ECE 341 Group 16

Accelerated Project

Written By: Francisco Velasco, Maxim Feoktistov and Kris Haro

Date Performed: December 4, 2020

Instructor: Matt Shuman

Grading TA: Shane Witsell

Contents

1 Introduction

This is a project that will encapsulate the skills of Arduino programming, MATLAB programming, LTSpice simulation and circuit construction. The goal of this project is to be able to create an audio amplifier with the supplied components and have many different variables tested. The circuit is to turn on one out of eight LED's based on the musical note being played. This document will outline various different activities beyond turning on the LED's, including: the frequencies that turn on an LED, the circuit working from a 10 foot distance relative to the audio source, the varying levels of voltage amplitude (when the microphone is stimulated and when it is idle), a minimum of 20 samples per period, a minimum three periods of audio graphed in MATLAB, computing the SNR (Signal to Noise Ratio), and plotting the Power vs Frequency graph with at least three harmonics.

2 Background

The amplifier circuit consists of many components including: an op amp, resistors, capacitors, LED's, and a microphone. The full schematic of the design is shown in Figure 1. The block diagram shown in Figure 2 specifies each filter, the inputs, and the outputs of the design. The Bill of Materials used in the project is shown in Figure 3.

This project will require the use of an FFT (Fast Fourier Transform) in either Arduino IDE or MATLAB that will correctly display on an LED when the audio source is stimulated. Though, only one of the LEDs must turn on when a specific frequency is played. The required range of frequencies is 261.626-523.521 Hz ignoring any flat notes within that range. It is recommended to use a phone or the supplied audio samples to test if the amplifier can light the correct LED.

It is important to have a good understanding on how to send data from an Arduino into MATLAB in order to graph the data that is read from the Arduino (that is connected the circuit shown in the schematic). Another factor to consider is how to use a protoboard, as it is a possible tool not included with the bill of materials that will be able to be used to make stronger connections with certain components (smaller through-hole components like the microphone have shorter legs). Having a good understanding of the circuit and its properties is critical to avoid damaging to the circuitry and Arduino. For example, it is important to ensure that the voltage going into the Arduino is less than 5V due to the threshold voltage being harmful to the microcontroller itself, if it is overloaded. The simulation schematic done on LTSpice is shown in Figure 4 with the results shown in Figure 5.

2.1 Schematic

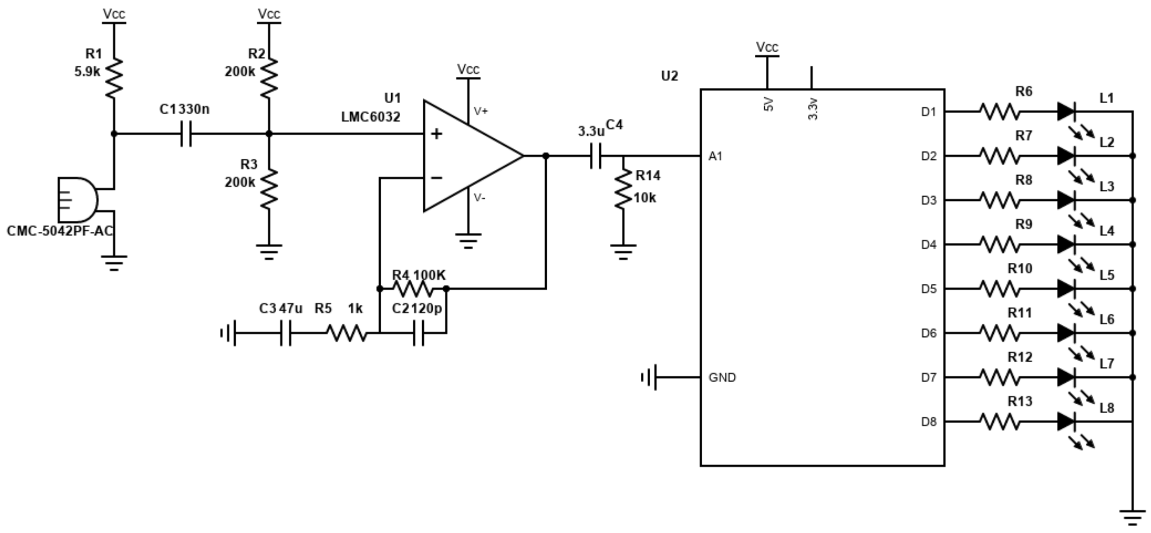


Figure 1: Schematic for the non-inverting pre-amplifier circuit.

