IDP Final Report

Report: Home Audio System Project

Alex Stoffelmayr and Mahdi Mountassir

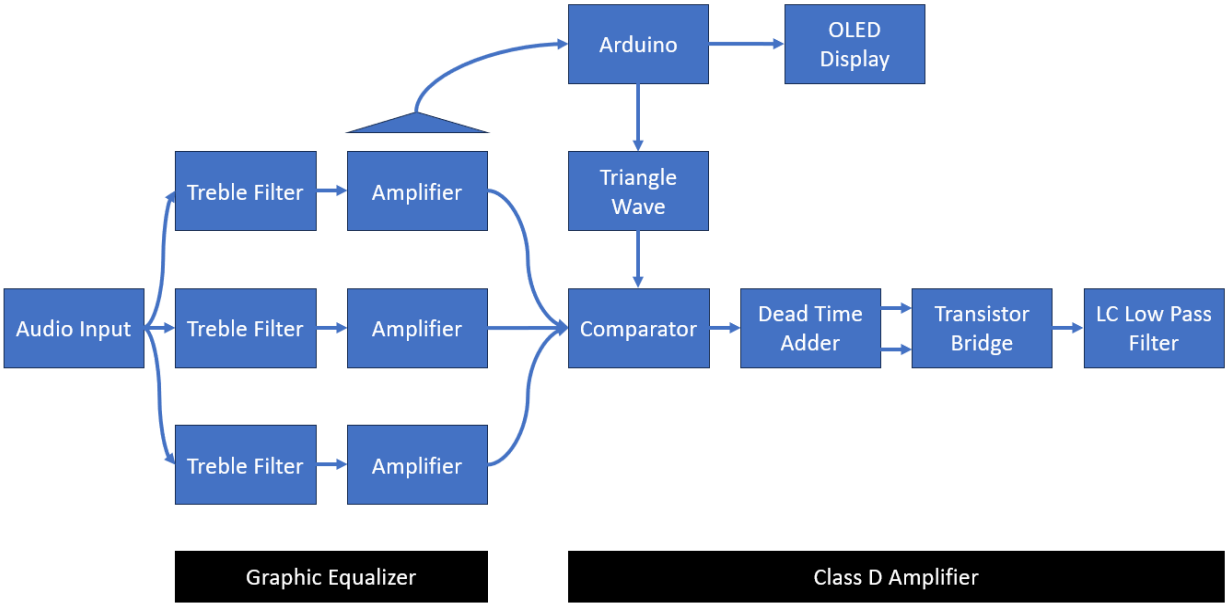
Contents

Introduction

The goal of this project was to create a home audio system. The system consists of three major parts: the graphic equalizer, the class D amplifier, and the OLED display. The graphic equalizer has three dials that control the relative volume of the bass, mid, and treble frequencies of a given audio input. The gain of each of these frequencies can be altered to zero, positive, or negative. To aid in the use of the equalizer, an Arduino generates a display showing the three frequency bands and their relative gains. After the equalizer stage, the signal is sent to a class D amplifier used for its high-efficiency switching stage. The transistor bridge is driven using an original dead time-adding circuit, which is described in the amplifier section.

High-Level Design

The overall design of the system focuses on three major components, each is shown in the block diagram of the entire system provided below. First is the graphic equalizer. This is a series of three bandpass filters: one that captures the bass range from 60Hz to 250Hz, one that captures the mid-range from 250 Hz to 6kHz, and a third filter that captures the treble frequencies from 6kHz to 20kHz. Each band is then separately amplified or attenuated using a simple op-amp gain circuit, which is controlled using a potentiometer. The frequencies are then recombined and sent to the Class D amplifier. The class D amplifier functions by comparing a 35kHz triangle wave to the audio signal from the equalizer, creating a PWM signal. The triangle wave is generated by a 35kHz square wave from the Arduino. The square wave is then sent through an integrator circuit to create a triangle wave. This PWM signal is then altered into two separate waves that each drive a pair of transistors in a half-bridge configuration. The output of this is voltage pulses from zero to five volts. Finally, an LC low pass filter removes the triangle carrier wave and outputs an amplified audio signal. In order to create a video output to monitor the operation of the system, an Arduino measures the voltages of the audio signals being generated by each of the three gain circuits. Then, the Arduino writes to an OLED display showing three bars representing the relative volume of each frequency band.



Graphic Equalizer

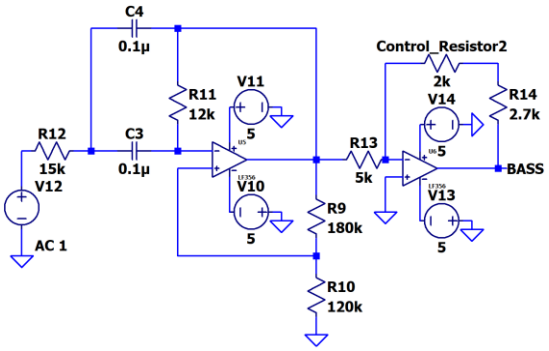
Design and Theory:

Frequency bands for the graphic equalizer were determined according to commonly accepted numbers provided by teachmeadudio.com [4]. The treble and bass filters were created using the filter wizard provided by the design document [2]. It was determined in testing that because the mid-range was much wider than the other two ranges, it would be more effective to use a high pass filter with a cutoff frequency at 250Hz in series with a low pass filter with a cutoff frequency at 6kHz. The feedback of each gain circuit is controlled through a potentiometer as well as an extra 2.7k ohm resistor. The potentiometer is how the user controls the equalizer. The 2.7k ohm resistors were included to boost the maximum possible resistance because it was found that the maximum 10k ohms provided by the potentiometer were not enough to produce a high level of amplification.

Calculations:

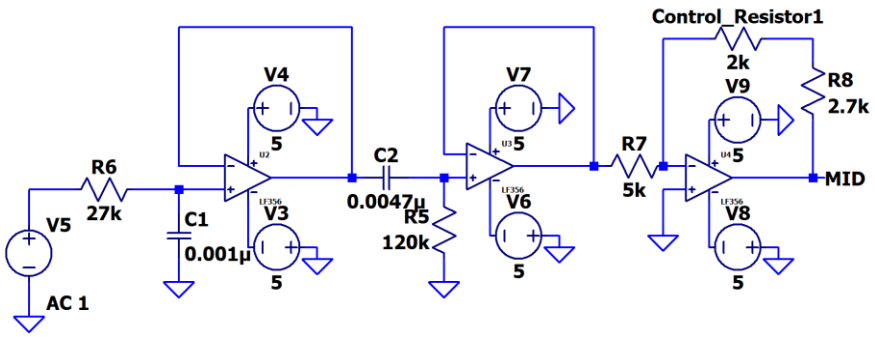
Circuit designs and transfer functions can be seen on the next page. Derivations of the transfer functions can be seen in the pdf linked in the appendix.

BASS AMPLIFIER



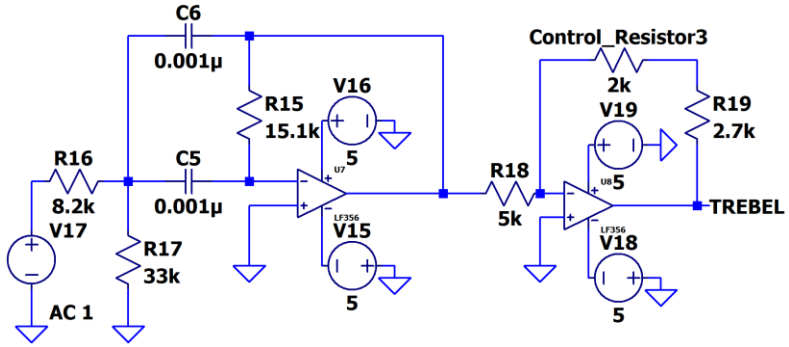
$$14500\left(\frac{-0.034}{j\omega} - j\omega(5.77 * 10^{-8}) + (-6.9 * 10^{-5})\right) * \left(\frac{R_{control} + 1k}{5k}\right) = \frac{V_{in}}{V_{out}}$$

MID AMPLIFIER



$$\left(\frac{1}{\sqrt{1 + \left(\frac{f}{240\ Hz}\right)^2}} * \frac{1}{\sqrt{1 + \left(\frac{f}{6005\ Hz}\right)^2}}\right)\left(\frac{R_{control} + 1k}{5k}\right) = \frac{V_{out}}{V_{in}}$$

TREBLE AMPLIFIER



$$\frac{44150.1}{j(2\pi f)} + 1 + 7.56k\left(\frac{1}{j(2\pi f) * 1.5 * 10^{-9} + \frac{1}{33k}} + \frac{1}{15.1k}\right) * \left(\frac{R_{control} + 1k}{5k}\right) = \frac{V_{in}}{V_{out}(x)}$$

Images of the results of the simulation and testing are included in the appendix. However, a table of relevant values is provided here, as well as references to the figures. As can be seen, both the simulated and measured systems worked exactly as described.

Bass (60 Hz - 250 Hz)

Setting	Simulated	Measured
dB < 0	*Figure 1* 59 Hz: -9.2 dB 117 Hz: -6.2 dB 250 Hz: -9.2 dB	*Figure 18* 57.9 Hz: -8.6 dB 119 Hz: -5.6 dB 259 Hz: -8.6 dB
dB = 0	*Figure 2* 60 Hz: -3.0 dB 122 Hz: -0.4 dB 232 Hz: -3.0 dB	*Figure 19* 57.9 Hz: -3.0 dB 119 Hz: 0.095 dB 258 Hz: -3.0 dB
dB > 0	*Figure 3* 56 Hz: 4.3 dB 123 Hz: 7.3 dB 248 Hz: 4.3 dB	*Figure 20* 57.9 Hz: 4.6 dB 119 Hz: 7.6 dB 258 Hz: 4.6 dB

Mid (250 Hz - 6 KHz)

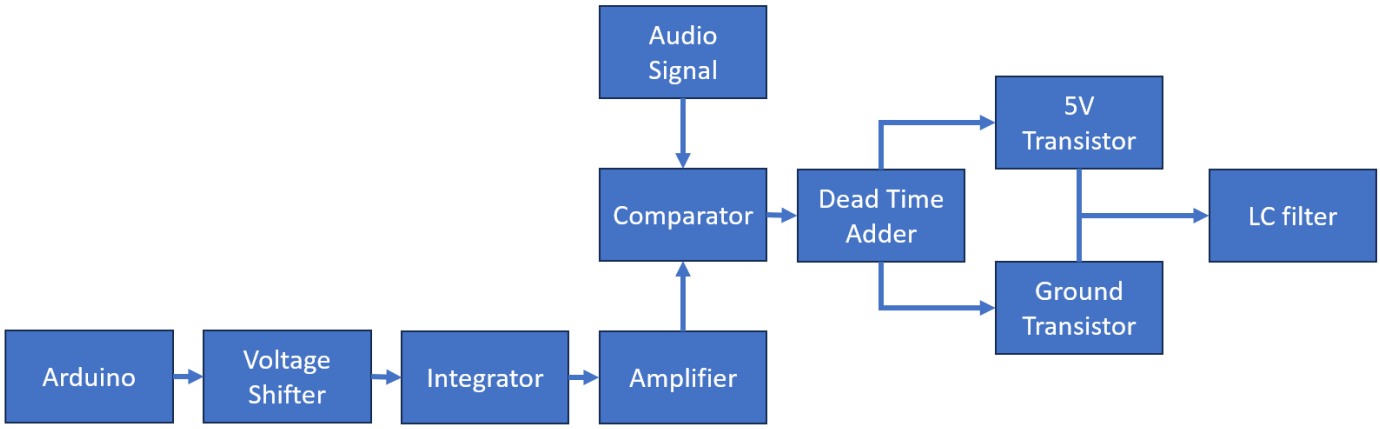
Setting	Simulated	Measured
dB < 0	*Figure 4* 260 Hz: -6.0 dB 1.1 KHz: -3.0 dB 6.4 KHz: -6.0 dB	*Figure 21* 252 Hz: -8.2 dB 1.4 KHz: -5.2 dB 6.6 KHz: -8.2 dB
dB = 0	*Figure 5* 260 Hz: -3.0 dB 1.2 KHz: -0.06 dB 6.4 KHz: -3.0 dB	*Figure 22* 252 Hz: -3 dB 1.4 KHz: 0.02 dB 6.6 KHz: -3 dB
dB > 0	*Figure 6* 258 Hz: 4.7 dB 1.2 KHz: 7.7 dB 6.4 KHz: 4.7 dB	*Figure 23* 252 Hz: 5.4 dB 1.4 KHz: 8.4 dB 6.6 KHz: 5.4 dB

Treble (6 KHz - 20 KHz)

Setting	Simulated	Measured
dB < 0	*Figure 7* 8.5 KHz: -6.3 dB 16 KHz: -3.3 dB 29 KHz: -6.3 dB	*Figure 24* 8.79 KHz: -8.6 dB 16.1 KHz: -5.6 dB 30.7 KHz: -8.6 dB
dB = 0	*Figure 8* 8.5 KHz: -3.3 dB 16 KHz: -0.3 dB 29 KHz: -3.2 dB	*Figure 25* 8.75 KHz: -3.0 dB 16.67 KHz: -0.0 dB 31.42 KHz: -3.0 dB
dB > 0	*Figure 9* 8.5 KHz: 4.3 dB 16 KHz: 7.3 dB 29 KHz: 4.3 dB	*Figure 26* 8.84 KHz: 5.0 dB 16.49 KHz: 8.0 dB 31.1 KHz: 5.0 dB

Design and Theory:

The Class D Amplifier has seven stages, all shown in the below block diagram. The input signal to the system is a 5v 35kHz square wave generated by the Arduino. The first stage is the voltage shifter, which alters the wave from being centered at 2.5v to being centered at zero volts. The integrator takes the integral of this wave. Because the wave is a square signal from -2.5v to 2.5v the integral is a triangular wave centered at zero. This triangle wave is then amplified to about 0.6v to better match the voltage of an audio wave. This method for creating a triangle wave is detailed by eeeguide.com [6]. Then, the triangle wave is passed through a comparator along with the audio input. This operation results in a PWM signal representing the sinusoidal audio wave. This method was adapted from the one detailed in the Texas Instruments document “Analog Pulse Width Modulation” [14]. The PWM signal is then sent to the dead time adder circuit. The dead time adder circuit is an original component based on the designs from “How to Add Dead-Time to PWM” [15]. The designs from the source were modified for our specific application. When a PWM signal is passed through the dead time adder two separate signals are produced, one of which goes negative slightly after and goes positive slightly before the other. This behavior can be seen in the validation section, figure 13.

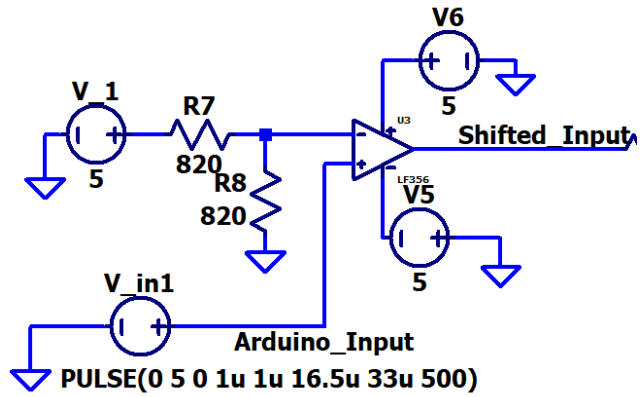


Calculations:

The behavior is mathematically pretty simple. The equations for the behavior of all of the blocks are included below as well as the values used.

Voltage Shifter:

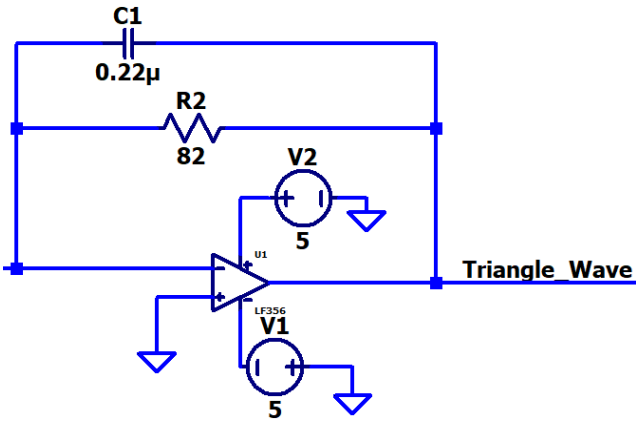
Values were calculated for a 2.5V voltage drop.



$$2.5v = 5v\left(\frac{820}{820 + 820}\right)$$

Integrator:

The optimal multiplier was determined through simulation.

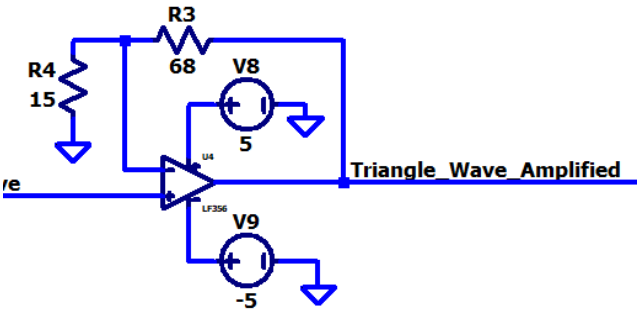


$$V_{out} = -\frac{1}{R * C}\int V_{in}dt$$

$$V_{out} = -\frac{1}{82\Omega * 0.22\mu F}\int V_{in}dt$$

Amplifier:

Amplification was set to scale voltage to standard audio voltage



$$Gain = 1 + \frac{R_F}{R_{gnd}}$$

$$5.53 = 1 + \frac{68}{15}$$

LC Filter:

The cutoff frequency for the filter was set to 20kHz, which is also the cutoff frequency for the treble filter. This design provides a theoretical attenuation of -10dB at the 35kHz carrier frequency.

$$f_c = \frac{1}{2\pi\sqrt{LC}}$$

$$20{,}000Hz = \frac{1}{2\pi\sqrt{1mH * C}}$$

$$C = 0.063\ \mu F$$

$$Q = R_L\sqrt{\frac{C}{L}}$$

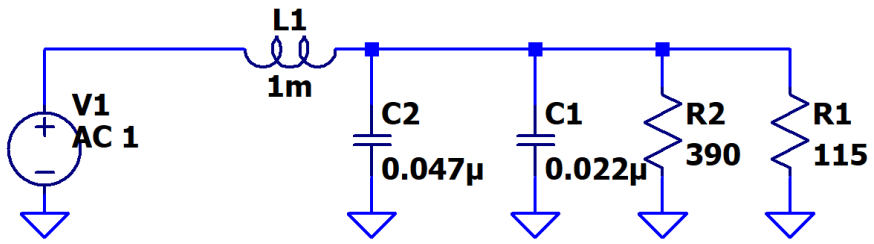
$$0.707 = R_L\sqrt{\frac{0.063\mu F}{1mH}}$$

$$R_L = 89\Omega$$

Load is 390Ω, therefore we must reduce resistance with a parallel resistor

$$\frac{1}{89\Omega} = \frac{1}{390\Omega} * \frac{1}{R_{reducer}}$$

$$R_{reducer} = 115\Omega$$



Measurements and verification:

Images of the results of the simulation and testing are included in the appendix. All stages function essentially as intended. Verification for the stages of the amplifier was done with a 10kHz sin wave. It can be seen that the LC filter does not perfectly filter out the carrier frequency. This causes the measured output to appear “thick.” However, the overall amplified sin wave is obvious and is close enough that any output audio is entirely legible. Stages are referenced along with their corresponding figures for verification below.

