was done in person together at one of the three member's houses. The team being comfortable with in-person meetings was an advantage as completing this project 100% virtually would likely not have went as smoothly.

Table 3: Time sheet of hours worked on the project.

Date	Kyle (hrs)	Samuel (hrs)	Miles (hrs)	Task
10/21	0.5	0.5	0.5	Getting familiar with project and requirements
10/22	1	1	1	Building microphone circuit
10/29	5	5	5	Troubles with gain, but simulated some working designs on LTspice.
10/30	1	1	1	
11/3	3	3	3	Mic amp gain and building and testing and circuit. Error in gain?
11/10	3	3	3	"first draft" of mic amp on protoboard and testing
11/12	2	2	2	
11/19	4	4	4	Arduino FFT and FFT background
11/22	3	3	3	Arduino IDE progress - voltage v time, note accuracy. MATLAB analysis
11/24	4	3	3	^
11/25	2	2	2	Arduino sampling improvements
11/27	1	1	1	^
11/28	2	4	4	^

11/29	3	3	3	Final signal sampling for MATLAB analysis (FFT, SNR, Voltage over time)
11/30	2	2	2	^
12/1	2	3	2	Final Report
12/2	2	3	3	^
12/3	1	2	1	^
12/4	4	3	4	^
Total	45.5	48.5	47.5	

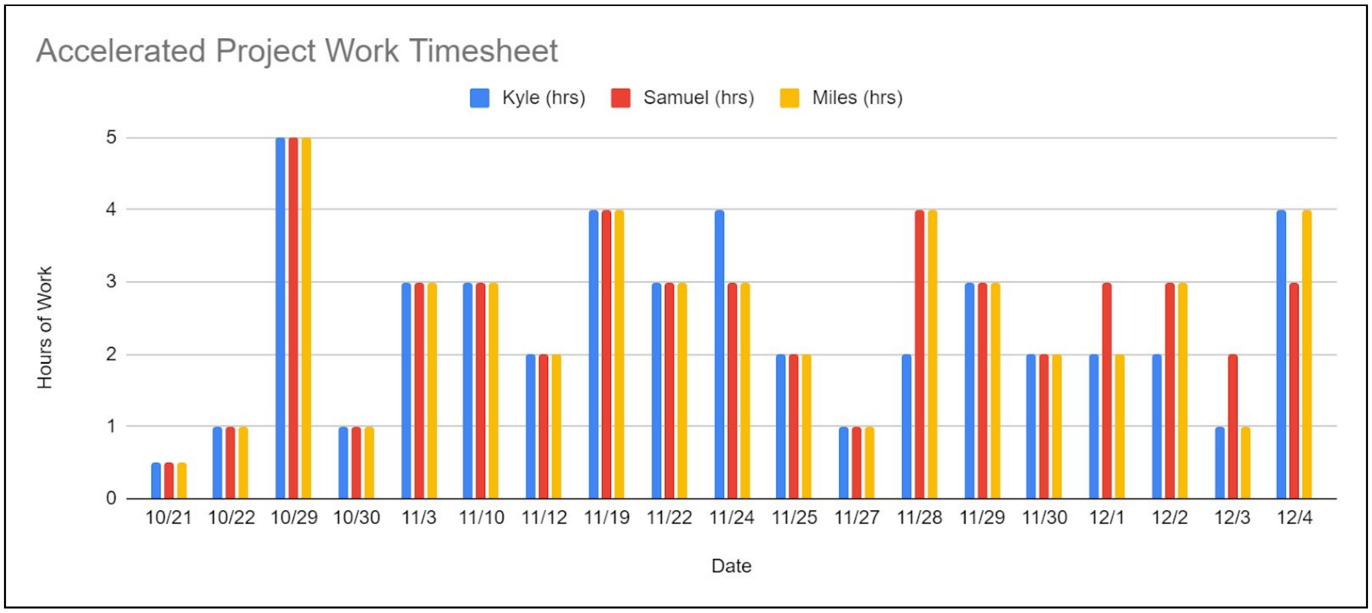


Figure 31: Work Time Graphs

As seen in the datasheet above, every member of the group put in a fairly equal amount of time on the project which was somewhere around 47 hours total. Over the course of 5 weeks this is a reasonable amount of time which resulted in a successful project. The team could have been more efficient with time as small problems often caused large hold ups, with hours or even days spent on fine details.