2.2 Block Diagram

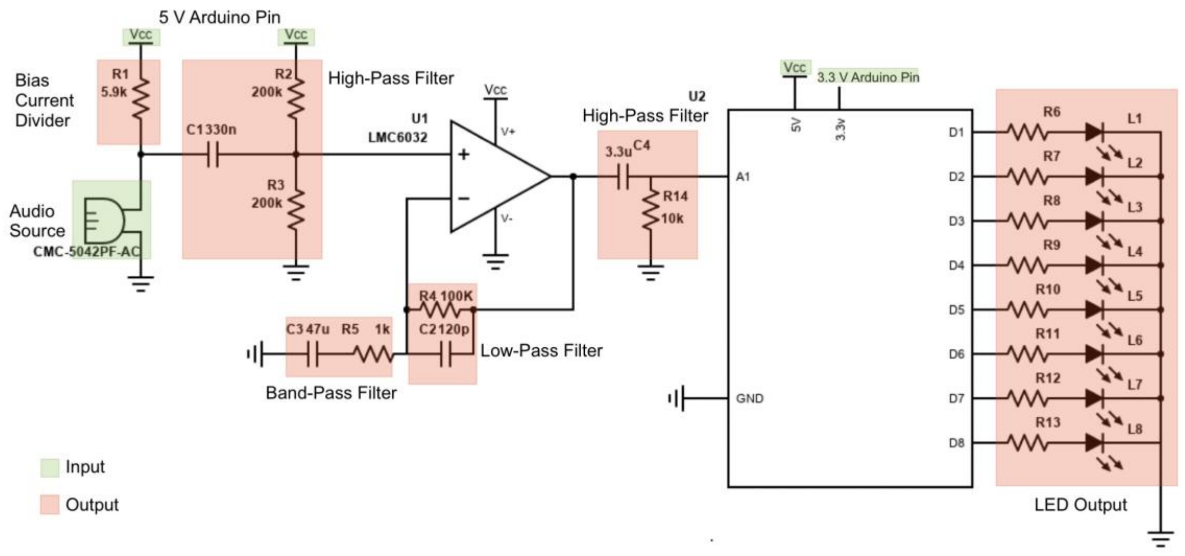
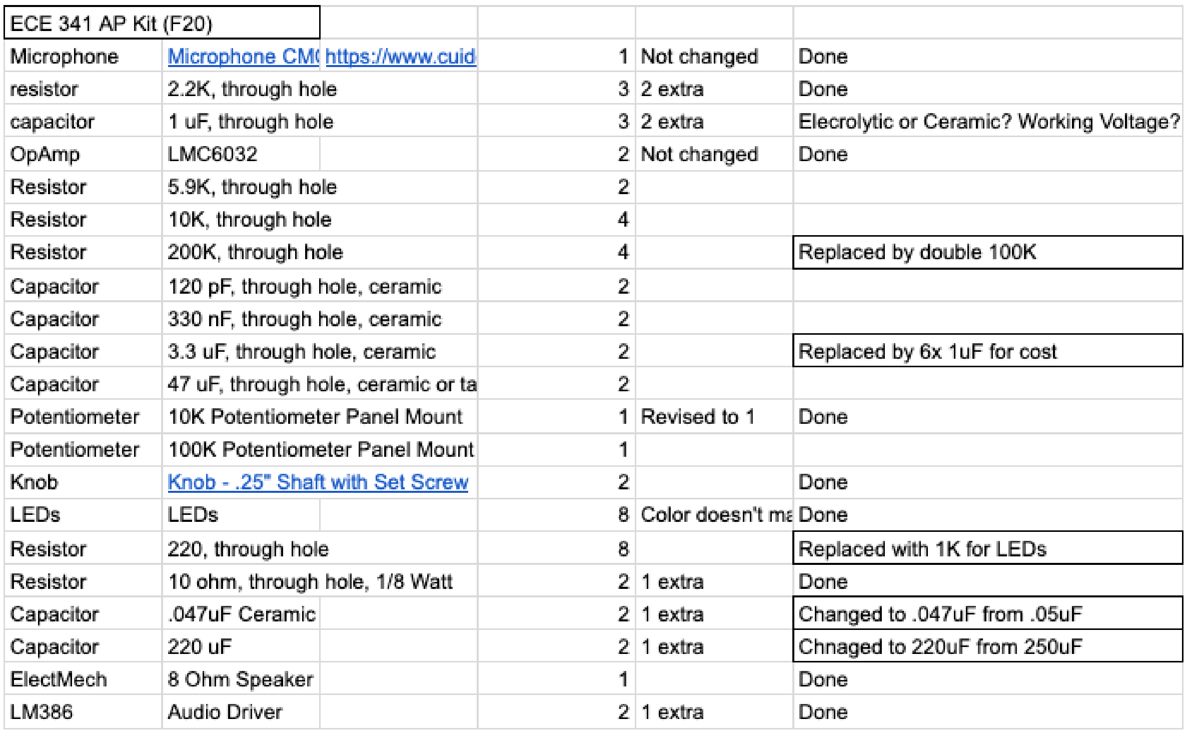


Figure 2: Block diagram for the circuit where specific blocks are labeled and their contributions are labeled.

2.3 BOM



ECE 341 AP Kit (F20)					
Microphone	Microphone CM(	https://www.cuid	1	Not changed	Done
resistor	2.2K, through hole		3	2 extra	Done
capacitor	1 uF, through hole		3	2 extra	Elecrolytic or Ceramic? Working Voltage?
OpAmp	LMC6032		2	Not changed	Done
Resistor	5.9K, through hole		2		
Resistor	10K, through hole		4		
Resistor	200K, through hole		4		Replaced by double 100K
Capacitor	120 pF, through hole, ceramic		2		
Capacitor	330 nF, through hole, ceramic		2		
Capacitor	3.3 uF, through hole, ceramic		2		Replaced by 6x 1uF for cost
Capacitor	47 uF, through hole, ceramic or ta		2		
Potentiometer	10K Potentiometer Panel Mount		1	Revised to 1	Done
Potentiometer	100K Potentiometer Panel Mount		1		
Knob	Knob - .25" Shaft with Set Screw		2		Done
LEDs	LEDs		8	Color doesn't ma	Done
Resistor	220, through hole		8		Replaced with 1K for LEDs
Resistor	10 ohm, through hole, 1/8 Watt		2	1 extra	Done
Capacitor	.047uF Ceramic		2	1 extra	Changed to .047uF from .05uF
Capacitor	220 uF		2	1 extra	Chnaged to 220uF from 250uF
ElectMech	8 Ohm Speaker		1		Done
LM386	Audio Driver		2	1 extra	Done

Figure 3: The Bill of Materials for the Accelerated Project.

2.4 Simulation of Design

The simulation of this design was done in LTSpice. The core elements of the circuit were the same as the reference circuit supplied in the data sheet for this simulation. However, there are some considerable changes that were made from the reference circuit in the data sheet[1]. One major change includes changing the values of R4 and R5, which will also affect future aspects of the project. The impact of this change will be further discussed in greater detail in later sections.