	Simulated	Measured
Voltage Shifter:	Figure 10	Figure 27
Integrator:	Figure 11	Figure 28
Amplifier:	Figure 12	Figure 29
Comparator:	Figure 13 & 14	Figure 30 & 31
Dead Time Adder:	Figure 15	Figure 32
Transistor Output:	Figure 16	Figure 33
LC Filter Output:	Figure 16 & 17	Figure 34 & 35
Shoot-Through:	FIigure 40	Figure 41
LC Filter FQ resp:	Figure 42	Figure 43

Arduino

The design for the functionality of the Arduino was separated into three main parts. First was to create a simple welcome screen with the project title and author names. The second was to deliver the input square wave PWM at the determined input frequency of 35kHz and lastly to design the spectrogram to read the changes of the voltage files from the three observed bands.

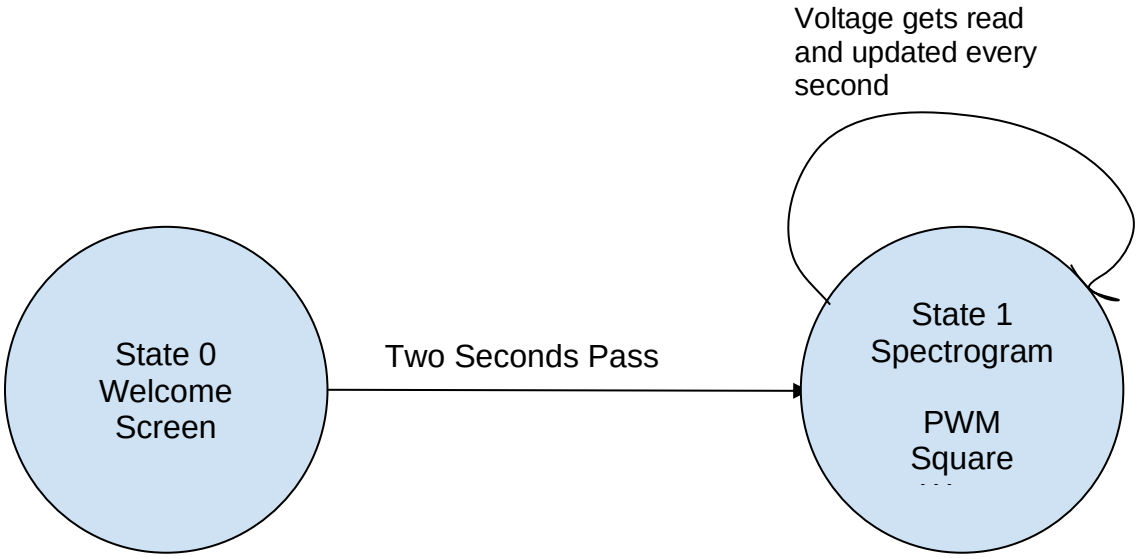
The approach to create the welcome screen, the hello world code from Random Nerd Tutorials [1] was used as a structure to understand how to initialize the OLED and display text correctly. Making sure to stay within the bounds of the libraries that are specified, the approach to create the welcome screen used was setting the cursor position and then printing the text to the OLED by using display.println. The result of the welcome screen on the OLED is shown in figure 37. The final implementation of the welcome screen.

For the Arduino to create the necessary square wave PWM input at the intended frequency of 35khz, an approach was found using the tone function from the Arduino site [5] where the function allows the user to select the pin to generate the wave and then set the frequency. A delay of two seconds is used for the wave being generated by the Ardiuno to become a square.

A large portion of the Arduino theory revolved around the working spectrogram. The Spectrogram created by Janux from hackster.io [10] was referenced to get an understanding of how spectrograms can function from an Ardiuno. The final approach was done where there

would be three large bars representing the three bands. When the voltage changed, the bars would move accordingly based on the input values the Arduino reads for the two second duration. The inputs for the voltage to be read from are defined as A0, A1 and A2 for the Bass, Mid and Treble in that order. The millis function learned from the Arduino reference site [13] is responsible for setting the new value that gets displayed with the time that is set for the voltage to be read. The map function also learned more about from the Arduino reference cite [12] is used to display the new voltage value of the bars once it needs to be updated as well. It serves the purpose of redrawing the bars in order for it to keep updating the values. The code also includes headers representing each band and defined limits so the bars don't go higher than they are intended to. The final design displays the welcome screen result shown in figure 37.Once two seconds pass, the spectrogram begins to display the bars representing the three bands and also sends the square input into the level shifter circuit within the class D amplifier. Figure 39.

Simple FSM Representing the states of the Arduino Implemented



Validation

The validation of each subsystem in the home audio system was done by testing each system alone then connecting the necessary subsystems together to ensure the input and output of each subsystem functioned as expected. This was done using LT spice and the physical components needed to build and test each subsystem.

The physical results of the filters for each band shown from figures 18 to 26 show that the slope results match the simulation tests and have similar gains. Where the filters have gain

control functionality where the gain can be boosted (>0 dB), cut (<0dB) or be in a neutral position where the gain is at 0 dB. The Arduino welcome screen output to OLED is represented in figure 36.

The modular circuit was tested by first inputting parameters on LT spice to mimic the square wave input that is set to the frequency of 35kHz.The simulated level shifter circuit allows a triangle wave to be generated as shown in figure 10 . Since the Arduino input is only positive, it shifts it down to range from a negative to positive value. The physical result of the level shifter is shown in figure 27. Figure 11 shows the simulated results of the triangle wave generated and the . The range of this triangle wave is from -2.5 to 2.5 volts. Figure 12 shows the triangle wave amplified to obtain those values and figure 13 shows the sin wave that is an input with the triangle wave from the modular circuit. The physical result of the triangle wave being generated is shown in figure 16 and 28. The comparator circuit simulation results are represented by figure 13 previously mentioned, which shows the triangle wave output and sine wave input going into the comparator.The output is shown in figure 14 where the low areas of the sine wave create longer pulses and the higher areas have the opposite effect. Which is the intended output for the modular circuit.

The switching was tested and validated using an op amp-based circuit that turned off and on based on the current voltage of the wave being observed. Which also includes the necessary transistors. The simulation result of the switching output stage is shown in figure 15 and the physical result is shown in figure 17. The results indicate that the two waves are never negative at the same time so shoot through current is prevented within the system.

The LC filter output is simulated results are shown in figures 16 and 17 and the physical result of the system and the LC filter is shown in figure 18. As mentioned in the graphic amplifier circuit, the filter doesn’t perfectly filter the audio frequency but when testing was done all audio heard from the system was legible and functioned as needed for the home audio system.

The spectrogram code and output are found in figures 38 and 39. The spectrogram was validated by testing individual bands and ensuring the bars matched what was occurring with the input from the graphic equalizer.

Modifications

The main modification that occurred with the Ardiuno was with the design approach visually and the method that was used to code the spectrogram within milestone 4. The design approach was originally based on the frequency rather than the reading the voltage difference as specified in the design requirements. Which quickly needed to be modified in order to have the entire system functional. Instead of trying to find the frequency changes within the audio, the code was changed to take inputs from the voltage being observed and show the changes/peak value found within the wave by updating the bar every second. To represent the voltage changes within the system. As a result of the change, the visual of the spectrogram changed from having 128 bars to originally show the frequency changes to having only three larger bars with a label above that represents each of the bands for the audio system.

Design modifications were also made in the development of the filtering system. Originally gain was controlled via potentiometers built into the feedback of the actual filters, not with a separate gain module. In testing, this proved to be a poor choice as changes to the gain control would also affect the exact frequency range that the filter operated over. The issue was enough of a problem that we decided to rework the filters and implement the current system. Another somewhat late modification was the addition of increased parallel resistance with the potentiometer. In testing, it was found that when the potentiometers were turned all the way down the near-zero resistance provided extremely high attenuation, however, the maximum resistance of 10kΩ was not enough to provide appreciable amplification. The addition of the parallel 2.7kΩ resistors means that the maximum amplification is increased in exchange for less, but still satisfactory, maximum attenuation.

Overall System Results

Overall system results are nearly identical to the simulated behavior and meet all the requirements of the project. Validation of all of the individual subsystems is provided in each sub-system section and the appendix. In order to validate the system as a whole, a real audio wave will be sent from one end of the system to the other. The following table explains the behavior of the system at each step with references to pictures in the appendix. In addition, a video of the performance of the system is linked below. The Arduino subsystem was verified by checking the input wave going into the level shifter and ensuring the voltage is changing with the correct behavior when the equalizer dB value is modified.

Figure 44	An audio wave viewed from the oscilloscope. This wave is from the output of the graphic equalizer.
Figure 45	The audio wave is converted into the 0-5v voltage pulses at the output of the switching stage. Note that high points on the wave correspond with short pulses and low points correspond with long pulses. This is exactly the desired behavior
Figure 46	The final output of the amplifier. Although there is noise in the wave is is very clear that the audio signal has been greatly amplified. The previous roughly 500mv audio wave has been converted into a 5v wave

Video demonstration of system operation:

https://photos.app.goo.gl/wn2KuvpYAwTF2X7z6

Future Work

Future work on the home audio system would consist of improving upon certain subsystems and taking into consideration different implementations that can be done to make a more advanced spectrogram. For the graphic equalizer, the mid filter can be improved upon within the circuit since the simulation results show that the drop off compared to the other two bands isn't as steep which can affect the performance of the mid band. Improving the results within the simulation first and testing the differences with the implemented system could allow the performance of the mid band to improve with a better slope. The triangle wave results for the integrator circuit can be improved by adjusting the range from 0 to 3 volts, as opposed to the current ranges of -2.5 to 2.5 volts. The results of the system don't seem to change when the triangle wave ranges from a negative to positive voltage value, but for better practice, since negative voltages tend to be frowned upon unless necessary for the device being created, the system can be modified to support a positive triangle wave result from the integrator circuit while also keeping the intended results of the output for the class D amplifier. For the spectrogram and Arduino code within the system, using different libraries such as using FFT based ones can be explored to improve and change how the spectrogram can function. For example, the audio frequencies being inputted into the system can be explored by creating a new spectrogram

using FFT based libraries as mentioned in order to better understand how the audio system processes the frequency and how the Ardiuno reads the input frequencies and displays them on OLED. The welcome screen on the Arduino can visually be worked on by including animations within the titan screen to have a creative layout for the system to look more professional. The LC filter could also have a better attenuation result which is something that can also be improved on within the system.

Conclusion

The project has given the opportunity to learn how individual subsystems learned in current and past courses can be combined to create an individual system as well as give the opportunity to work on building a complete system without being what the best method to do so is. Which allowed us to learn new types of circuits and methods of programming to reach the designated results for the audio system. As the later stages of the project were being worked on, it was realized that time was not being allocated during the week to give more time for debugging and testing when there were issues for the system which caused certain constraints when meeting the deadline for the requirements. If the project was done again, more time would be planned to create a working simulation so more can be documented when the milestones needed to be met. Too much time was being spent fixing flawed designs as opposed to trying different approaches that have a better chance of working. Better research with timers for implementing the dead time circuit could have been explored more efficiently as well since too much time was taken trying to figure out what design would be able to work effectively which limited the time there was to test the physical circuit.