The graph compiled from the timesheet shows that a relatively even amount of time was spent throughout the project from start to finish. The team got an early start which helped to even out the time and reduce the 'crunch time' intensity.

Improvements

The system worked, but it was not perfect. Below are a few improvements that could be made if the project was replicated.

A simple improvement that could be made was better circuit layout design and physical construction. The protoboards worked well, but lots of jumper wires and breadboards were still required for testing which often proved problematic. Loose connections took sometimes 15 minutes to troubleshoot which was all time wasted. Once the circuit is tested and verified, a well thought out protoboard or ‘workbench’ board with easy access and sturdy testing points would be very helpful.

Another aspect that could be improved upon is the noise of the system as a whole. Our team made an effort towards this by separating clean and dirty ground, and using clean power sources, but the end system was still noisy and it could be seen in the data. Further detecting and isolation where the noise is coming from would give a cleaner signal to be analyzed and a more accurate frequency calculation. Some of the unfiltered noise may be attributed to the LMC6032 Op-Amp, as increasing the input voltage to the op-amp resulted in higher noise levels on the output terminal. The LMC6032 consists of 2 separate op-amps with pins located on either side of the component. For the purpose of this project, only one of the two op-amps were used, and (as brought up in lecture) connecting the remaining pins to ground may have decreased the noise enough to bring our SNR values up to reach the desired value of 20 dB. Another solution to the noise problem found may be reducing the gain to the minimum level to achieve an output voltage when the microphone was stimulated may reduce the clipping found in our analysis. This clipping may be increasing noise by pushing the boundaries of the LMC6032 and other components, making a more distorted and “noisy” output signal.

Another possibly more complex improvement would be further streamlining the Arduino code. This could be done by creating a self adjusting frequency that changes the sampling frequency based on a first rough estimate of the frequency of the tone. Using a sampling frequency as close to twice the expected tone frequency makes the FFT calculation much more accurate. Other information surrounding the FFT could likely be found and implemented into the code as well.

One improvement that could be made to the MATLAB portion of the project would be to automate the process of taking in data from the CSV file, or even directly from the Arduino. As it is now, the Arduino outputs voltage data to serial, and then that data is copy and pasted into MATLAB along with the sampling frequency. If the Arduino could tell MATLAB the sampling frequency as well as loading the data in automatically, the process would be much more streamlined.

Skills Gained

This project had many parallels to professional engineering. As with every large project, the management and process as a whole is an important skill that will be used in nearly any career.

Understanding what needs to get done, making a plan, then executing it in a timely manner is at the base of every project no matter the size.

Along with these soft skills of professional work and communication, many technical skills were gained that will be useful in future schooling and eventually careers. Testing designs using available resources and verifying your own designs before moving on is one example.

Some more specific technical skills gained were introductions to signal processing and analog circuits. Some of the team members are interested in focusing on these topics and creating careers around them so this project was a very good first taste in that aspect.

One thing that can not be overlooked throughout this project is independence. Given the circumstances with remote instruction, no available labs or lab equipment, and no ability for in person TA or instructor help to verify our designs, this project was a lesson on using what you have. Nearly all of the problems and concepts were figured out via our own research. Although this is not always possible and it is more efficient to use resources as they are available, knowing that a project like this can still be completed with minimal help boosts confidence.

Final Thoughts

Overall this project was both fun and meaningful. Lots of experience was gained on a technical level as well as on a personal level. It was challenging but paid off in the end to see a finished product.