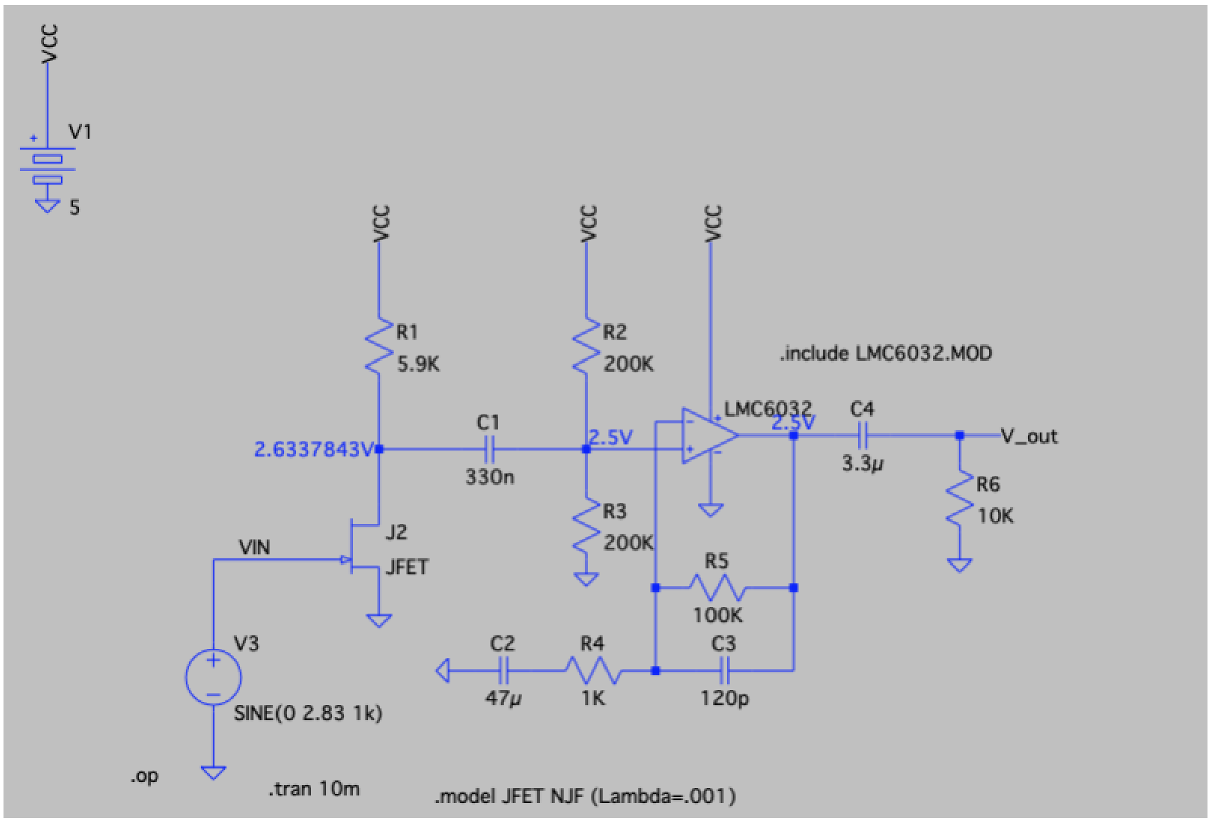


Figure 4: The schematic that was used to simulate the design in LTSPice. As stated previously, the changes are in the values of R4 and R5.

[1]https://www.ti.com/lit/an/sboa290/sboa290.pdf?ts=1607022071762

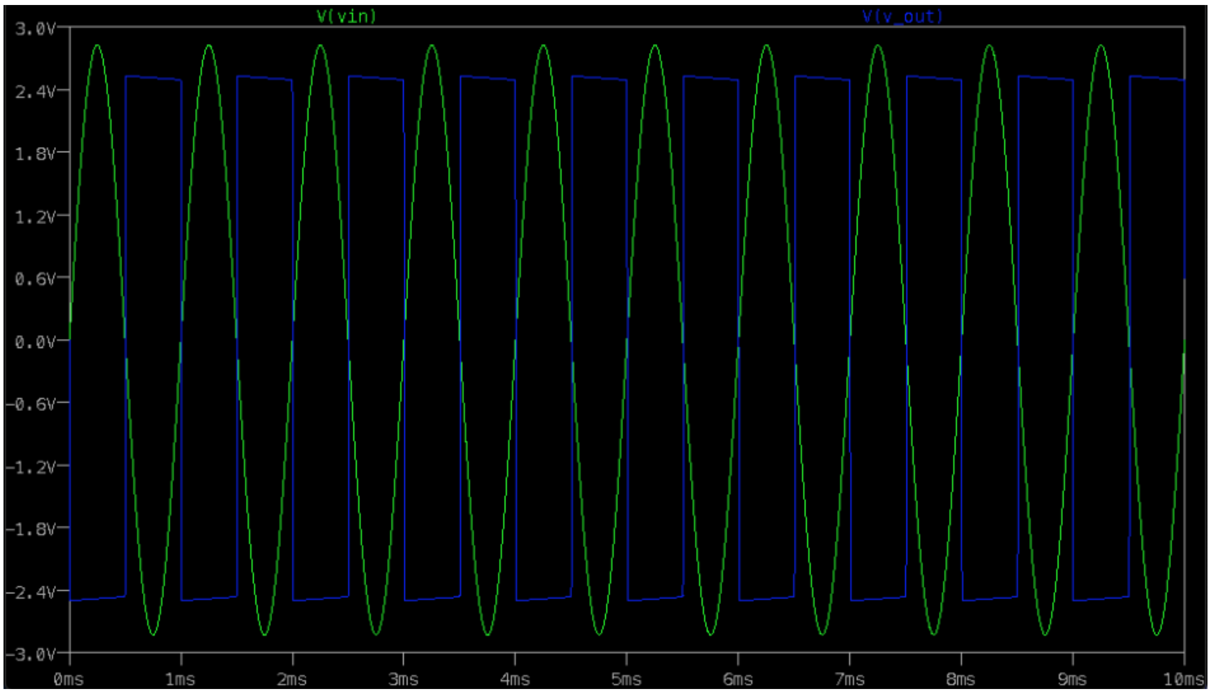


Figure 5: The simulation results from the above schematic in Figure 4.

3 Procedure

The procedure consisted of multiple sections that involved picking the components needed in the circuit, building the circuit, and testing the circuit. Testing the circuit involved testing the audio voltage, signal-to-noise ratio, sampling rate and doing an fft of the audio data to turn on LEDs while playing certain notes.

3.1 Audio Data

During the process of picking the right components for the circuit the following links were referenced:

Non-inverting microphone pre-amp circuit `ti.com/lit/an/sboa290/sboa290.pdf?ts=1605986370018`
Microphone data sheet `https://www.cuidevices.com/product/resource/cmc-5042pf-ac.pdf`

A similar circuit as the circuit in the first link above was used in this project. All components were the same except for the microphone, op amp, and resistors R4 and R5. The microphone and op amp in the BOM (Figure 3) were used. R4 and R5 had to be calculated to meet the wanted gain. The following equation was used to find the R4 and R5 values:

$$\frac{R5}{R4} = Gain - 1$$

To find the highest gain possible without breaking the Arduino the following equations were used:

$$I_m = 2Pa * \frac{10^{(\frac{Sensitivity_m}{20})}}{Z_m}$$

$$R_i = R1||R2||R3$$

$$V_m = I_m * R_i$$

$$Gain_m = \frac{5v}{V_m}$$

Since the design had to be able to read frequencies from at least 10 feet away the gain had to be large enough to read the stimulated sound with a voltage amplitude greater than 1V and when idle must have an amplitude of less than .2V. Once the gain was set and the R4 and R5 values were calculated the circuit in Figure 1 was constructed on a breadboard. A phone app containing all the notes with their frequencies was used to simulate notes. The Arduino IDE code used to collect samples is shown in the Appendix in section 7.1.1. The samples were collected in the serial monitor and then stored in a .csv file for signal processing in MATLAB.