Authorship Page

Introduction Page - Alex Stoffelmayr

High-Level Design - Alex Stoffelmayr

Graphic Equalizer - Alex Stoffelmayr

Class D Amplifier - Alex Stoffelmayr

Ardiuno - Mahdi Mountassir

Validation - Mahdi Mountassir

Modifications - Alex Stoffelmayr / Mahdi Mountassir

System Results - Alex Stoffelmayr / Mahdi Mountassir

Future work - Mahdi Mountassir

Conclusion - Mahdi Mountassir

Appendix and Work Cited - Mahdi Mountassir

Appendix

Figure 1: Bass Simulation for dB < 0

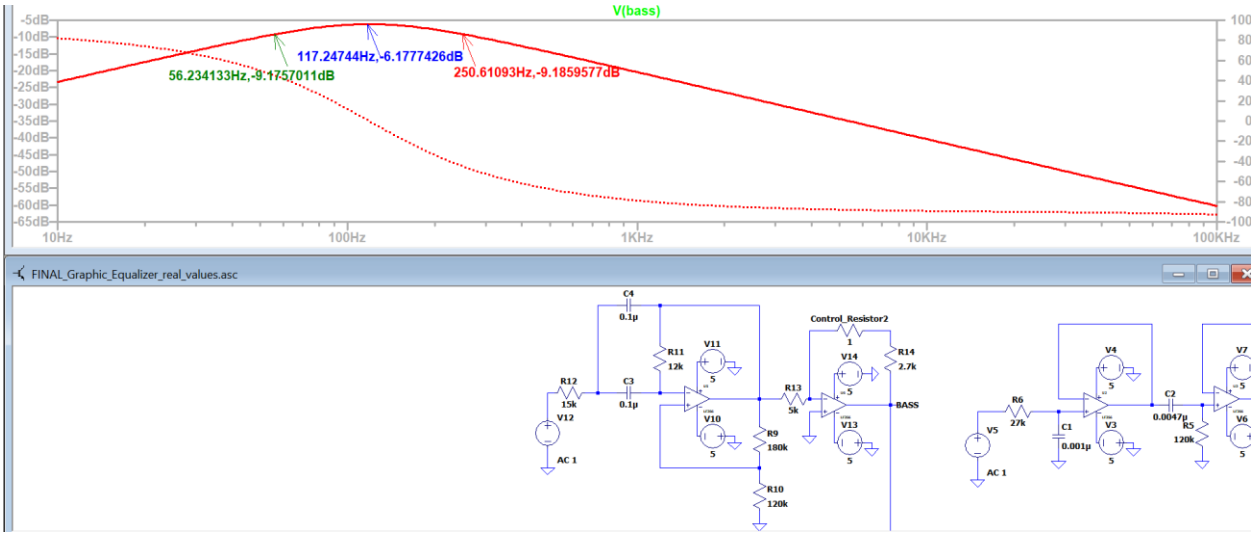


Figure 2: Bass Simulation for dB = 0

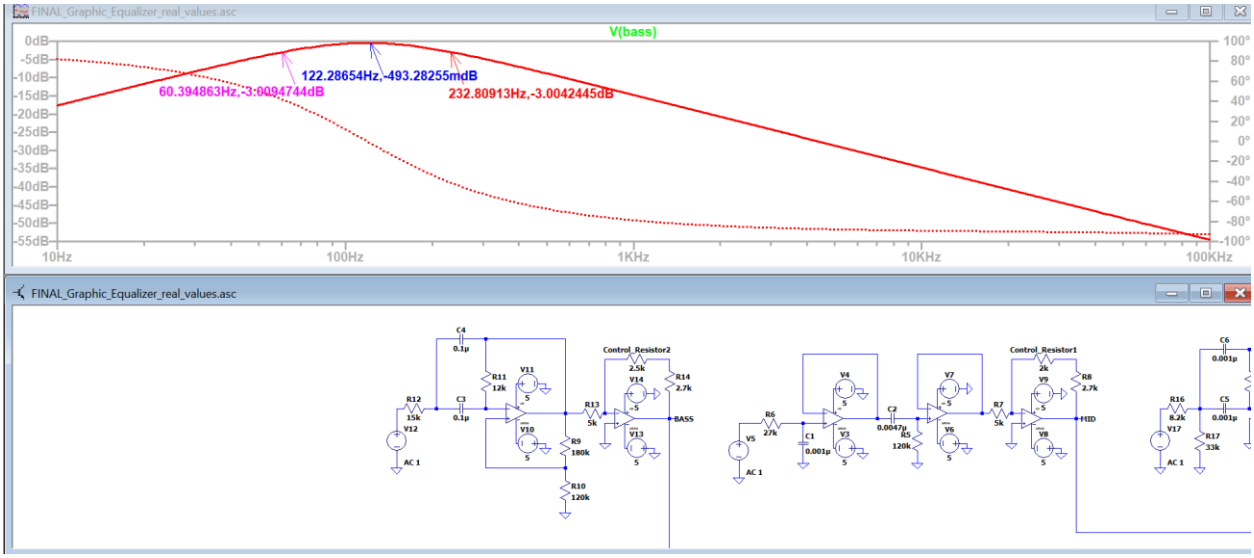


Figure 3: Bass Simulation for dB > 0

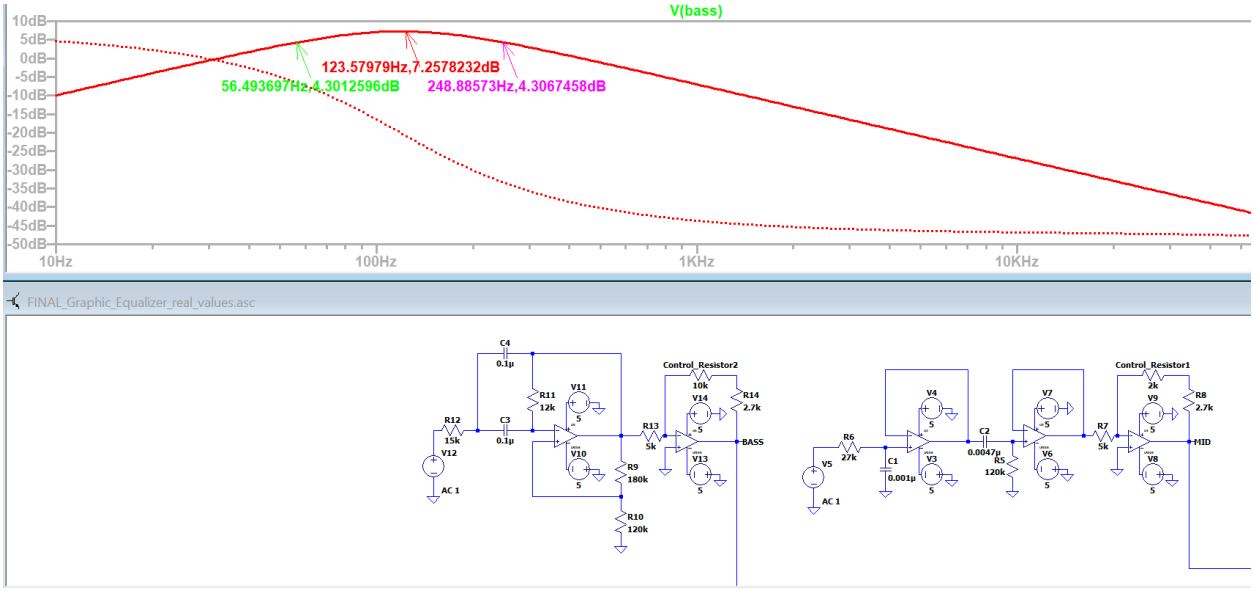


Figure 4: Mid Simulation for dB < 0

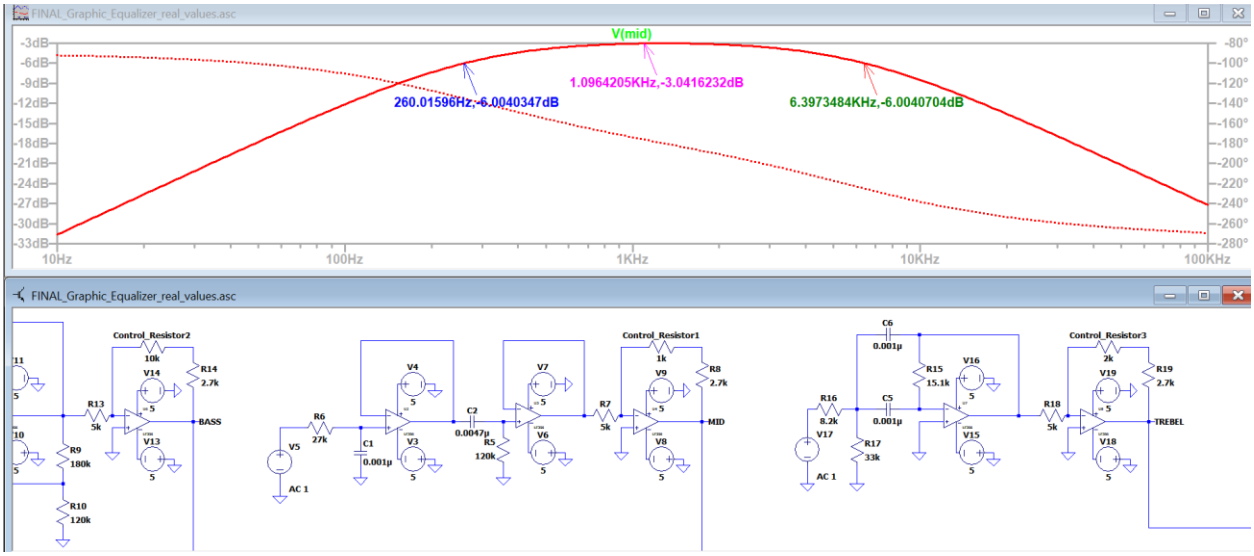


Figure 5: Mid Simulation for dB = 0

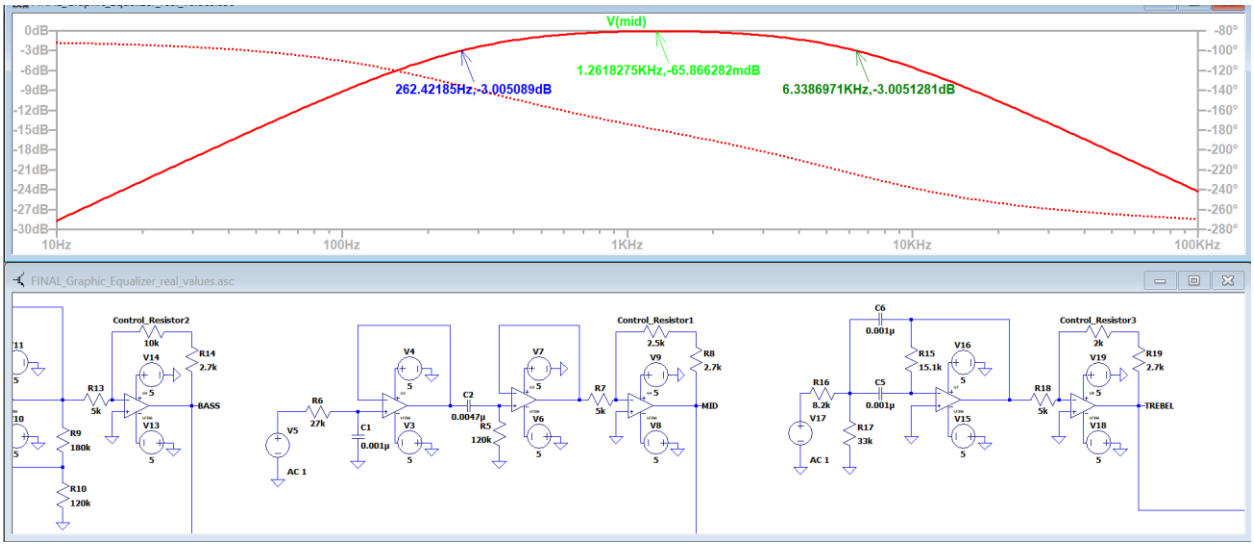


Figure 6: Mid Simulation for dB > 0

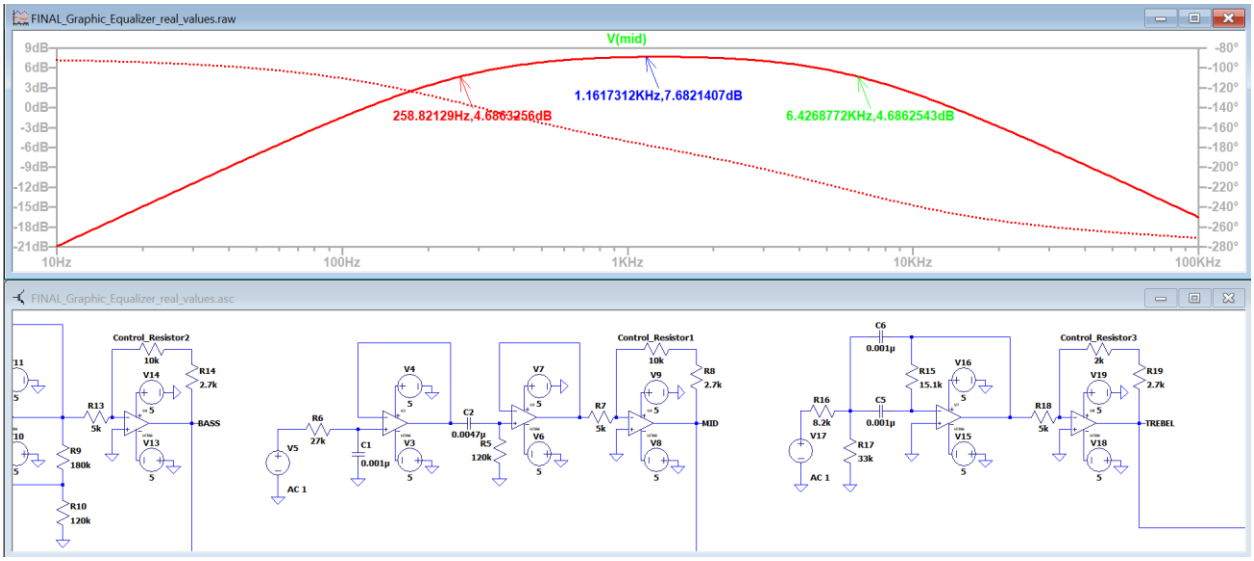


Figure 7: Treble Simulation for dB < 0

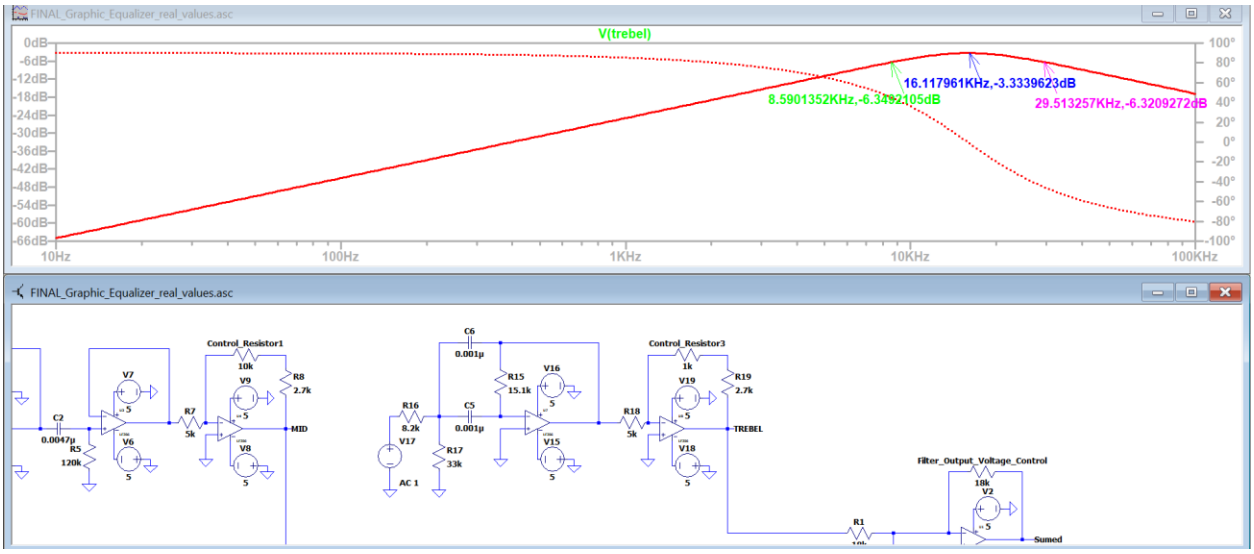


Figure 8: Treble Simulation for dB = 0

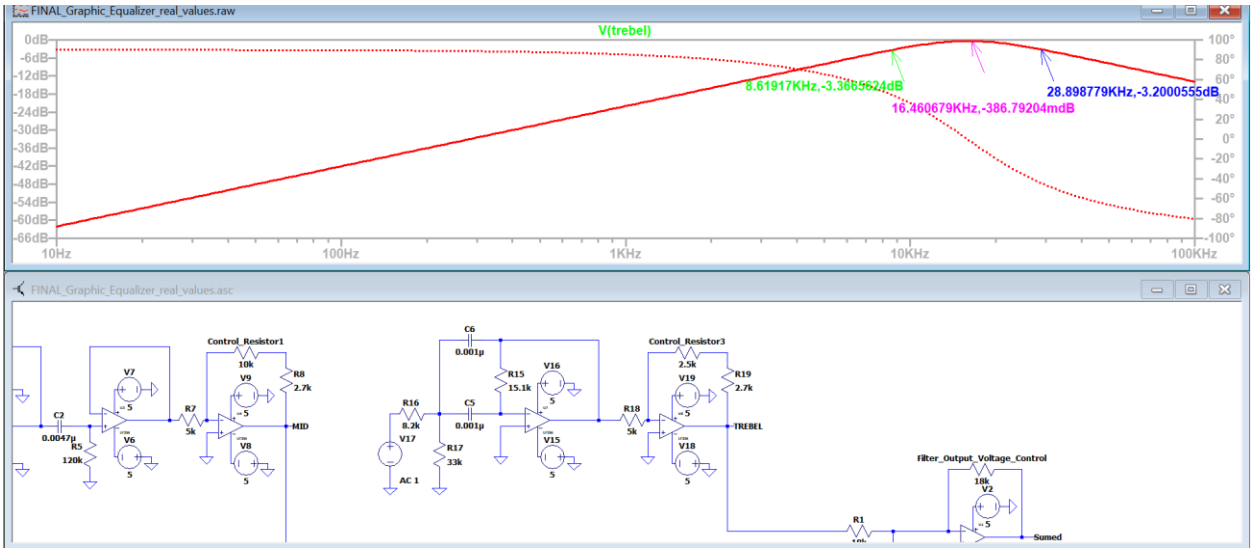


Figure 9: Treble Simulation for dB > 0

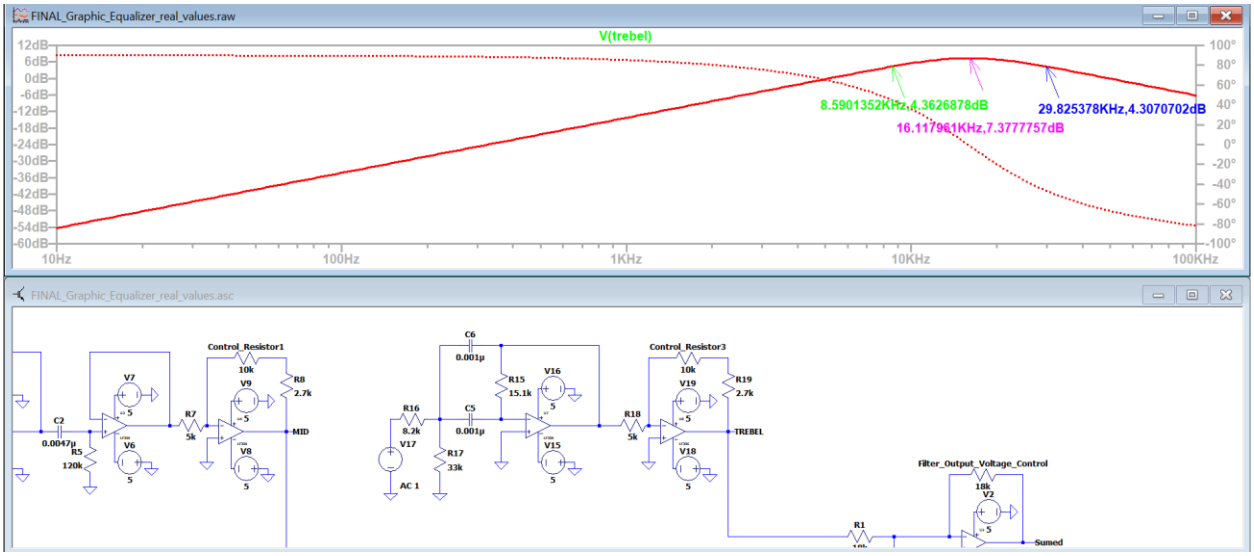


Figure 10. The arduino input is shifted down

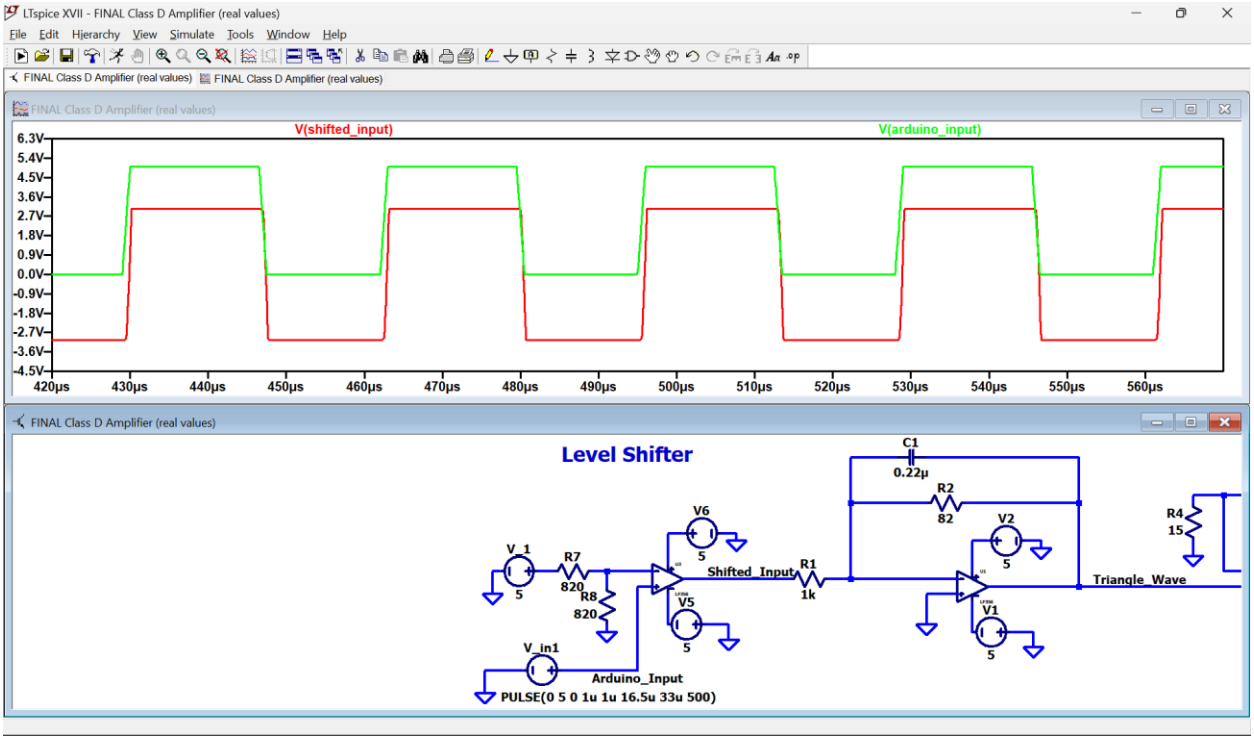


Figure 11. The square wave is integrated into a triangle wave

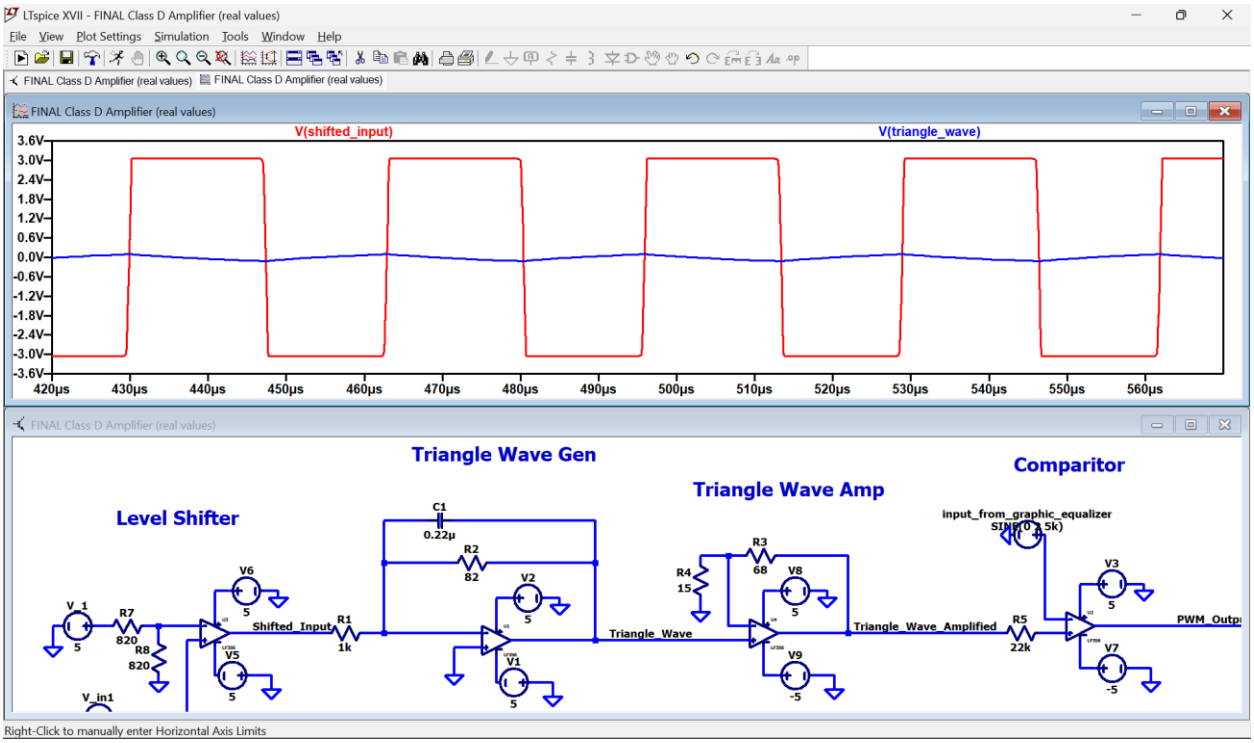


Figure 12. The triangle wave is amplified to roughly +-600mV

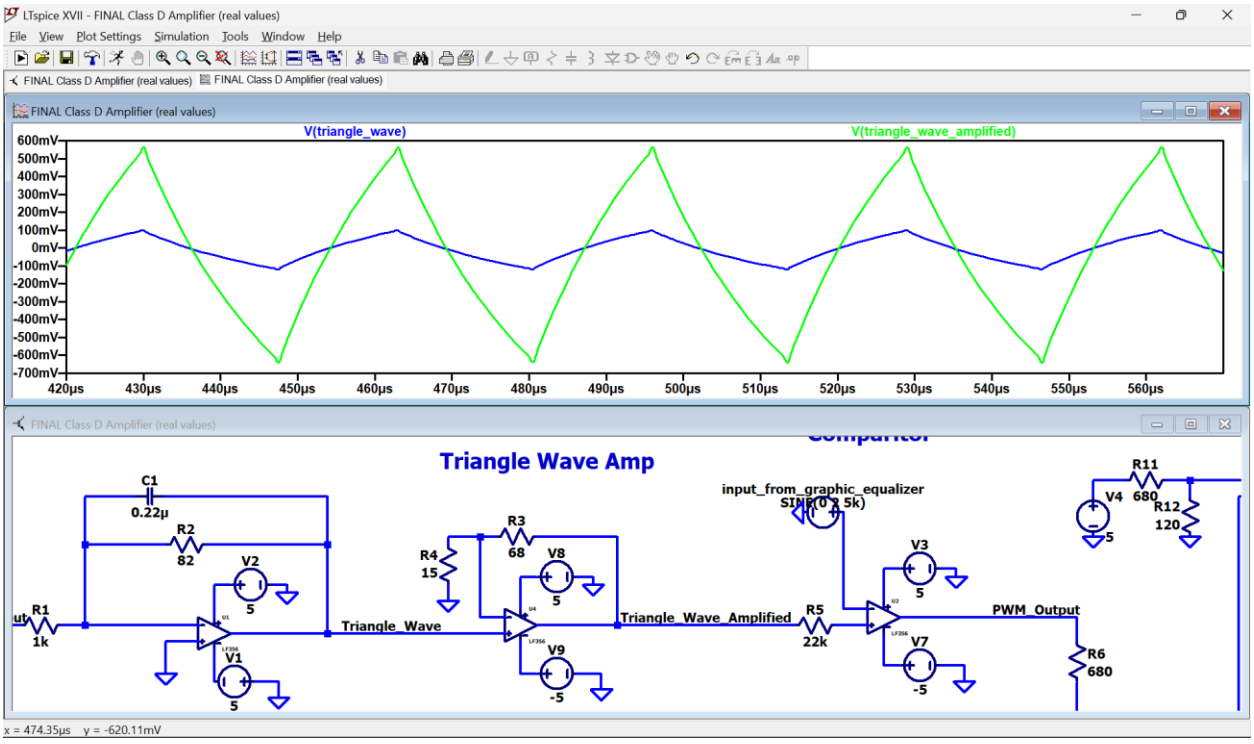


Figure 13. The triangle wave is compared with a 0.4V sin wave.