7 Resources

CMC-5042PF-AC Electret Condenser Microphone Data Sheet. CUI Devices.
https://www.cuidevices.com/product/resource/cmc-5042pf-ac.pdf
Texas Instruments Non-Inverting Microphone Pre-Amplifier Circuit.
https://www.ti.com/lit/an/sboa290/sboa290.pdf?ts=1607140402505
Condes, Enrique. (2020, October 6). Arduino FFT Library.
https://github.com/kosme/arduinoFFT
Lettsome, Clyde. (2019, December 18). Audio Frequency Detector. Retrieved from
https://create.arduino.cc/projecthub/lbf20012001/audio-frequency-detector-617856

8 Appendixes

Appendix 1 - Overall Arduino Sketch for FFT and Tone Detection

```
/* ECE 341      TONE DETECTOR ARDUINO CODE
 *
 * AUTHORS:     Kyle Barton , Samuel Barton, Miles Drake
 *
 * DATE:        November 22, 2020
 *
 * DESCRIPTION: This code takes in raw analog voltage data from the microphone
 *              amplifier circuit via ADC sampling and performs a Fast Fourier Transform calculation
 *              on the data, finding the frequency of the highest energy sound detected. This
 *              information is then used to power a digital output pin with 5V based
 *              on which frequency is detected.This program only provides an approximation of the frequency.
 *
 * REFERENCES:  Code is based off Clyde A. Lettsome's code retrieved from this Arduino project:
 *              https://create.arduino.cc/projecthub/lbf20012001/audio-frequency-detector-617856
 */


/* Required libraries */
#include "arduinoFFT.h"

/* Global Variables */
#define SAMP_FREQ 1050
#define NUM_SAMP 64

unsigned int samp_period;
unsigned long timing;

/* Vectors to hold samples */
double real[NUM_SAMP];
double imag[NUM_SAMP];
double freq;

/* Create object named FFT from library */
arduinoFFT FFT = arduinoFFT();



void setup() {

  /* Begin serial comm */
  Serial.begin(115200);

  /* Set sampling period */
  samp_period = round(1000000 * (1.0 / SAMP_FREQ));

  /* Set digital output pins to power LEDs */
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);

}



void loop() {

  /* Sampling */
  for (int i=0; i<NUM_SAMP; i++) {

    /* Time since Arduino began script */
    timing = micros();

    /* Take in ADC measurement from pin A0 */
    real[i] = analogRead(0);

    /* Set imaginary term of sample to 0 */
    imag[i] = 0;
    /* Pause to set frequency */
    while((samp_period + timing) > micros()){
    }

  }


  /* FFT */
  FFT.Windowing(real, NUM_SAMP, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
  FFT.Compute(real, imag, NUM_SAMP, FFT_FORWARD);
  FFT.ComplexToMagnitude(real, imag, NUM_SAMP);

  /* Calculate most prominant frequency */
  freq = FFT.MajorPeak(real, NUM_SAMP, SAMP_FREQ);
  Serial.println(freq);


  /* Calculate note and power corresponding output pin */
  if ((freq > (261.63 * .95)) && (freq < (261.63 * 1.05))) {
    /* Print note */
    Serial.println("Note: c4");
    /* Write pin high for 2 seconds */
    digitalWrite(3, HIGH);
    delay(2000);
    digitalWrite(3, LOW);
    delay(2000);

  }
  if ((freq > (293.66 * .95)) && (freq < (293.66 * 1.05))) {
    Serial.println("Note: d4");
    digitalWrite(4, HIGH);
    delay(3000);
    digitalWrite(4, LOW);
    delay(3000);
  }
  if ((freq > (329.63 * .95)) && (freq < (329.63 * 1.05))) {
    Serial.println("Note: e4");
    digitalWrite(5, HIGH);
    delay(3000);
    digitalWrite(5, LOW);
    delay(3000);
  }
  if ((freq > (345.45)) && (freq < (349.23 * 1.05))) {
    Serial.println("Note: f4");
    digitalWrite(6, HIGH);
    delay(3000);
    digitalWrite(6, LOW);
    delay(3000);
  }
  if ((freq > (392 * 0.95)) && (freq < (392 * 1.05))) {
    Serial.println("Note: g4");
    digitalWrite(7, HIGH);
    delay(3000);
    digitalWrite(7, LOW);
    delay(3000);
  }
  if ((freq > (440 * .95)) && (freq < (440 * 1.05))) {
    Serial.println("Note: a4");
    digitalWrite(8, HIGH);
    delay(3000);
    digitalWrite(8, LOW);
    delay(3000);
  }
  if ((freq > (493.88 * .95)) && (freq < (493.88 * 1.05))) {
    Serial.println("Note: b4");
    digitalWrite(9, HIGH);
    delay(3000);
    digitalWrite(9, LOW);
    delay(3000);
  }
  if ((freq > (518.574)) && (freq < (523.25 * 1.05))) {
    Serial.println("Note: c5");
    digitalWrite(10, HIGH);
    delay(3000);
    digitalWrite(10, LOW);
    delay(3000);
  }


  /* Only run code once */
  while(1);
```

```
}
```