3.2 SNR

To calculate the signal-to-noise ratio, a snr() function was used in MATLAB. MATLAB would read the .csv file and output the SNR of the signal samples collected. The code used to implement the snr() function can be found in the Appendix section 7.2.

3.3 Frequency Domain

To calculate the frequency domain of the signal samples collected by the Arduino, MATLAB was also used. The MATLAB code can be found in the Appendix section 7.2.

3.4 Frequency dependent LEDs

To turn on an LED at a certain frequency, the fft was done in Arudino IDE rather than MATLAB. When the mic was simulated the LED's would change right away based on the frequency being played. The Arduino IDE code used can be found in the Appendix section 7.1.2.

4 Results

4.1 Audio Data

One of the objectives of this project where an induced audio amplifier had a minimum of 1V amplitude and when idle the audio amplifier had a amplitude below 0.2V. After making prior adjustments within the circuit (higher gain), the data sent from the Arduino into MATLAB was able to correctly show the voltage amplitude for the respective requirements. There were some hurdles when it came to this part due to the noise in the background of the work spaces that we had and the microphone was able to pick up the sounds of different sources like a fan from a computer for example. Therefore, in a silent room, the amplitude of the signal from an idle audio amplifier was less than 0.2V as seen in Figure 6. The other requirement did not rely on specific conditions and when the audio source was stimulated, the amplitude of the signal was above the 1V minimum as seen in Figure 7.

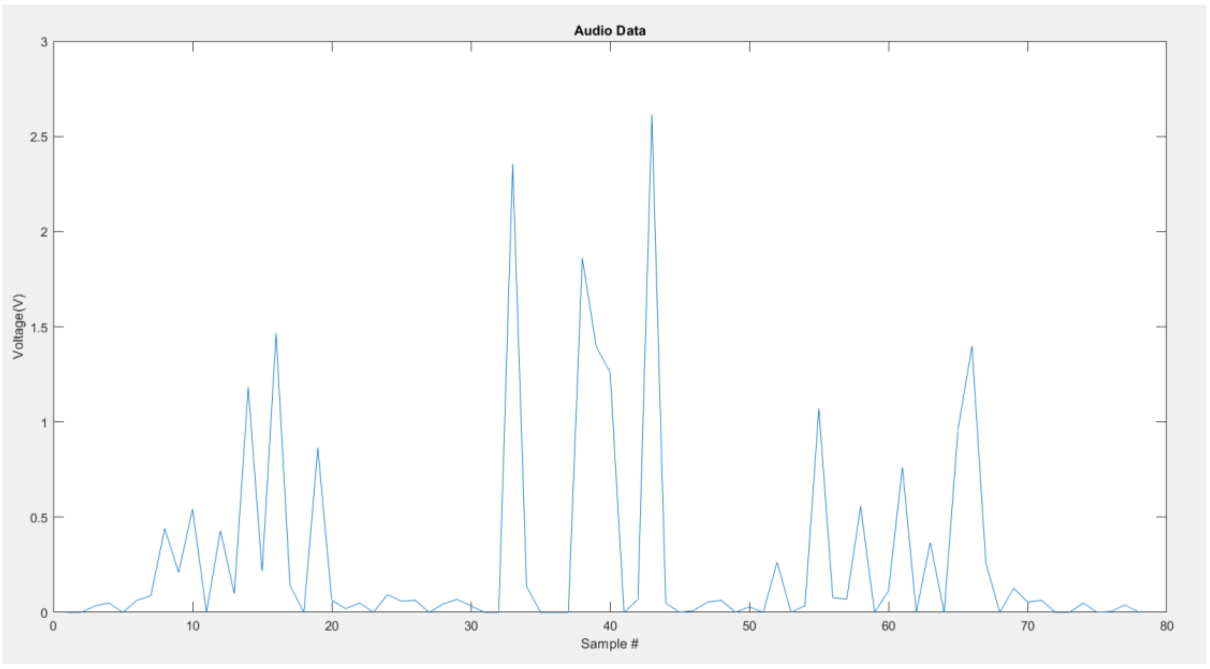


Figure 6: The signal of an idle audio amplifier that shows the amplitude being below 0.2 Volts.

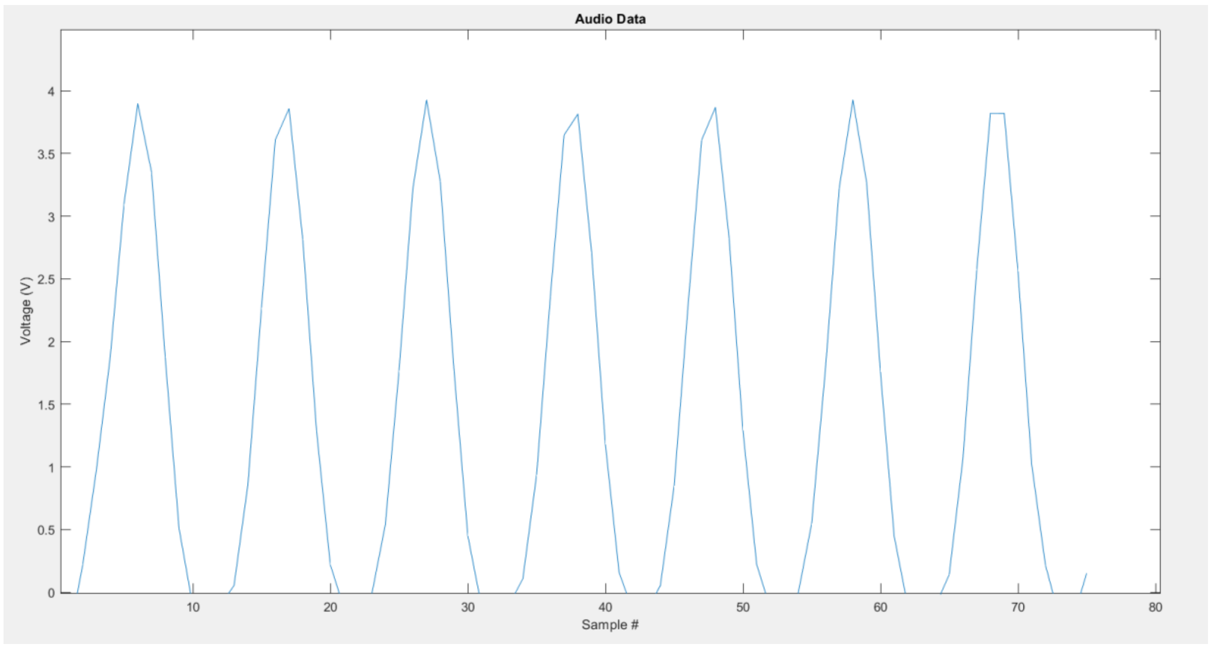


Figure 7: The signal of a stimulated audio amplifier that shows the amplitude being above 1 Volt.