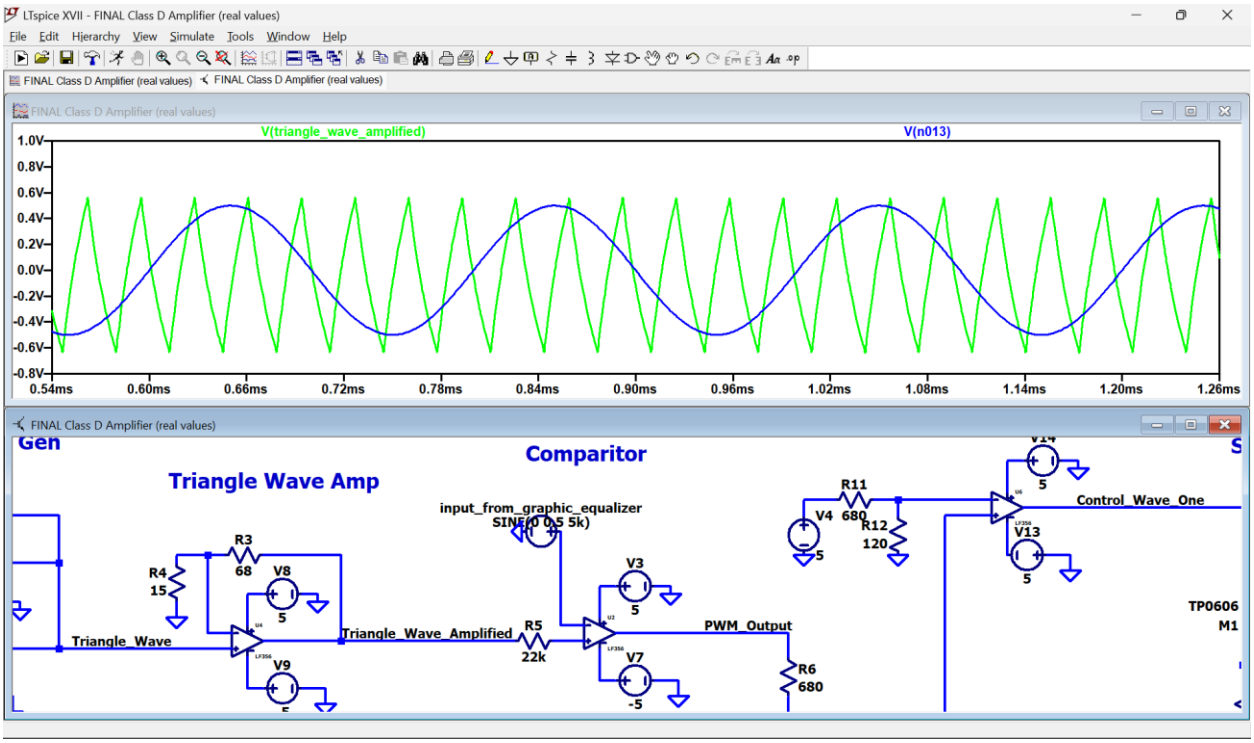


Figure 14. PWM wave Results. Low areas of the sin wave create long pulses while higher areas result in thin pulses.

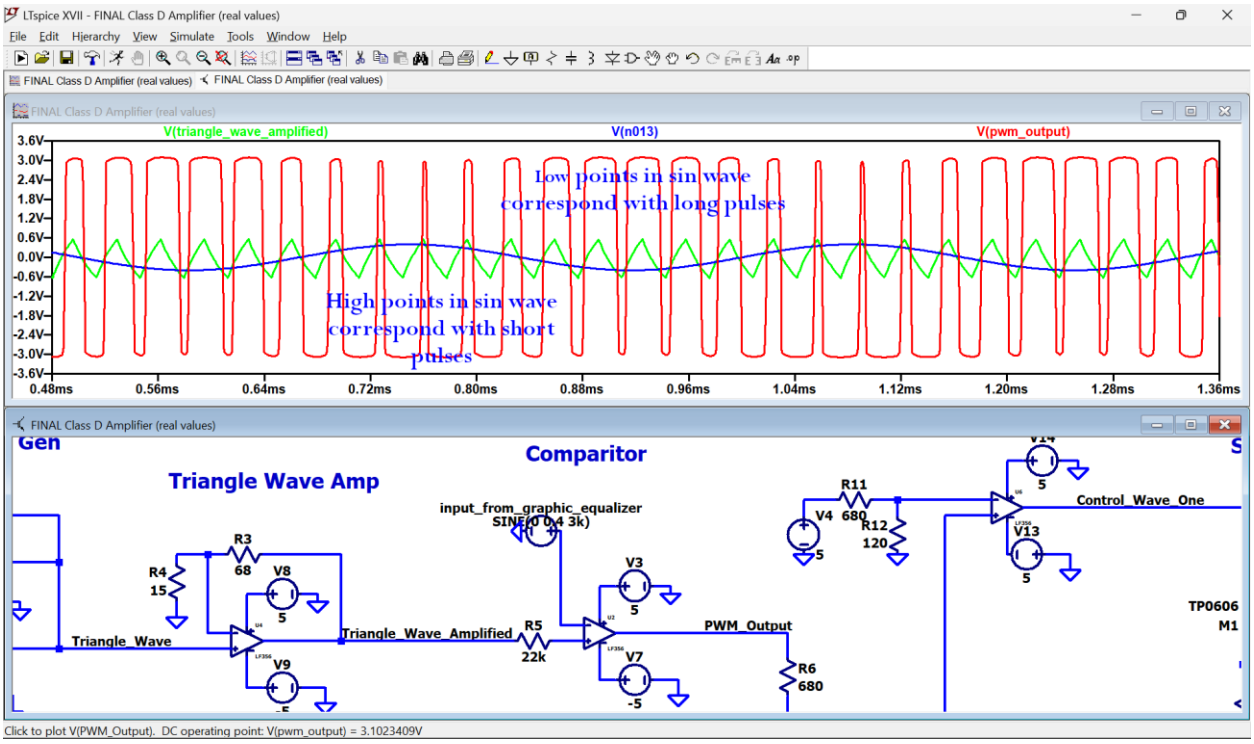


Figure 15. Dead time circuit simulation results

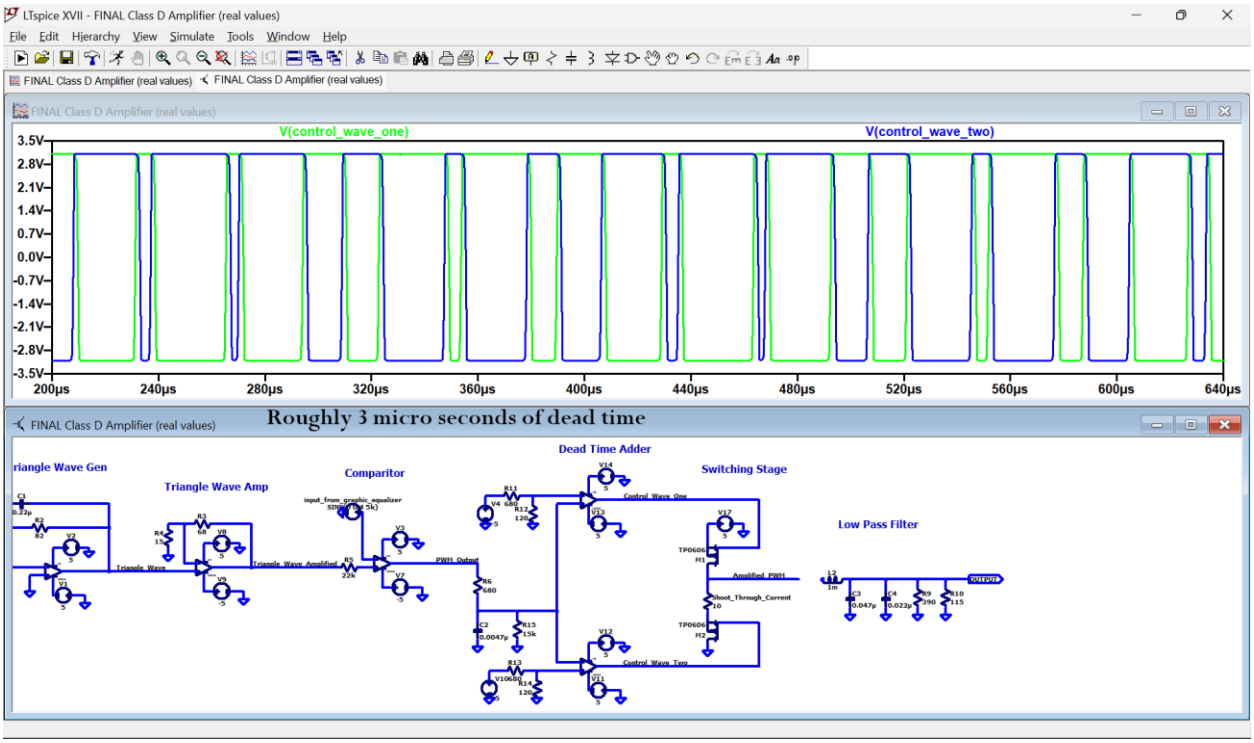


Figure 16. The output of the transistor switching stage along with the output of the LC filter.

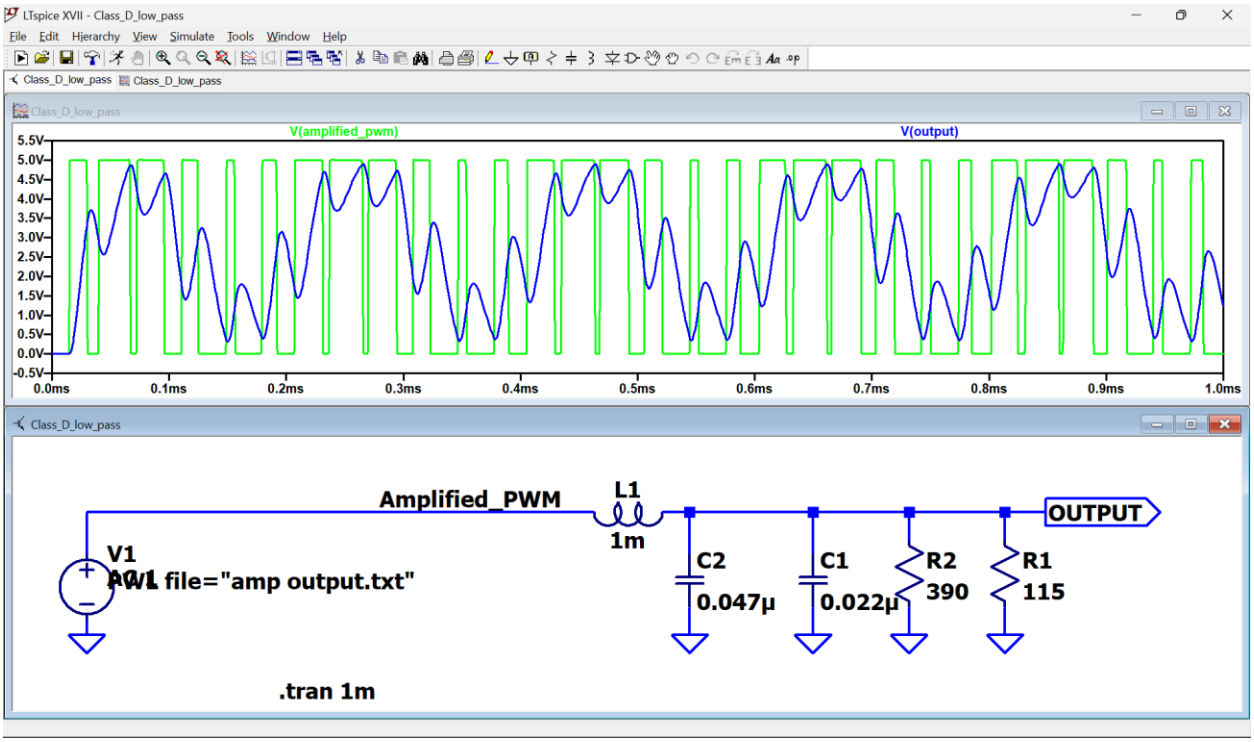


Figure 17. The amplifier output with the signal input.