Appendix 2 - Arduino script for .CSV file of voltage over time plot to be analyzed in MATLAB.

```
#include <avr/io.h>


#define SAMP_FREQ 8800 //set to 20x expected frequency of signal
#define NUM_SAMP 512

unsigned int samp_period;
unsigned long timing;
unsigned int adcin[NUM_SAMP];

/* Vectors to hold samples */
double real[NUM_SAMP];


void setup() {

  /* Begin serial comm */
  UCSR0A = 0b00100010;
  UCSR0B = 0b00001000; // bit 2 in this one and
  UCSR0C = 0b00000110; // bits 2 and 1 in this one control the # bits of the serial out
  UBRR0H = 0b00000000;
  UBRR0L = 0b00000000;

  ADCSRA = 0b11110110;//Bit 7 (ADEN) is set to 1 to enable the ADC
                      //Bit 6 (ADSC) is set to 1 to start conversion
                      //Bit 5 is set to 1 to enable auto trigger
                      //Bit 4 is set to 1 to clear the interrupt flag
                      //Bit 3 is set to 0 to deactivate complete interrupt
                      //Bits 2 through 0 are set to 0 to select a 1/2 clock division for the ADC

  ADMUX = 0b01100000; //Bit 7 and 6 are set to 0 and 1 respectively to select AVcc as the reference voltage
                      //Bit 5 is a 1 to left adjust ADC output
                      //Bit 4 is unused
                      //Bits 3 through 0 are set to 0 to select Analog Input A0 as the input to the ADC using the
input mux

  /* Set sampling period */
  samp_period = round(1000000 * (1.0 / SAMP_FREQ));
}


void loop() {
  unsigned long begintime = micros();
  /* Sampling */
  for (int i=0; i<NUM_SAMP; i++) {

    /* Time since Arduino began script */
    timing = micros();

    adcin[i] = (ADCH<<2); //take in the 8 MSBs from the ADC and shift to the left.

    /* Pause to set frequency */
    while((samp_period + timing) > micros()){
    }

  }

  //Serial.println(micros()-begintime);
  /* Only run code once */
  for (int n=0; n<NUM_SAMP; n++) {
    Serial.println(adcin[n]);
  }
  while(1);
}
```

Appendix 2 - MATLAB code for taking in data from CSV file and plotting voltage, FFT, SNR

```
%--------------- Sampling Info -------------------%
Fs = 9860; %Sampling Freq
Ts = 1/Fs; %Minimum timestep

%--------------- Data Input ---------------------%
data = b49860512;                 %Take in data as a table
adcinput = table2array(data);     %Change data into an array
```

```
voltage = adcinput/1024*5;        %Turn ADC values into voltage values

time = 0:Ts:(length(voltage)*Ts-Ts); %Create a time array for the given sampling freq

%--------------- FFT Calc ------------------------%
nfft = length(voltage);           %Length of fft
nfft2 = 2^nextpow2(nfft);         %Find nearest power of 2 for new length
ff = fft(voltage,nfft2);          %Take the fft
fff = ff(1:nfft2/2);              %Just the first half because it's mirrored
xfft = Fs*(0:nfft2/2-1)/nfft2;    %Set the frequency scale using Fs

%--------------- Plotting ------------------------%
%Plot 1, Time Domain
subplot(3,1,1);
plot(time, voltage,'r');
title('Voltage Over Time');
xlabel('Time (s)');
ylabel('Voltage (V)');

%Plot 2, Frequency Domain
subplot(3,1,2);
plot(xfft,abs(fff),'r');
title('Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
ylim([0 300]);
xlim([0 2500]);

%Plot 3, SNR
subplot(3,1,3);
plot(time,voltage);
snr(voltage,Fs,2);
```