One of the objectives was to ensure that each period had at least 20 samples for every frequency. This design was able to implement only 10 samples per period. This objective was not met partly due to the hardware limitations within the supplied Arduino and the limited hardware that was given to do this project. If cost wasn't an issue in a project like this, we would need to address the need for a higher frequency that the samples are being collected at. One possible solution is to be able to purchase a microcontroller that has more SRAM than of the 1 kB on the Arduino Nano 168p supplied in the kit. If this were a more robust Arduino and cost weren't an issue, the likelihood of being able to meet the requirement would have been more attainable. Due to the fact that the current Arduino being limited to approximately 1000 samples per period, the samples were needed to be post processed and required the need to calculate the samples in MATLAB. If it were an ideal scenario, it likely would have been more attainable if the microcontroller could both collect the samples needed and also calculate the FFT (not limited by the memory). This would have lead to the ability for more control of the number of samples being collected per period.

While collecting the data from the audio amplifier, there are to be at least three periods of data that will be graphed in MATLAB. This part of the project required some thought due to the fact that it was unlikely for the data to be processed live, which was the approach that was initially used in the attempt to accomplish the minimum number of periods within a trial run. As mentioned above, there were some hardware limitations that did not allow for full sampling and FFT calculations due to the limited memory in the supplied Arduino Nano with the ATMega 168p. However, this was something that was able to transfer well to MATLAB because the calculations within MATLAB was not limited in memory. The FFT calculations and the data transferred post-process were able to be read correctly and the data was able to be displayed correctly. In the figure above in the Figure 7, there are clear periods shown within the data that is collected.

4.2 SNR

The signal to noise ratio was something that needed to be at a minimum of 20bB's. The sensitivity of the mic was an issue because it was constantly picking up other audio that was not part of the circuit, like the fans of a computer, the typing of keys on a keyboard, and general background noise in a home. This required an increase of the amplitude of the audio that was stimulating the audio amplifier and that was something that was able to raise the SNR. The increase in amplitude for the stimulated audio source was able to limit the background noise within the testing area and it was able to lower the noise of the SNR. There was a clear increase in signal amplitude with the increase in volume. In the graph shown in Figure 8, the signal-to-noise ratio and is larger than the required SNR of 20dB's.

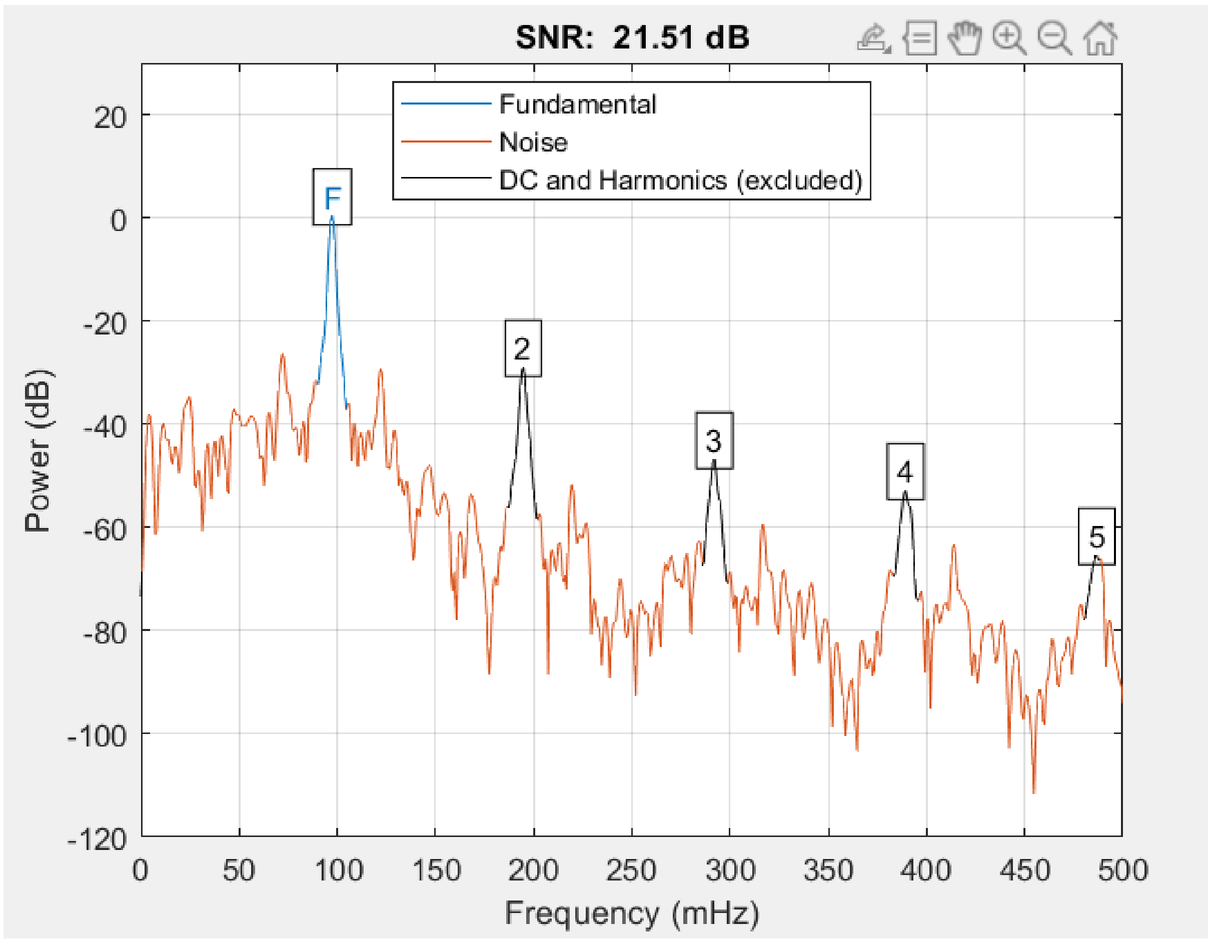


Figure 8: The graph that shows the SNR at the top of the figure, which satisfies that minimum SNR of 20.