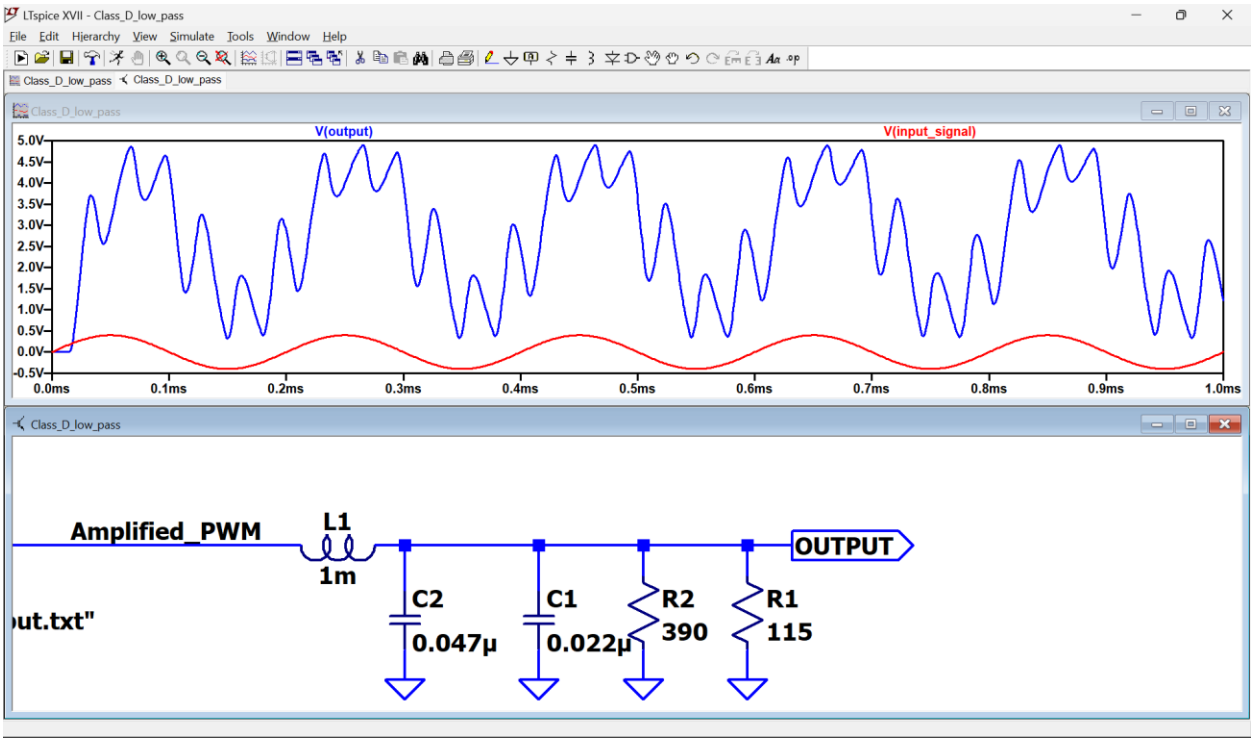


Figure 18: Bass Measurement for dB < 0

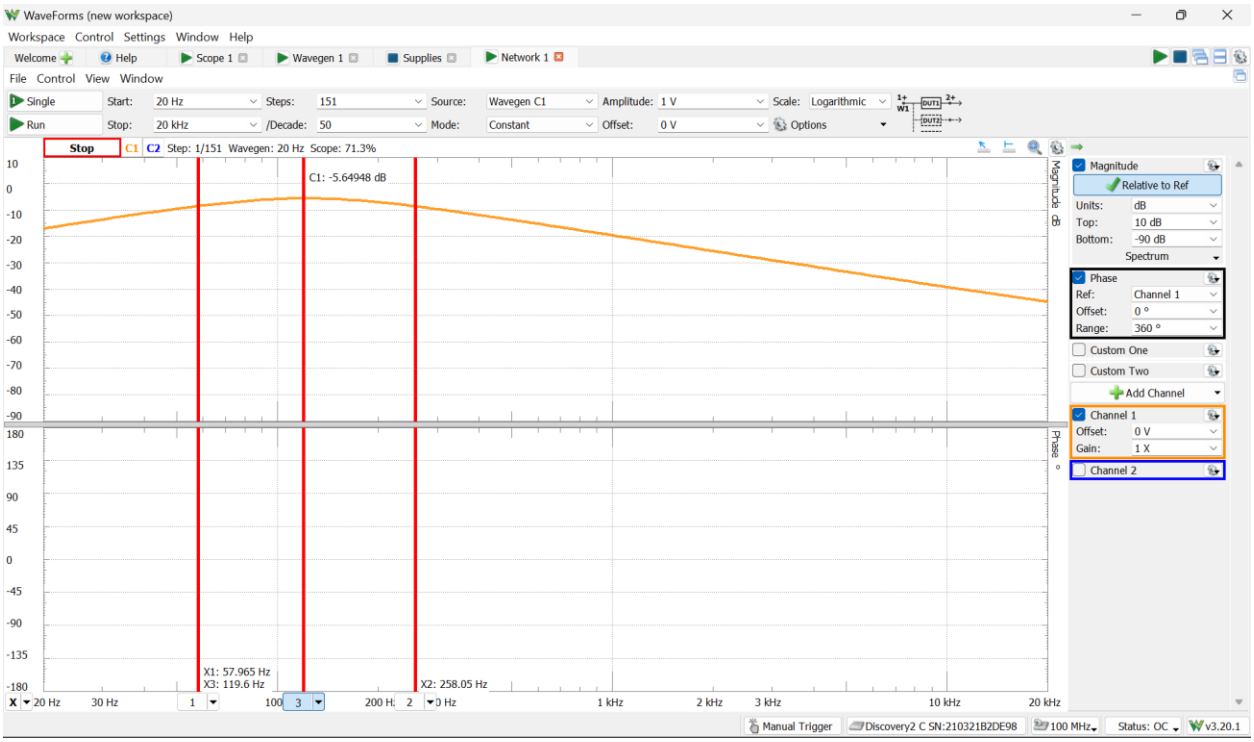


Figure 19: Bass Measurement for dB = 0

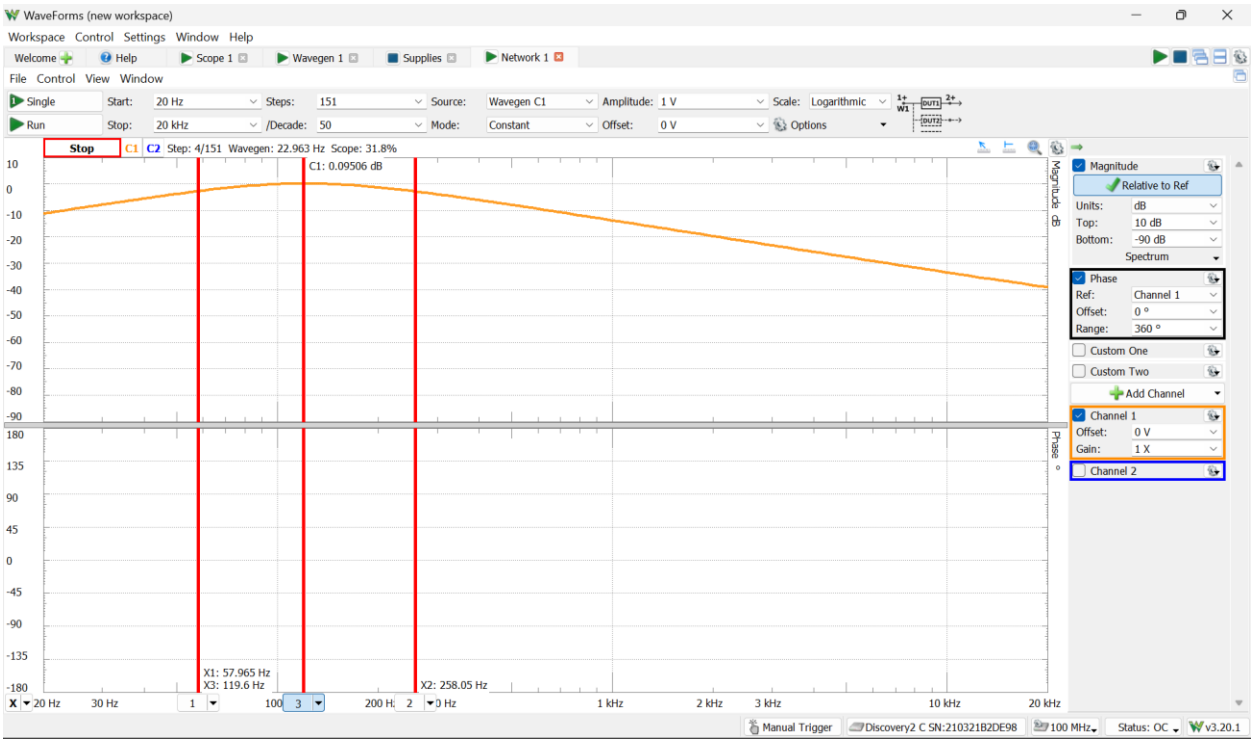


Figure 20: Bass Measurement for dB > 0

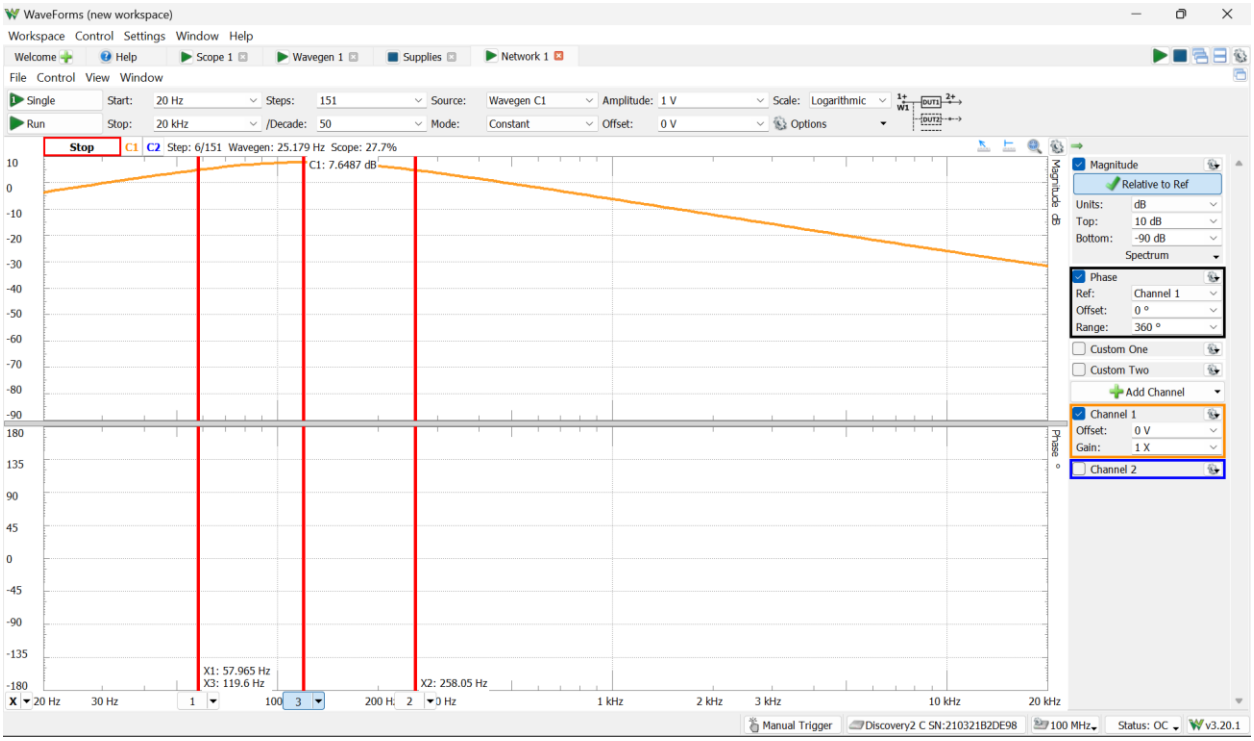


Figure 21: Mid Measurement for dB < 0

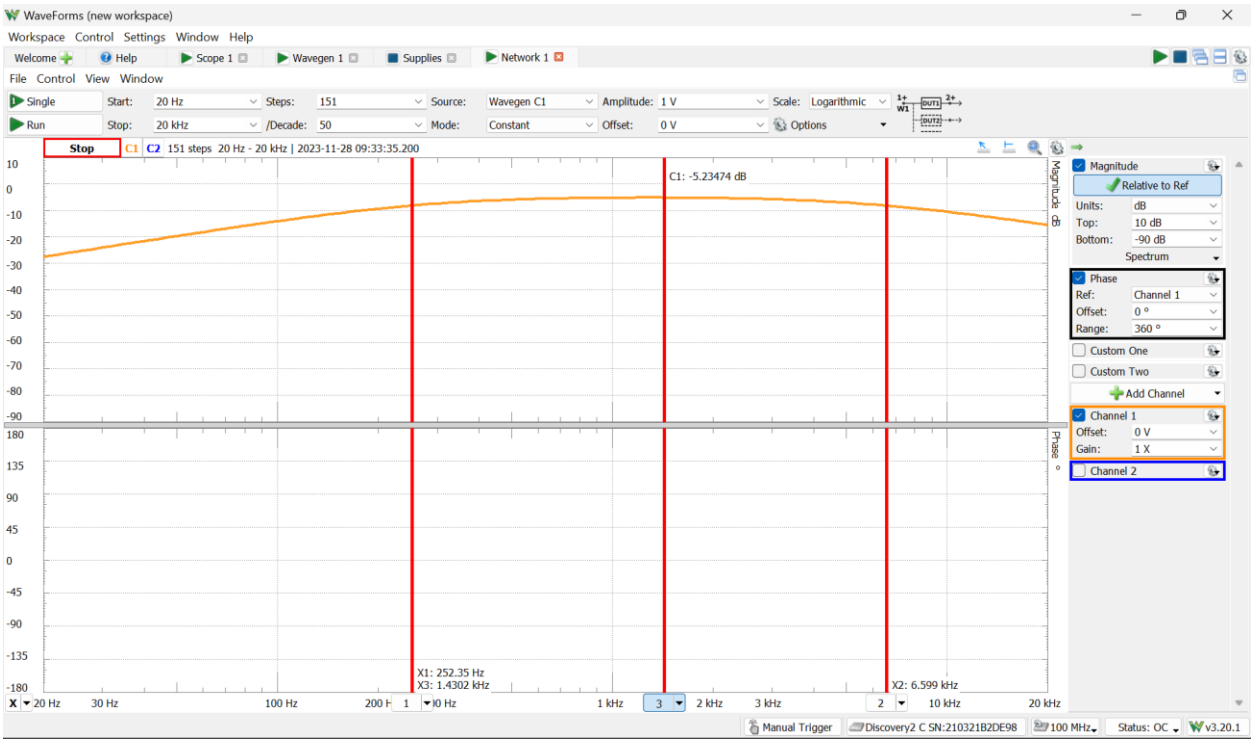


Figure 22: Mid Measurement for dB = 0

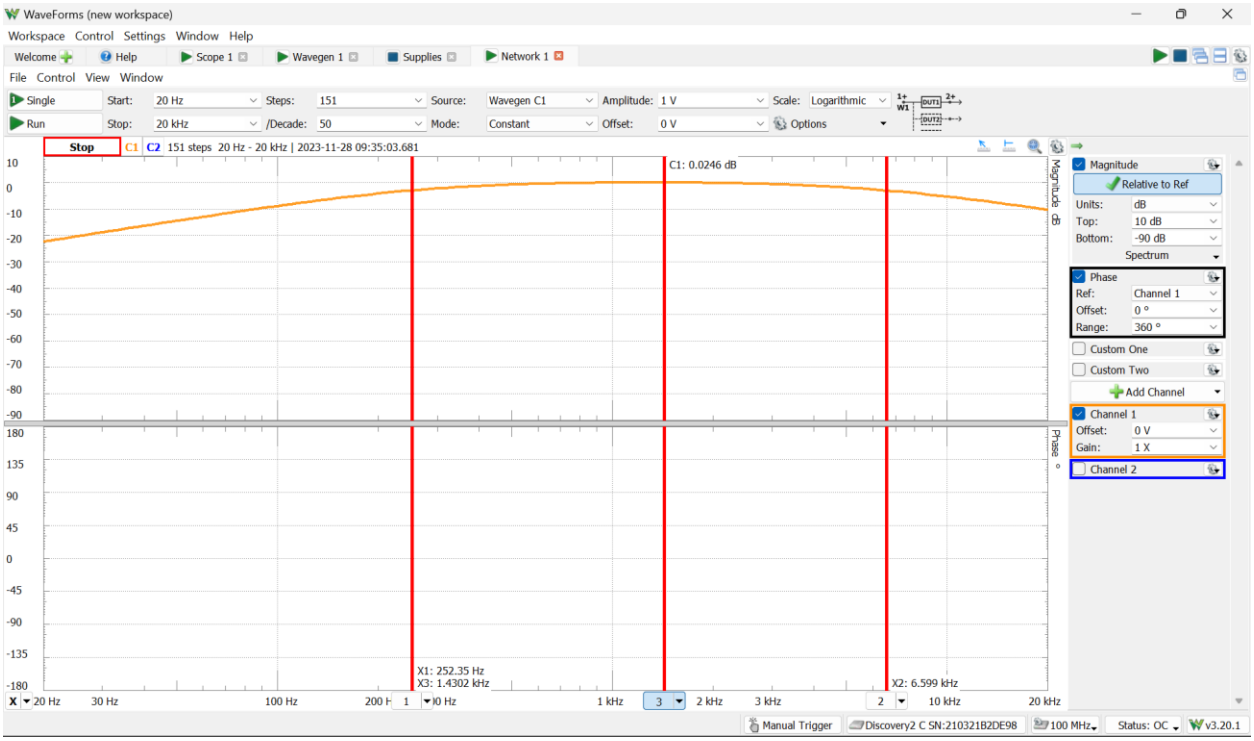


Figure 23: Mid Measurement for dB > 0

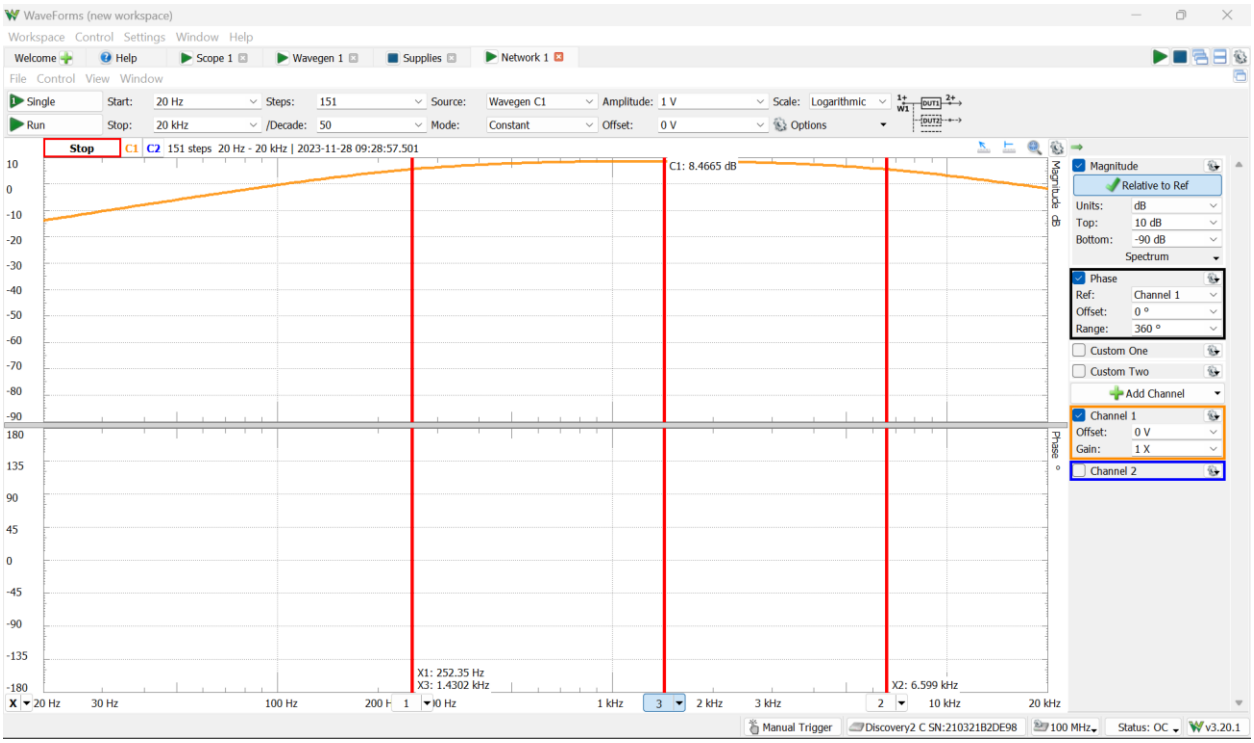


Figure 24: Treble Measurement for dB < 0

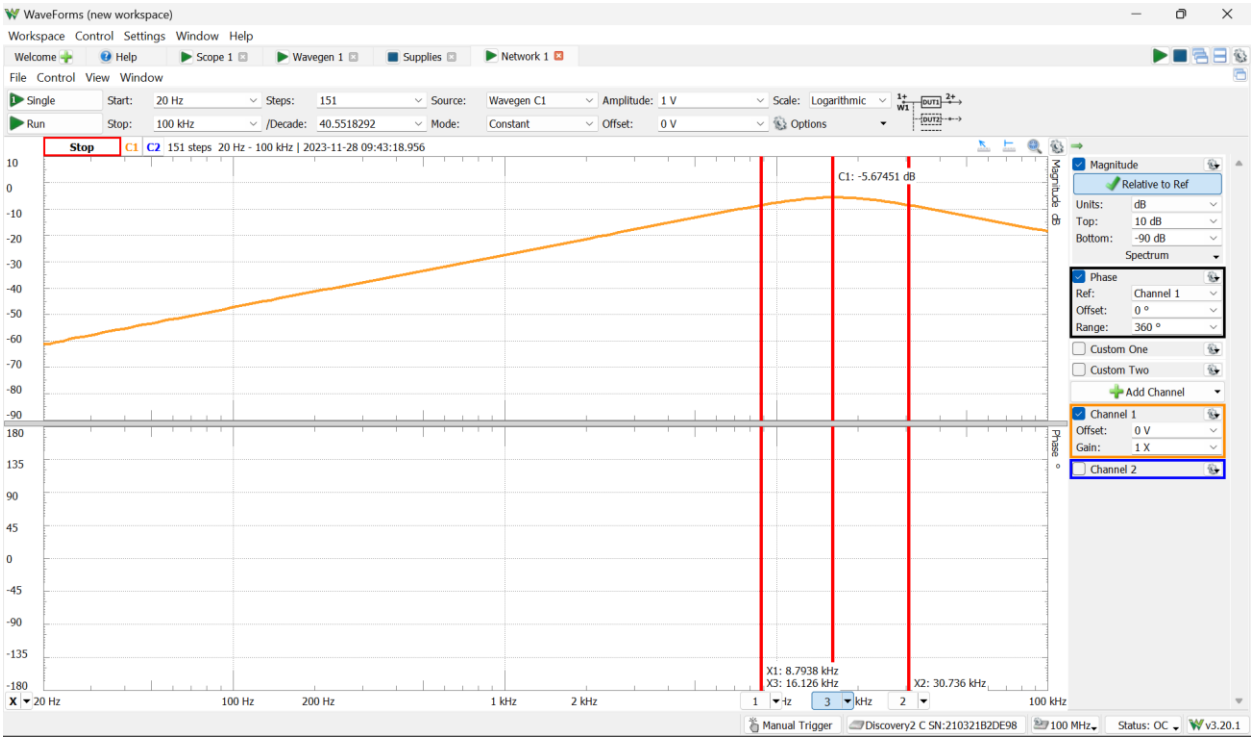


Figure 25: Treble Measurement for dB = 0