4.3 Frequency Domain

The final part of this project was to plot the Power vs Frequency spectrum that shows a minimum of three harmonics. Figure 9 (attached below) shows the final output that was produced with the FFT used in MATLAB that actually shows five harmonics. However, upon first glance, it may appear that there are only two clear harmonics, but if looked at closely, there are actually five harmonics. The peaks of the harmonics have been pointed out to show that there is indeed five harmonics instead of two. A notable takeaway from this graph is that it matches with the harmonics seen in Figure 8 above. Also, it is apparent that the amplitude of the harmonic decreases significantly as the number of harmonics increase. The other validation in the graph is the fact that the note is clearly being picked up as expected since the frequency sampled was 523Hz, which validates our hardware.

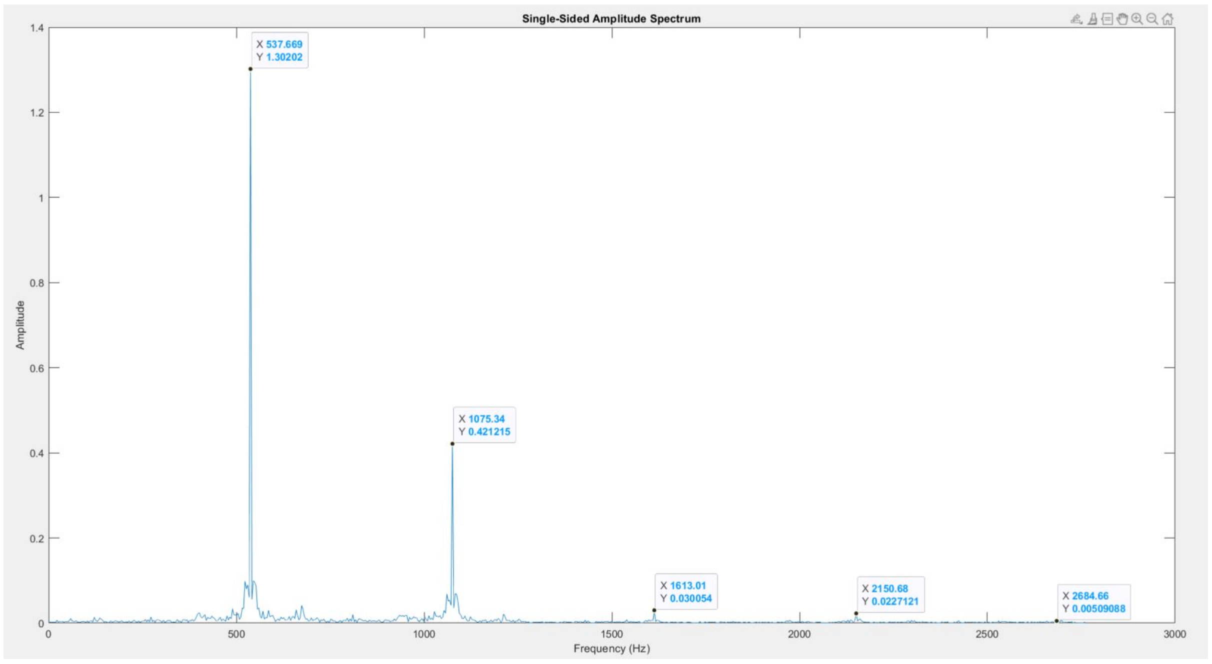


Figure 9: Frequency domain with 5 harmonics

4.4 Frequency Dependent LEDs

This last activity was the main focus of the project and it will require the use of an FFT that will be validating the hardware the proposed circuit. The Arduino code for this step did not require any data transfer to MATLAB, but instead, it focuses on testing the hardware and setting up the Arduino to read data from the circuit and the output on the LEDs would vary depending on the frequencies that are played. The simulated frequencies were used from example videos that played various different frequencies, or an app on a cell phone that can play any given frequency. This task required to use the FFT on the samples that were taken of the audio and return the highest frequency of a sample. With the highest frequency, we were able to narrow down the ranges of frequencies that would only light certain LEDs with a test audio (frequency). Due to the requirements of ignoring sharps and flats, it again narrowed down what frequency range each LED was going to be in. Though, there is another requirement that must light the LED with the eight notes being within +/- 0.5% and that requires some fairly quick calculations that will be shown in the appendix section where all of the code that was used during the project will be presented with comments. As an example from a previously recorded demo, Figure 10 showcases a snapshot of a trial of the circuit in action.

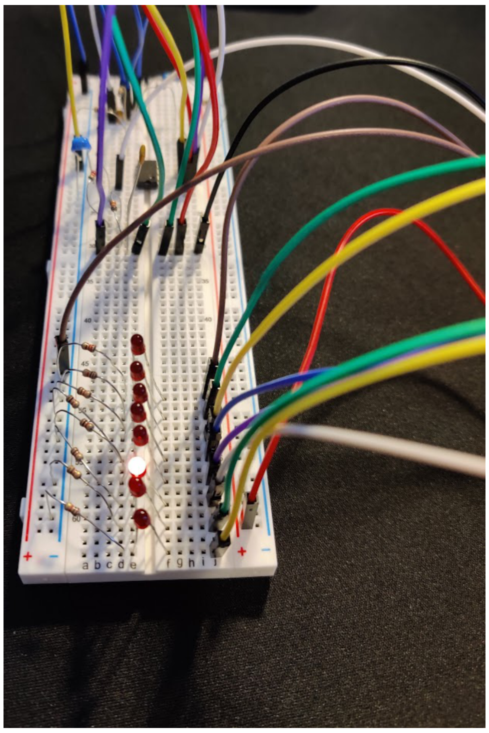


Figure 10: An example of an LED being turned on from a past demo. Here, LED 6 was lit when a frequency of (approximately) 440 Hz was played (refer to code in Appendix).

The next activity of the project was something that was building on the prior requirement above. In the example from Figure 6, the audio source was very close to the microphone. However, our initial design was not adequate enough to pick up any frequencies (with the accuracy required to satisfy the requirement). Using the initial design that was supplied for the project (mentioned previously in the "Simulation of Design" section) resulted in a very low gain of approximately 13. With the gain in the original design, the microphone could not clearly sense any audio coming from the source if it was past three feet from the sound. This required some adjusting of some different components within the circuit. The main issue here was the gain, therefore, in order to increase the gain to a much higher level, the components of R4 and R5 (seen in Figure 1) were adjusted accordingly. With the adjustment of the resistors, the resulting gain was about 100 and that was sufficient enough to have an LED to light from picking a frequency from an audio source that was at least 10 feet away. This process isn't something that is simple to show in a photo as above, but a demonstration will be part of a video that will showcase an audio source being 10 feet away from the mic and will show the required result being fulfilled.