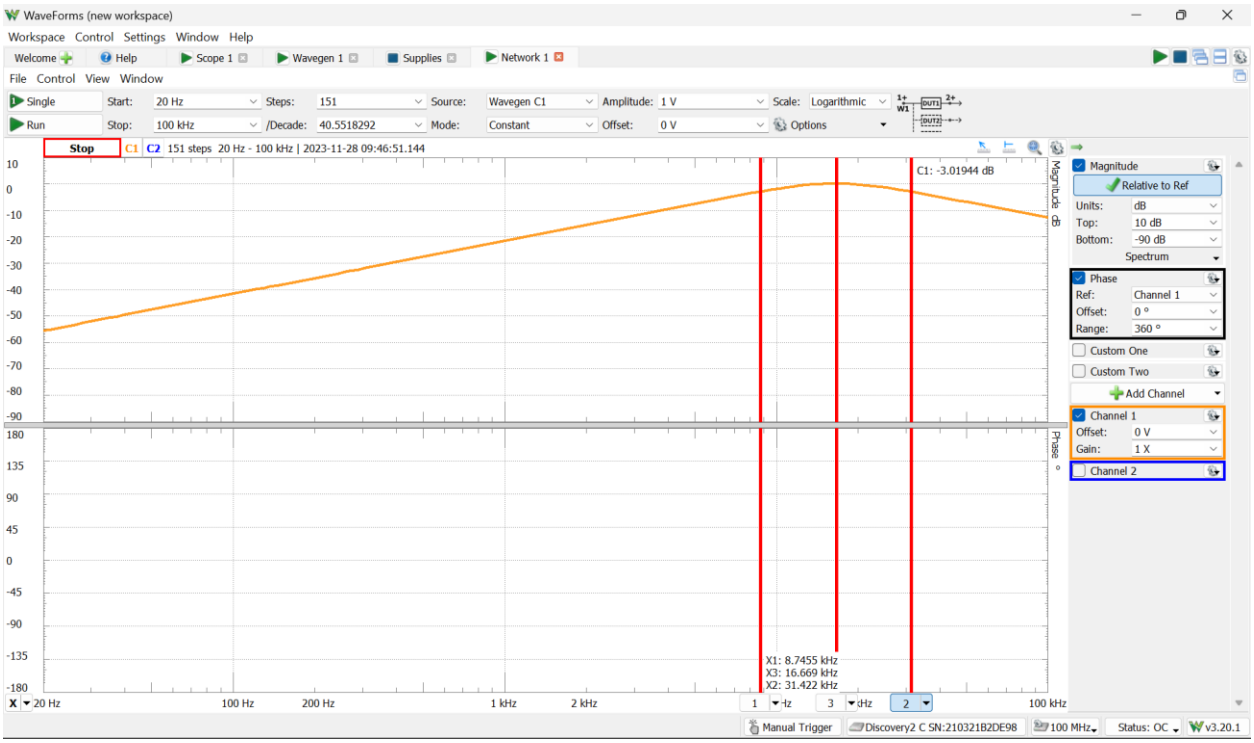


Figure 26: Treble Measurement for dB > 0

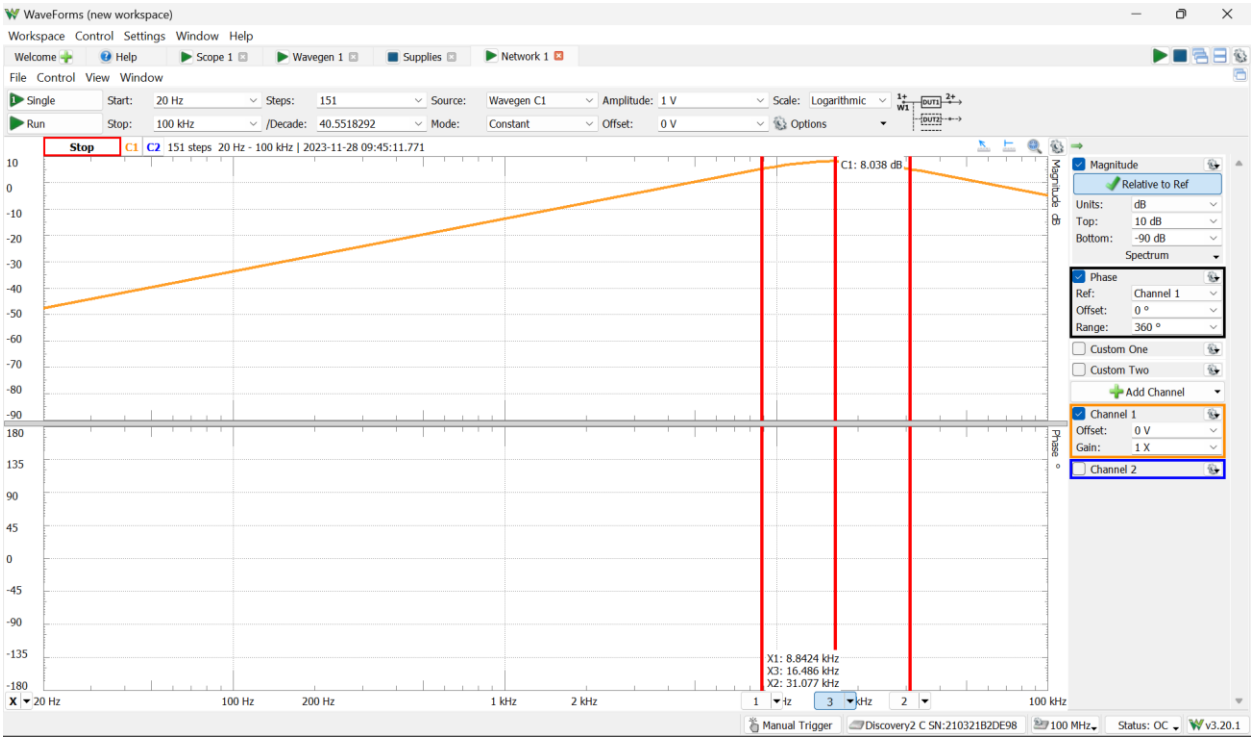


Figure 27. The Arduino input is shifted down

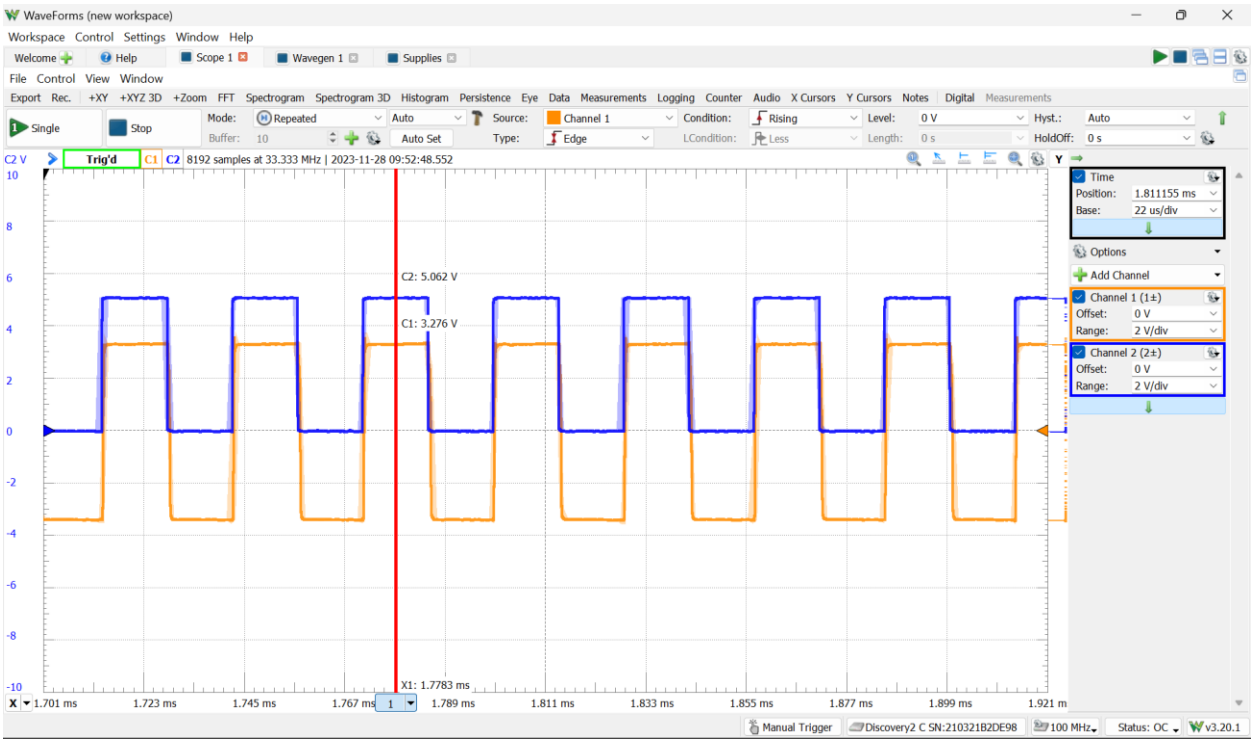


Figure 28. The square wave is integrated into a triangle wave

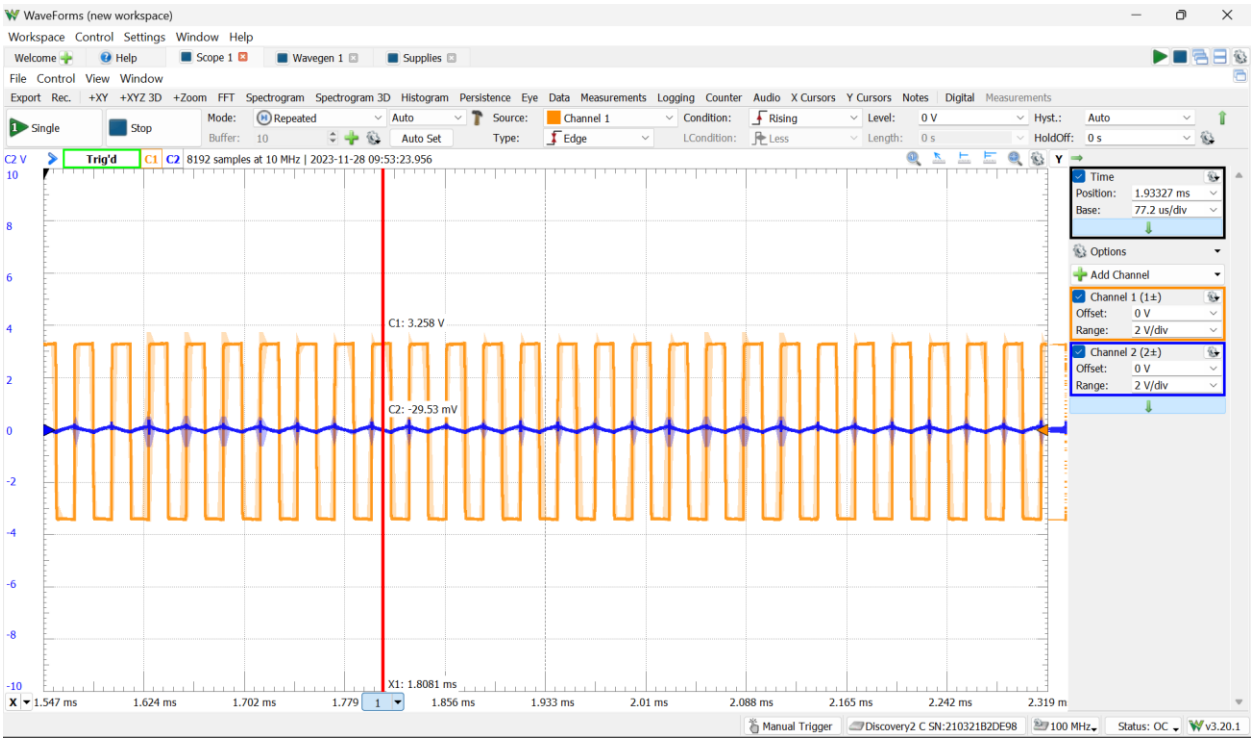


Figure 29. The triangle wave is amplified to roughly +-600mV

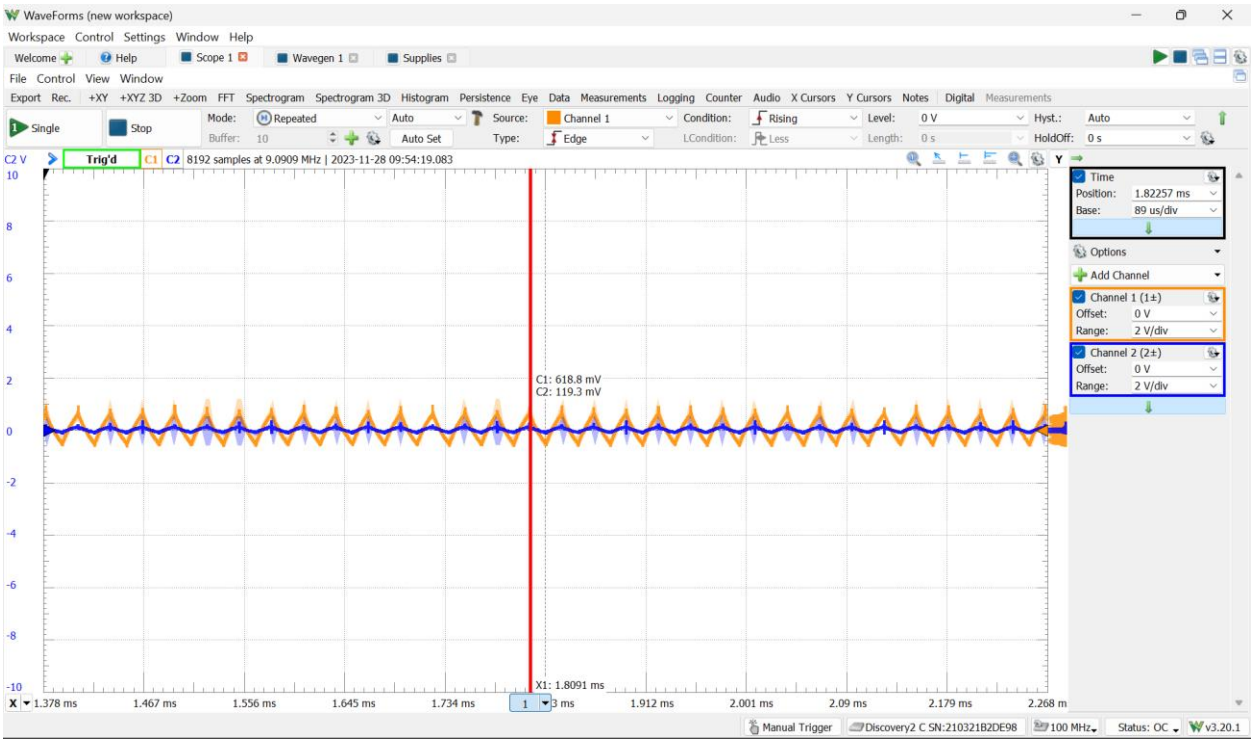


Figure 30. The triangle wave is compared with a 0.4V sin wave.

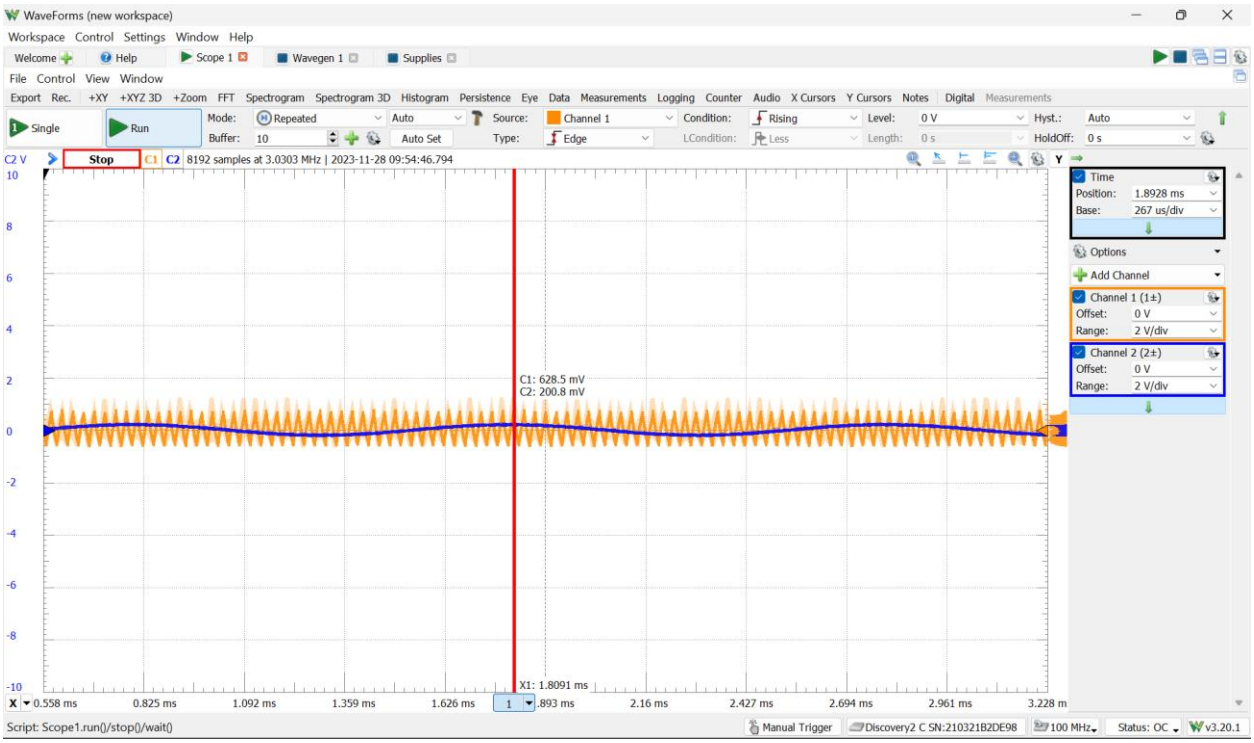


Figure 31. PWM wave Results. Low areas of the sin wave create long pulses while higher areas result in thin pulses.