5 Conclusion

Many of the simulated tasks coincided with the our hardware which verifies the legitimacy of the proposed design that was during this project. Some aspects of the project that we could've improved are the frequency domain graphs and the number of samples per period. To increase the amplitude of the power vs frequency graph, a louder signal can be played or increasing the gain of the circuit. To improve the sampling rate, a better microcontroller can be used like an Arduino with more SRAM. With the given Arduino, the sampling rate was improved by directly using ADC registers of the microcontroller but was not enough to get the required 20 samples a period. USART registers could also be directly controlled to possibly increase the sampling rate but was not implemented in the design. The project clearly shows the implementation of a amplifier circuit and how the op amp amplifies the signal sent by the mic. This allowed proper signal analysis meeting most of the requirements of the project.

6 Executive Summary

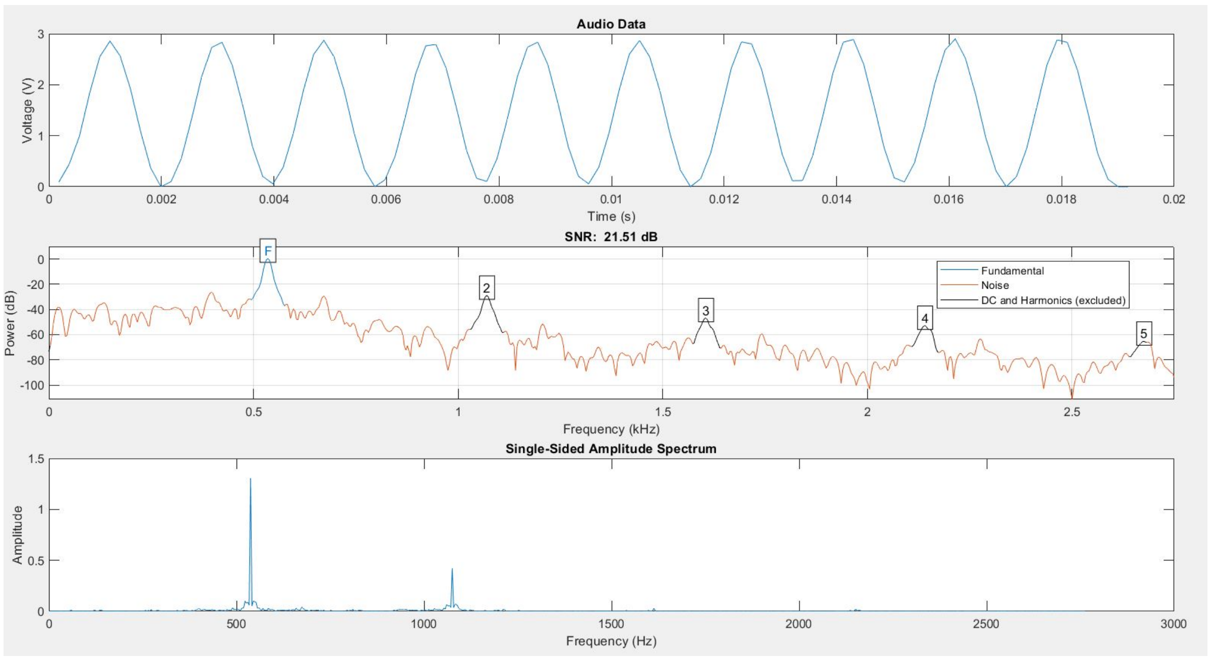


Figure 11: The final results of the amplifier circuit

This is a project that will encapsulate the skills of Arduino programming, MATLAB programming, LTSpice simulation and circuit construction. The goal is to create a frequency detector that will have hardware elements and also elements that need to be simulated and analyzed. The project contains different activities beyond turning on the LED's, including: the frequencies that turn on an LED, the LED's working from a 10 foot distance (from source), the varying levels of amplitude (in Volts) when the microphone is stimulated and when it is idle, a minimum of 20 samples per period, a minimum three periods of audio graphed in MATLAB, computing the SNR (Signal to Noise Ratio), and plotting the Power vs Frequency graph with at least three harmonics.

After the construction and testing of the circuit, nearly all of the simulated tasks coincided with the designed hardware. Some aspects of the project that we could have improved are the frequency domain graphs and the number of samples per period. These were two parts that were not fully complete or they could have been more improved to ensure that the outcome was as expected. Referring to Figure 11, the graphs shows many of the previously stated requirements being met. The first row in the figure is proving that a stimulated audio amplifier has more than 1V of amplitude and has more than three periods in one collection of data. Following that, the second row in the figure shows the signal-to-noise ratio being above the minimum of 20 and also shows a preview of the harmonics. The final row of the figure is the Power vs. Frequency graph that shows the minimum of three harmonics in the graph. For more details as to why the harmonics aren't inherently clear upon first glance, refer to section 4.3. As for the lack of a visual for the number of samples per period, refer to section 4.1. Finally, for further information and a visual for a non-stimulated audio amplifier refer to Figure 7 in section 4.1.

7 Appendix

7.1 Code used for firmware programming

7.1.1 Sketch used to collect data

```
#include <avr/io.h>
/** Code used to collect samples as quickly as possible.
     It is currently able to collect 10 samples per period at the
     highest frequency.
     For more information on how the registers were programmed,
     visit http://www.optiloading.be/willem/Arduino/speeding.pdf
*/

const int analogInPin = A0;
int sensorValue = 0;


void setup()
{
  Serial.begin(1000000); //Baud rate for the Serial Monitor

  DDRC= 0b11111110; //sets bit 0 as input

  PORTC=0b00000000; //pullups are disabled

  ADMUX = 0b01100000; //Voltage refrence at AV_cc, connects ADC to A0

  ADCSRA = 0b11000110; //ADC is enabled, bits 3:0 use clock that is divided by 64
}


void loop()
{
  long start, finish ;

  start = micros();

  for(int i=0; i<1500; i++){ //makes 1500 samples

   int val=analogRead(analogInPin);
   Serial.println(val);  //prints sample
}
 finish= micros()-start;

 Serial.print("time per sample(us): ");
 Serial.println((float)finish/1500); //calculates time per sample

 Serial.print("frequency (HZ): ");
 Serial.println((float) 1500*1000000/ finish ); //calculates frequency
 Serial.println();

   while(1);
}
```

7.1.2 Sketch used to program the LEDs[2]

```
/*
 * Orignal Author: Clyde A. Lettsome, PhD, PE, MEM
 * Alterations by Kris Haro, Francisco Velasco, and Maxim Feoktistov
 * Team 16's alterations speed up the ADC clock speed and added
 * a section based on the value of the frequency after the FFT
 * calculation.
 * Lettsome's work is also based on the examples provided by
 * the Arduino FFT library. For Lettsome's orignal code and the
 * Arduino FFT examples, visit
 * the link in the footnote, as well as
 * https://github.com/kosme/arduinoFFT
*/
#include "arduinoFFT.h"

#define SAMPLES 128 //SAMPLES-pt FFT. Max 128 for Arduino Nano 328
#define SAMPLING_FREQUENCY 1046
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

arduinoFFT FFT = arduinoFFT();

unsigned int samplingPeriod;
unsigned long microSeconds;

int led1 = 5; //Corresponding LED Pin
int led2 = 6; //Corresponding LED Pin
int led3 = 7; //Corresponding LED Pin
int led4 = 8; //Corresponding LED Pin
int led5 = 9; //Corresponding LED Pin
int led6 = 10; //Corresponding LED Pin
int led7 = 11; //Corresponding LED Pin
int led8 = 12; //Corresponding LED Pin


double vReal[SAMPLES]; //create vector of size SAMPLES to hold real values
double vImag[SAMPLES]; //create vector of size SAMPLES to hold imaginary values

void setup()
{
    Serial.begin(115200); //Baud rate for the Serial Monitor
    samplingPeriod = round(1000000*(1.0/SAMPLING_FREQUENCY)); //Period in microseconds
    pinMode(led1, OUTPUT); // modify LED1 to be an output
    pinMode(led2, OUTPUT); // modify LED2 to be an output
    pinMode(led3, OUTPUT); // modify LED3 to be an output
    pinMode(led4, OUTPUT); // modify LED4 to be an output
    pinMode(led5, OUTPUT); // modify LED5 to be an output
    pinMode(led6, OUTPUT); // modify LED6 to be an output
    pinMode(led7, OUTPUT); // modify LED7 to be an output
    pinMode(led8, OUTPUT); // modify LED8 to be an output
    sbi(ADCSRA,ADPS2) ; //64 prescaler for speeding up analogRead
    sbi(ADCSRA,ADPS1) ;
    cbi(ADCSRA,ADPS0) ;
}

void loop()
{
```

[2]https://clydelettsome.com/blog/2019/12/18/my-weekend-project-audio-frequency-detector-using-an-arduino/

```
    /*Sample SAMPLES times*/
    for(int i=0; i<SAMPLES; i++)
    {
        microSeconds = micros(); //Finds elapsed time

        vReal[i] = analogRead(0);
          //Reads the value from analog pin 0 (A0),
          //quantize it and save it as a real term.
        vImag[i] = 0; //Makes imaginary term 0 always

        /*remaining wait time between samples if necessary*/
        while(micros() < (microSeconds + samplingPeriod))
        {
          //do nothing
        }
    }

    /*Perform FFT on samples*/
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD); //Weigh data
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD); //Compute FFT, this would be the
     //equivalent of MATLAB's fft function
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES); //Computes the magnitudes
     //in the frequency domain

    /*Find peak frequency and print peak*/
    double peak = FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);
    Serial.println(peak);     //Print out the most dominant frequency.

    //Turn on LED based on the value of peak
       if(peak >= 251 && peak < 271){
      digitalWrite(led1, HIGH); // this will turn on LED1
    }

    else if(peak >= 283 && peak < 303 ){
      digitalWrite(led2, HIGH); // this will turn on LED2
    }

    else if(peak >= 319 && peak < 339){
      digitalWrite(led3, HIGH); // this will turn on LED3
    }

    else if(peak >= 339 && peak < 359){
      digitalWrite(led4, HIGH); // this will turn on LED4
    }

    else if(peak >= 382 && peak < 402){
      digitalWrite(led5, HIGH); // this will turn on LED5
    }

    else if(peak >= 430 && peak < 450){
      digitalWrite(led6, HIGH); // this will turn on LED6
    }

    else if(peak >= 483 && peak < 503){
      digitalWrite(led7, HIGH); // this will turn on LED7
    }

    else if(peak >= 513 && peak <= 533){
      digitalWrite(led8, HIGH); // this will turn on LED8
    }

    else{
       digitalWrite(led1, LOW); // this will turn off the LED
```

```
        digitalWrite(led2, LOW); // this will turn off the LED
        digitalWrite(led3, LOW); // this will turn off the LED
        digitalWrite(led4, LOW); // this will turn off the LED
        digitalWrite(led5, LOW); // this will turn off the LED
        digitalWrite(led6, LOW); // this will turn off the LED
        digitalWrite(led7, LOW); // this will turn off the LED
        digitalWrite(led8, LOW); // this will turn off the LED

    }
    delay(1000);
    //Turn off all LEDs before sampling again
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
    digitalWrite(led5, LOW);
    digitalWrite(led6, LOW);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);


    delay(200);
}
```

7.2 MATLAB code used for FFT calculations

```
%%Code based on the "Noisy Signal" example from Mathworks
%for more information, visit
%https://www.mathworks.com/help/matlab/ref/fft.html
%% Read ADC Values from files
clear,clc,clear all

file = dlmread('voltagesamples.csv') * 5 / 1023;
%This file contains 1500 samples and will be used for the FFT
file2 = dlmread('shortsample.csv') * 5 / 1023;
%This file contains about 80 samples, and is used to plot the voltages

for i = 1:numel(file2)

    time(i) = 181*i/1000000;
     %181 is the calculated time per sample
     %in microseconds
end

%% Plot Voltage over sample count
subplot(3,1,1)
plot(time, smoothdata(file2));
title('Audio Data')
xlabel('Time (s)')
ylabel('Voltage (V)');

%% SNR calculation
subplot(3,1,2)
snr(smooth(file,0.01,'rloess'), 5500)

%% FFT
%The next few lines normalize the signal data
%If these were omitted, there would be an offsetting
%spike at 0Hz
file = double(file);
mean = mean(file, 'all');
file = file - mean;
file = normalize(file);
Fs = 5524; %Value based on measurement perfomed in Arduino IDE
T = 1/Fs;
L =1500;
Y = fft(file);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
subplot(3,1,3)
plot(f,P1)
title('Single-Sided Amplitude Spectrum')
xlabel('Frequency (Hz)')
ylabel('Amplitude');
```