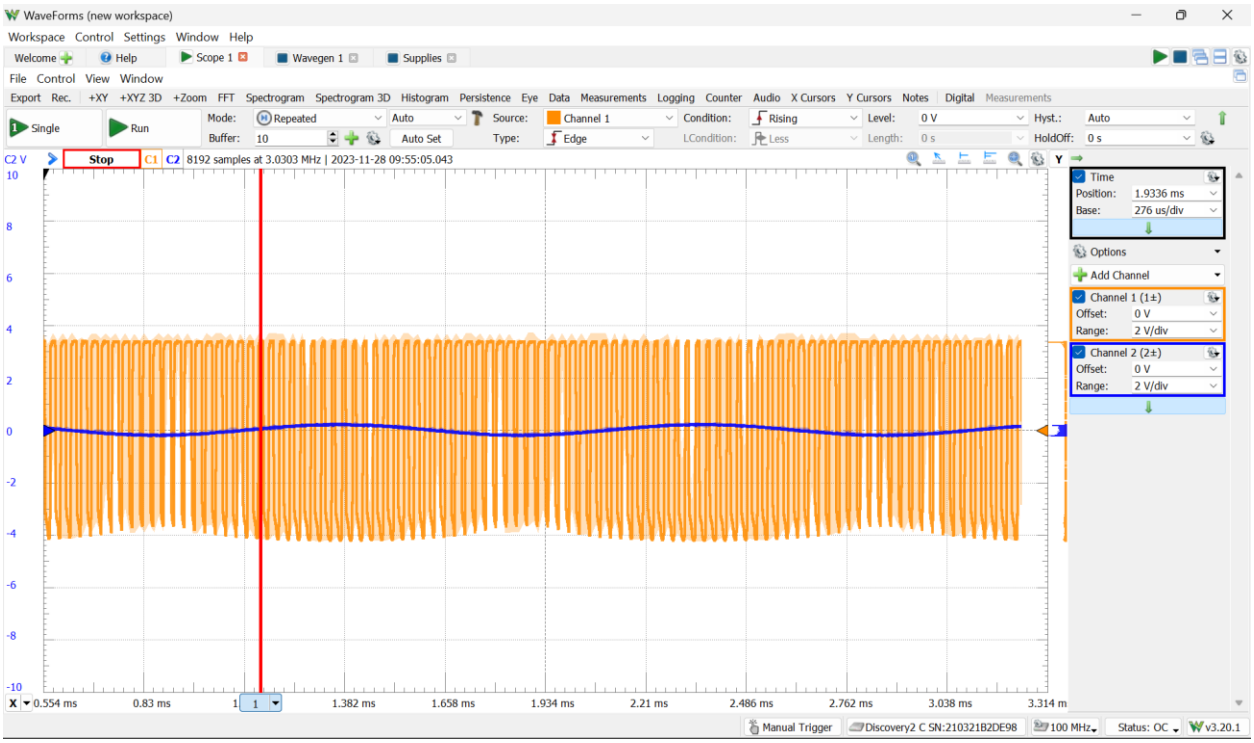


Figure 32. Dead time circuit simulation output

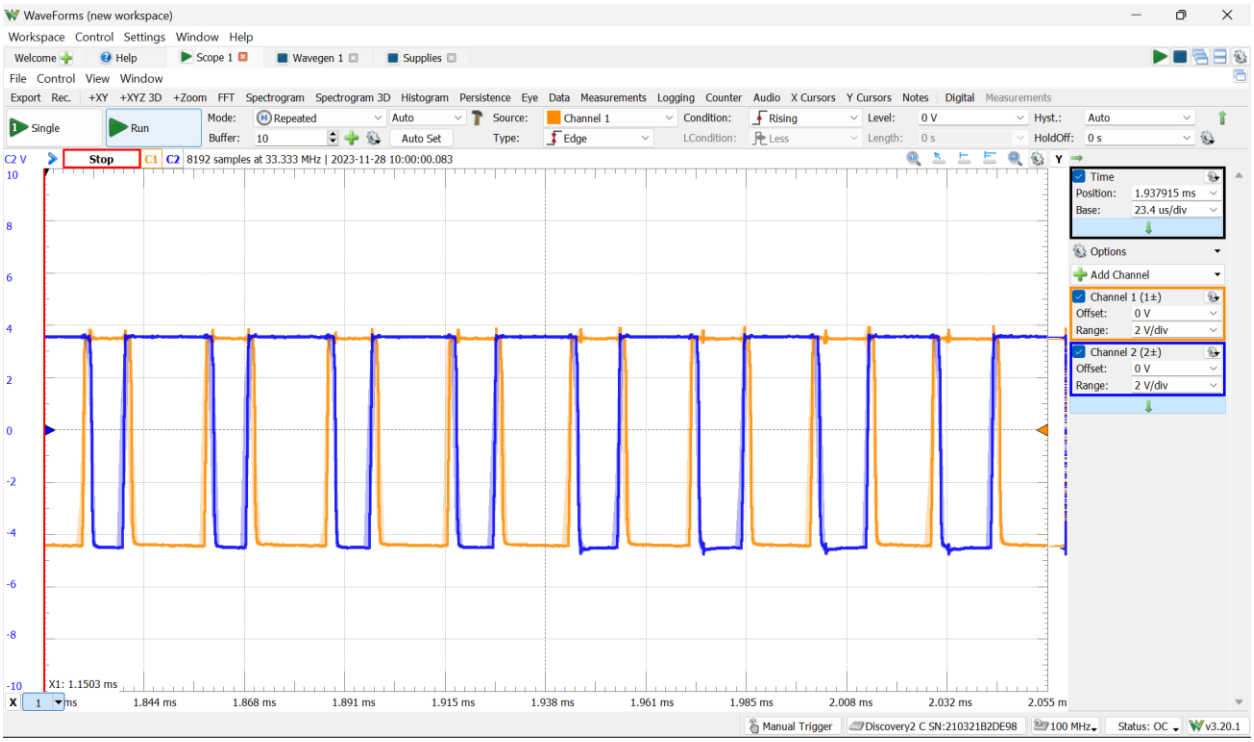


Figure 33. The output of the transistor switching stage

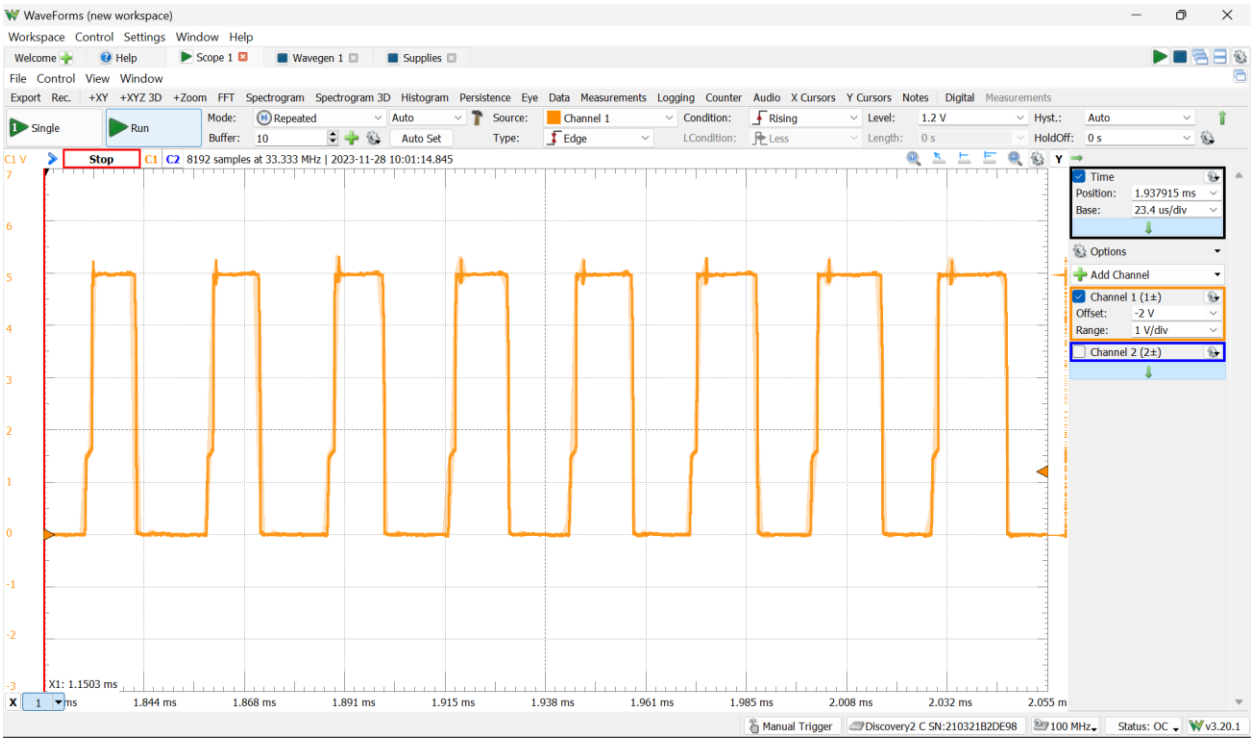


Figure 34. The output of the transistor switching stage along with the results of the LC low pass filter. Voltage pulses appear to go negative but this is just a quirk of waveforms.

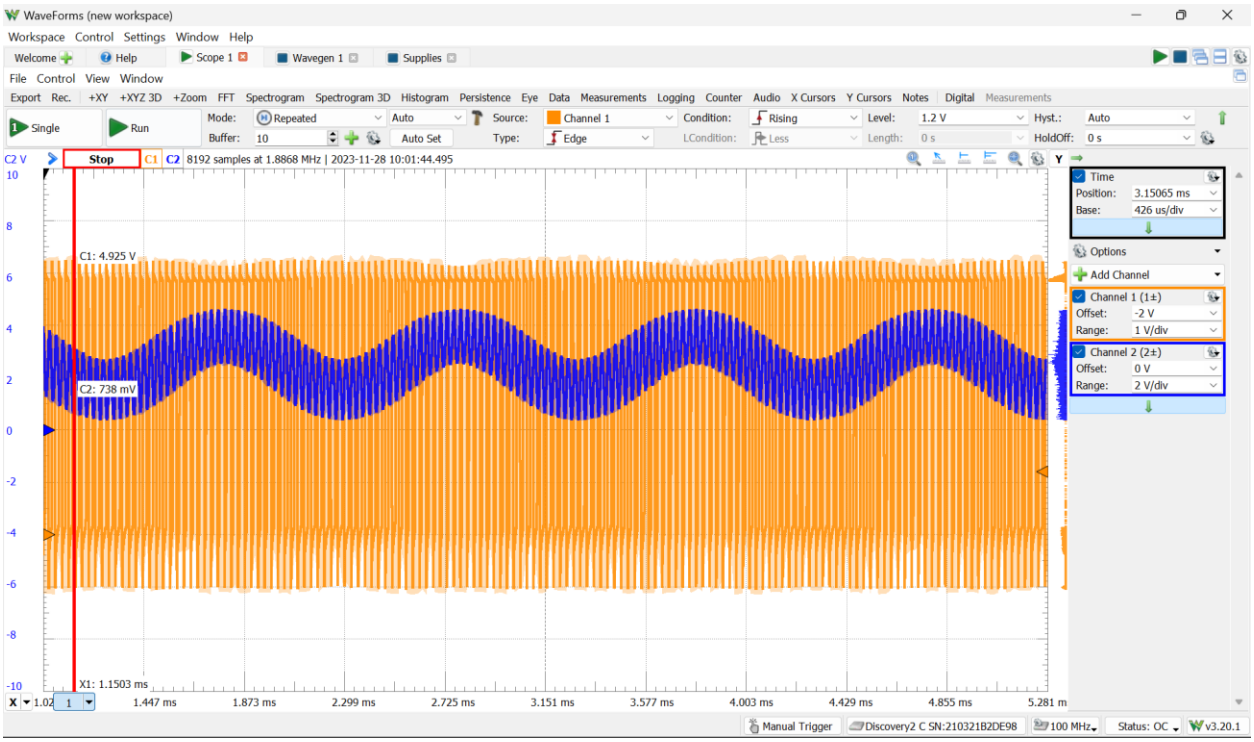


Figure 35. The final amplifier output with the original sin wave

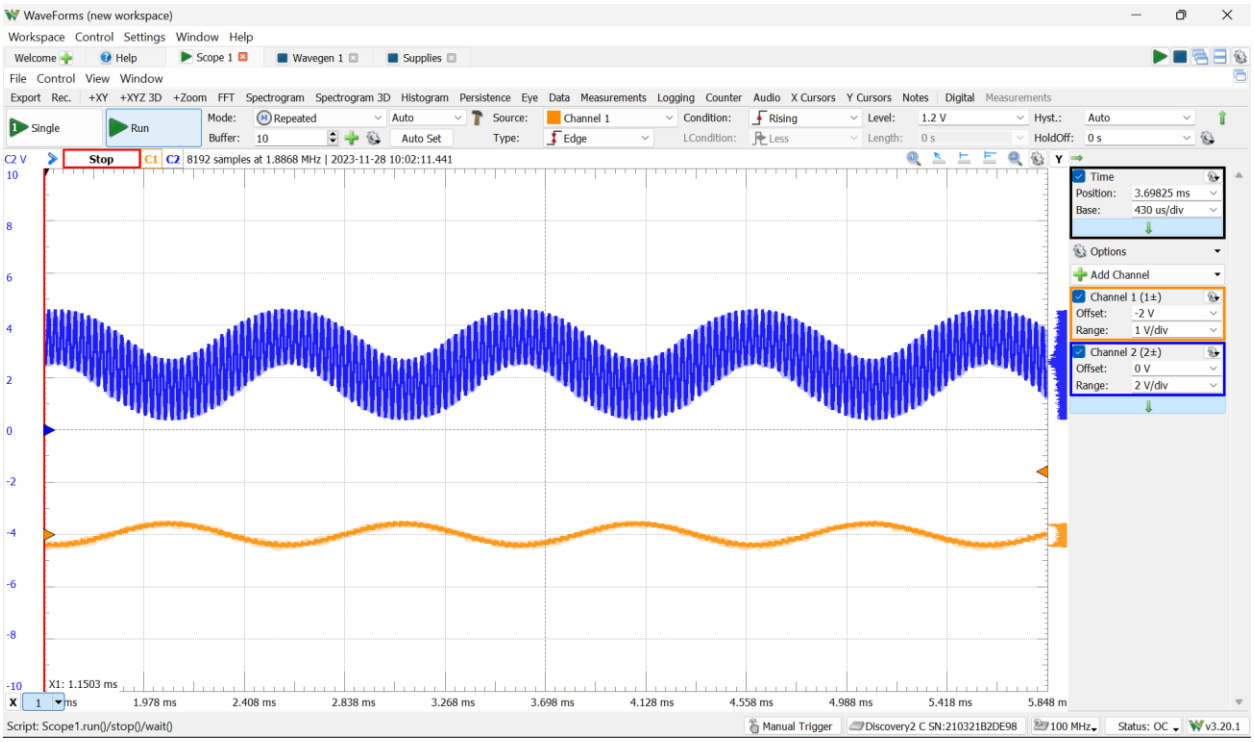


Figure 36. Welcome Screen code for the OLED to Display.

```
// Calling the libaries
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // Display Width for the OLED in pixels
#define SCREEN_HEIGHT 64 // Display Height for the OLED in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
  Serial.begin(115200); // serial port for the OLED

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { //  Calls The address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
      }
 delay(3000); // 3 second delay
  display.clearDisplay(); // clears the display each time the board is ran

  display.setTextSize(1); // smallest text size
  display.setTextColor(WHITE); // setting the textcolor
  display.setCursor(0, 10); // cursor location for the title of the project
  display.println("Home Audio System");
   display.setTextColor(WHITE); // Setting the text white again
   display.setCursor(0, 30); // Cursor location changed for the authors
   display.println("Mahdi M and Alex S");
  display.display();
}

void loop() {

}
```

Figure 37 .Welcome Screen Output Displayed on the OLED

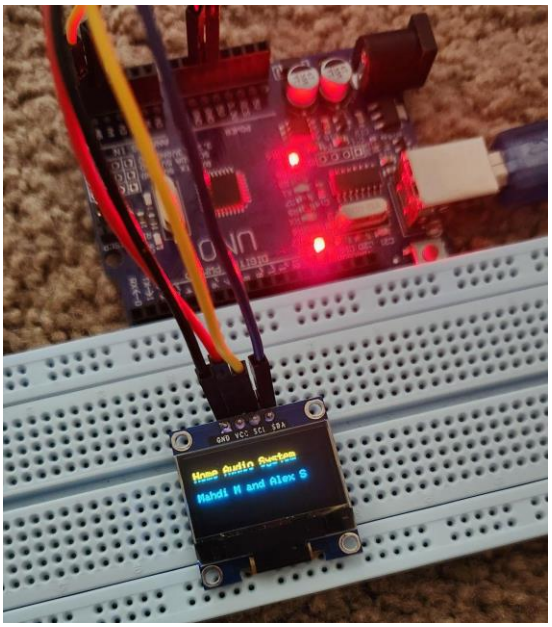


Figure 38) Final Spectrogram Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire);

const int barWidth = 42; // Adjust width as needed
const int ylim = 45;    // Max height so it doesn't go over
const int samplingInterval = 2000; // Update every 2 seconds
const int readDuration = 2000;  // Read voltage for 2 seconds
unsigned long lastUpdateTime = 0; // var to keep track when the value read needs to be updated
// Initial max value for each band until it's updated
int bassMax = 0;
int midMax = 0;
int trebleMax = 0;

int state = 0; // 0: Display home audio project, 1: Generate PWM signals
```

```
unsigned long startTime = 0; // Variable to store the start time

void drawBar(int x, int height) {
  // Draw a bar on the OLED display
  display.fillRect(x, SCREEN_HEIGHT - height, barWidth, height, SSD1306_WHITE);
}

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.display();
  startTime = millis();
}

void loop() {
  unsigned long currentTime = millis();

  if (state == 0) { // Time to swap from title screen to spectrogram
    if (currentTime - startTime >= 3000) {
      state = 1;
      lastUpdateTime = currentTime;  // Reset lastUpdateTime when transitioning to state 1
    } else {
      // Welcome screen
      display.clearDisplay();
      display.setCursor(0, 10);
      display.println("Home Audio System");
      display.setCursor(0, 30);
      display.println("Mahdi M and Alex S");
      display.display();
    }
  }

  if (state == 1) {
    if (currentTime - lastUpdateTime >= samplingInterval) {
      lastUpdateTime = currentTime;

      // Set initial values for voltage readings
      int startTime = currentTime;
      int endTime = startTime + readDuration;
      // Initial values to start until one is read
      int bassVal = 0;
      int midVal = 0;
      int trebleVal = 0;

      // Read voltage for duration inputted to get the peak value read
```

```
    while (millis() < endTime) {
      bassVal = max(bassVal, analogRead(A0));
      midVal = max(midVal, analogRead(A1));
      trebleVal = max(trebleVal, analogRead(A2));
    }

    display.clearDisplay();

    // Showing names for each corresponding bar
    display.setCursor(0, 0);
    display.print("Bass ");
    display.print("    Mid ");
    display.print("  Treble");

    // Display for the bass bar
    int bassHeight = map(bassVal, 0, 1023, 0, ylim);
    drawBar(0, bassHeight);

    // Display for the mid bar
    int midHeight = map(midVal, 0, 1023, 0, ylim);
    drawBar(barWidth + 1, midHeight);

    // Display for the treble bar
    int trebleHeight = map(trebleVal, 0, 1023, 0, ylim);
    drawBar(2 * (barWidth + 1), trebleHeight);

    // Updates the values when a new one is read
    bassMax = bassVal;
    midMax = midVal;
    trebleMax = trebleVal;
   tone(6, 35000); // Square Wave generation. Pin 6 is the output with a 35kHz generation
   delay(2000);    // Generates the frequency for 2 seconds

   }

   // Final command to display what's found
   display.display();

  }
}
```

Figure 39) Spectrogram Output Example. Where Treble > 0 Cursor Titles are blurred.

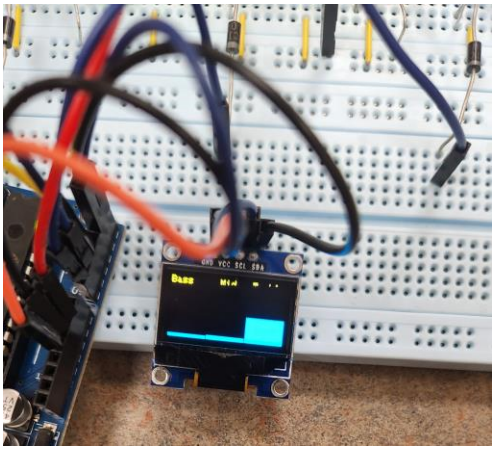


Figure 40. Simulation of Transistor shoot-through current, never exceeding 120 microamps

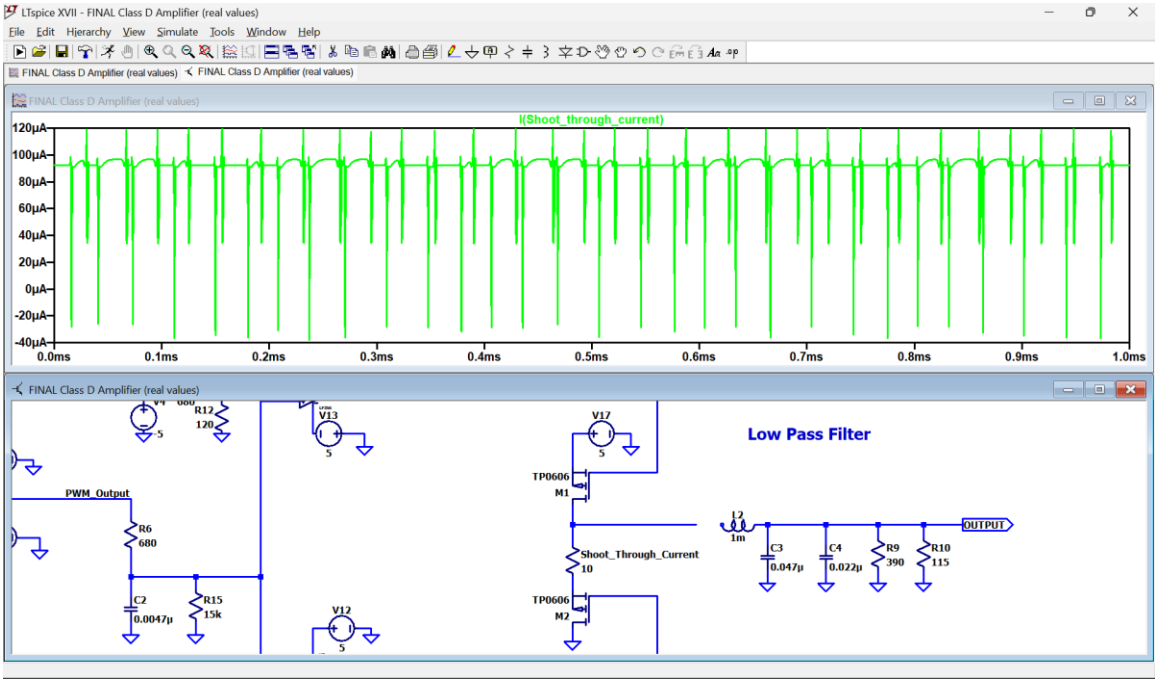


Figure 41. Measurement of Transistor shoot-through current. The red line is current, never exceeding -30 milliamps.

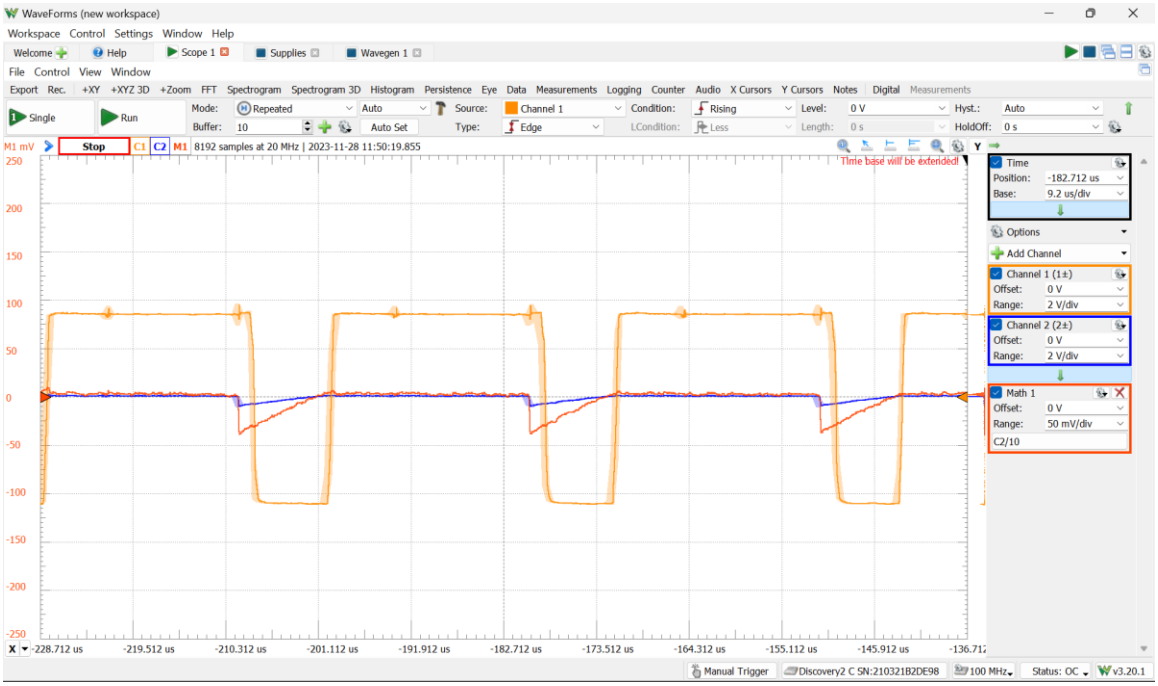


Figure 42. Simulation of the frequency response of the LC filter.

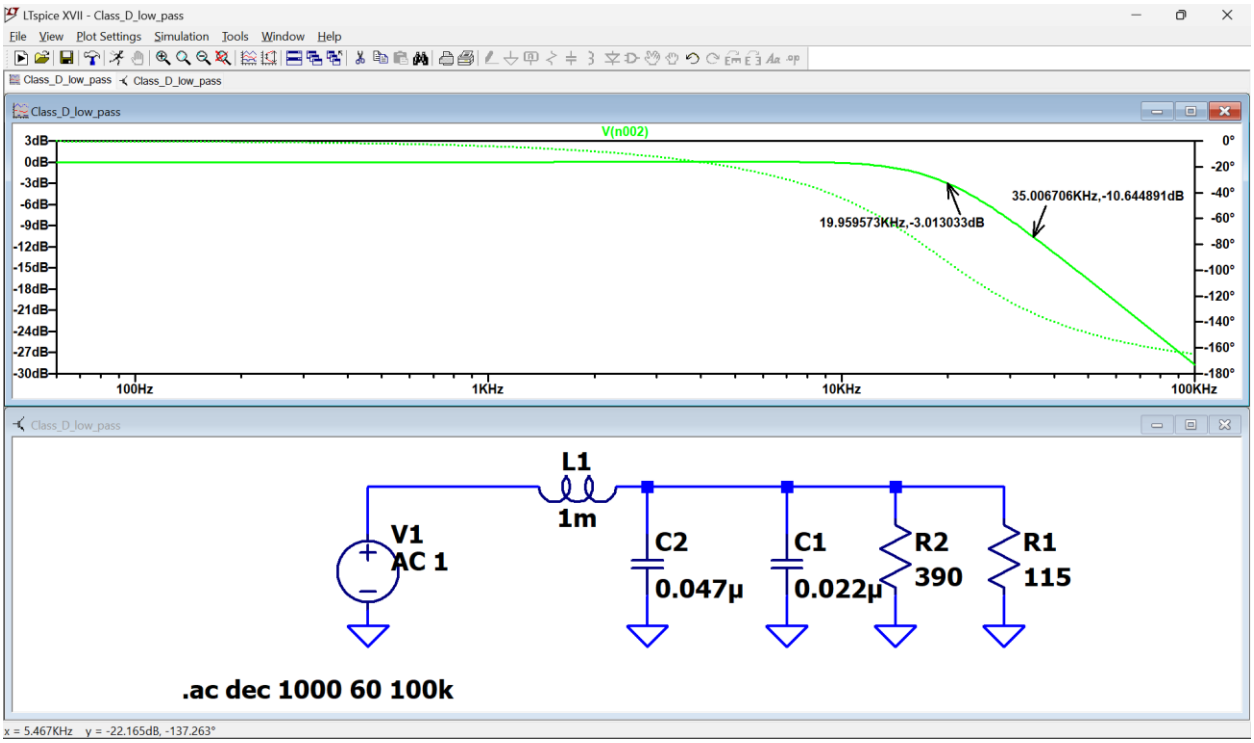


Figure 43. Simulation of the frequency response of the LC filter.

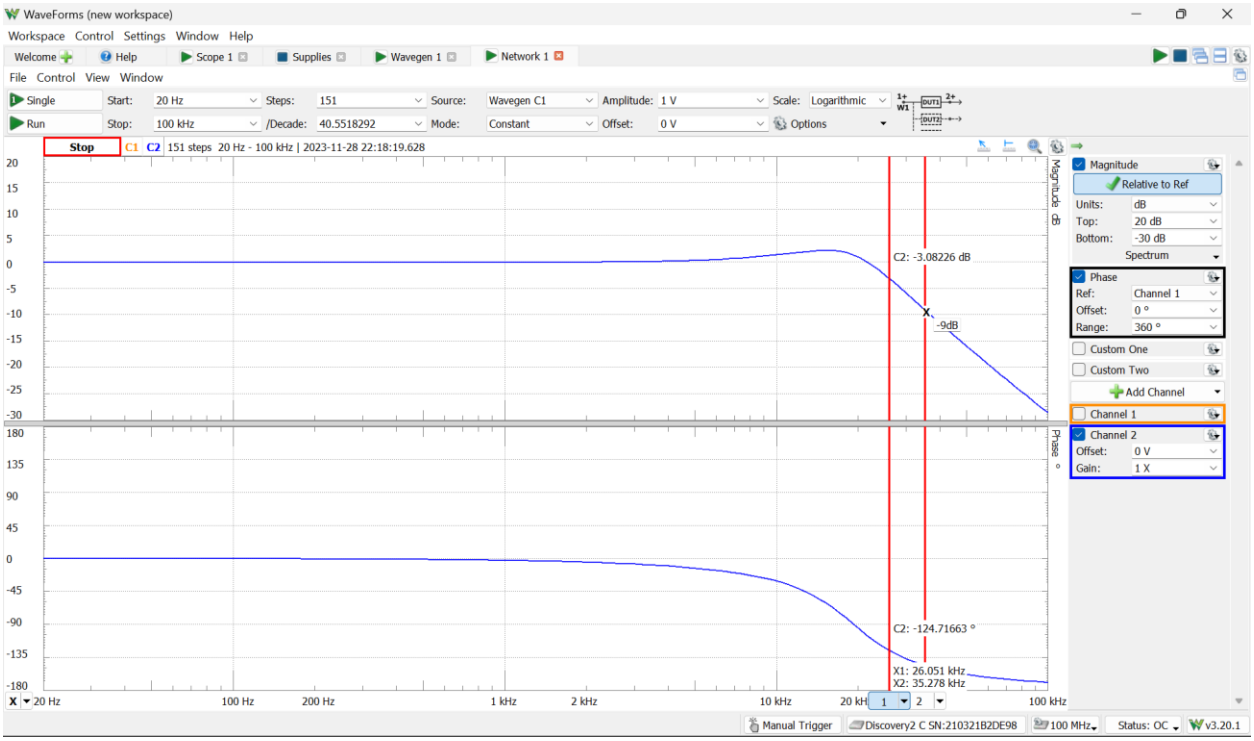


Figure 44. A song's audio wave viewed through the oscilloscope at the output of the graphic equalizer.

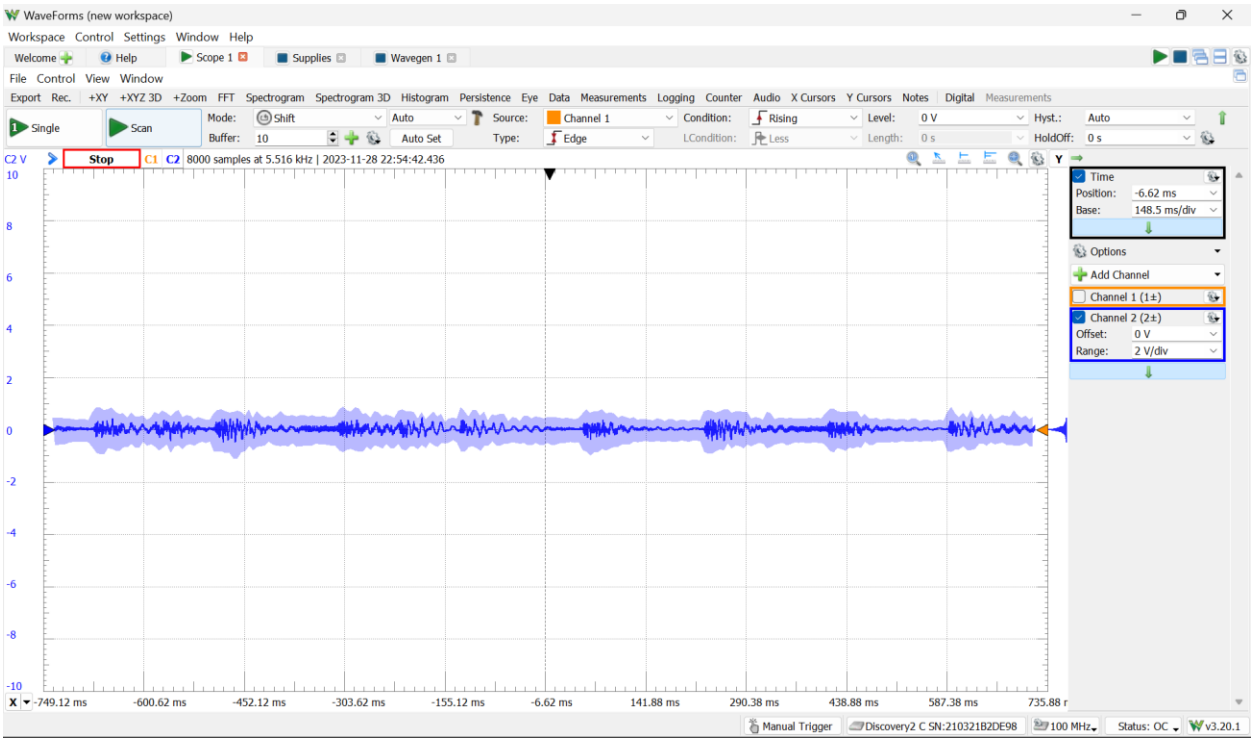


Figure 45. The audio wave is converted into voltage pulses, observe the short and long pulses.

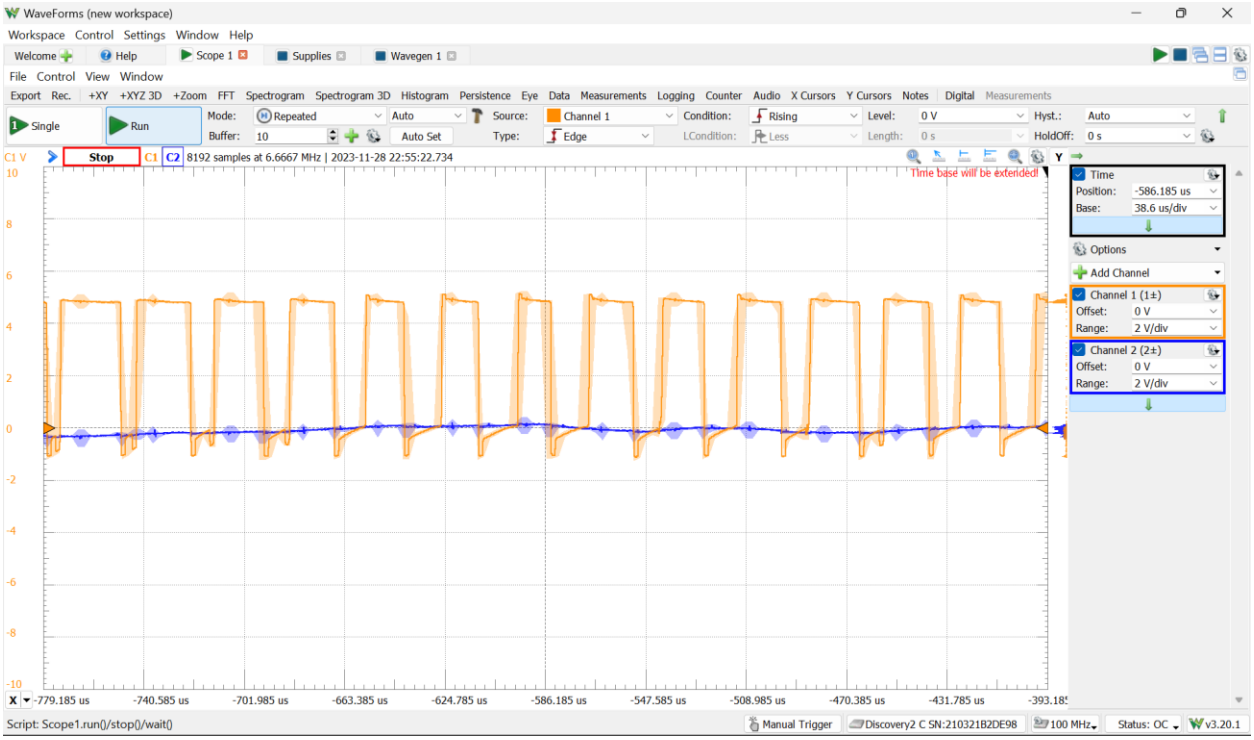
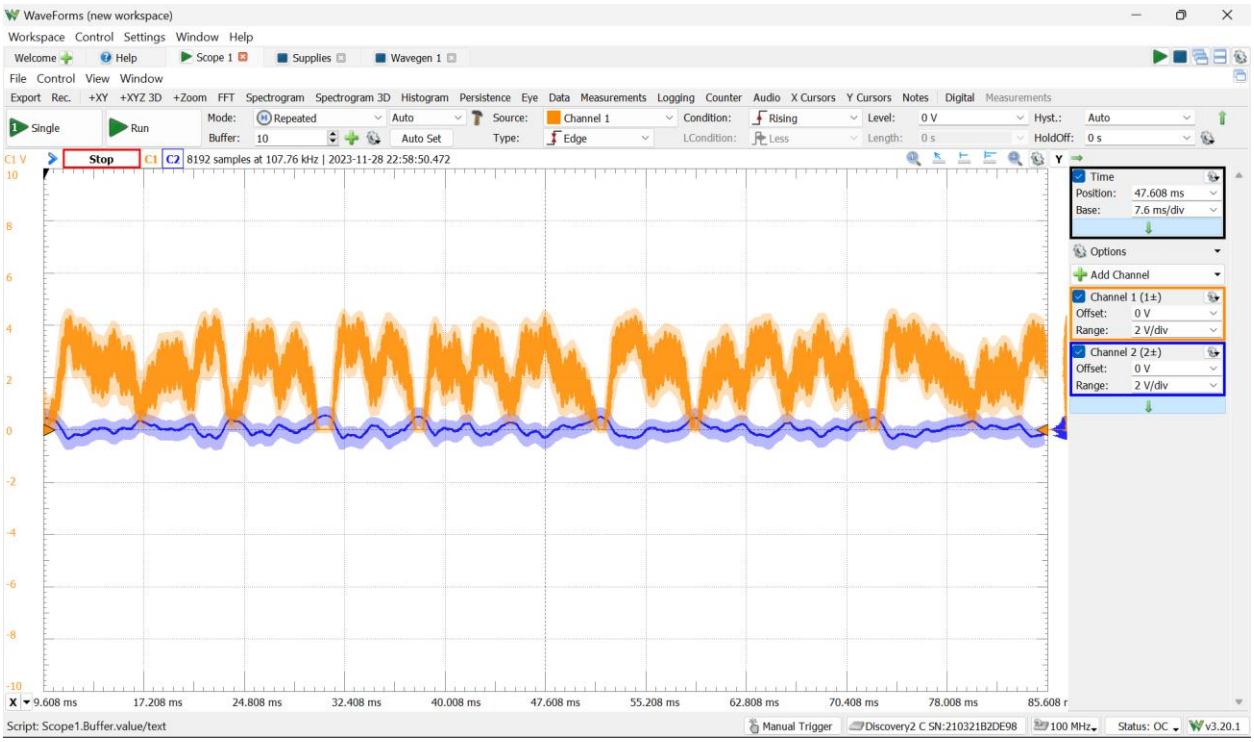


Figure 46. The output audio wave is compared to the input. The wave is greatly amplified.



Works Cited

[1] “Guide for I2C OLED Display with Arduino | Random Nerd Tutorials,” May 23, 2019. https://randomnerdtutorials.com/guide-for-oled-display-with-arduino (accessed Sep. 25, 2023).

[2] “Filter Design Tool | Filter Wizard,” *tools.analog.com.* https://tools.analog.com/en/filterwizard/

[3] Electrical4U, “Band Pass Filter: What is it? (Circuit, Design & Transfer Function) | Electrical4U,” *https://www.electrical4u.com/,* Apr. 16, 2021. https://www.electrical4u.com/band-pass-filter/

[4] “Audio spectrum,” Teach Me Audio, https://www.teachmeaudio.com/mixing/techniques/audio-spectrum (accessed Nov. 24, 2023).

[5] “tone() - Arduino Reference,” *www.arduino.cc*. https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

[6] “Triangular Wave Generator Using Op amp,” *EEEGUIDE.COM*, Sep. 14, 2016. https://www.eeeguide.com/triangular-wave-generator-using-op-amp/

[7] “Digital Level Shifter with Op-Amp,” *Electrical Engineering Stack Exchange*. https://electronics.stackexchange.com/questions/196409/digital-level-shifter-with-op-amp (accessed Nov. 03, 2023).

[8] “(Sample)RLC Low-pass Filter Design Tool - Result -,” *sim.okawa-denshi.jp*. http://sim.okawa-denshi.jp/en/RLCtool.php

[9] R. Keim, “Low-Pass Filter a PWM Signal into an Analog Voltage - Technical Articles,” *www.allaboutcircuits.com*, Apr. 11, 2016. https://www.allaboutcircuits.com/technical-articles/low-pass-filter-a-pwm-signal-into-an-analog-voltage/

[10] Janux, “JX Audio Spectrometer,” Hackster.io, https://www.hackster.io/janux/jx-audio-spectrometer-33a5c5 (accessed Nov. 6, 2023).

[11] EETimes, “Understanding output filters for class D amplifiers,” EE Times, https://www.eetimes.com/understanding-output-filters-for-class-d-amplifiers/ (accessed Nov. 6, 2023).

[12]“Map(),” map() - Arduino Reference, https://www.arduino.cc/reference/en/language/functions/math/map/ (accessed Nov. 14, 2023).

[13] “Millis(),” millis() - Arduino Reference, https://www.arduino.cc/reference/en/language/functions/time/millis/ (accessed Nov. 14, 2023).

[14] J. Caldwell, “Analog pulse width modulation U1B VREF - Texas Instruments India,” Analog Pulse Width Modulation, https://www.ti.com/lit/ug/slau508/slau508.pdf (accessed Nov. 26, 2023).

[15] Krakkus, “How to add dead-time to PWM,” YouTube, https://www.youtube.com/watch?v=avoAbZekMME&t=80s (accessed Nov. 25, 2023).