Final Report

By Charlotte Pei & Joaquin Tuckett

1. Introduction

The goal of this project was to design, build, test, and validate a functional home audio system that accepts an audio input, modifies the signal through a three-band graphic equalizer, amplifies the signal using a Class-D amplifier, and displays the strength of each frequency band on an OLED spectrogram. The required system contains three main subsystems: a graphic equalizer, a Class-D amplifier, and an Arduino-controlled OLED display. The graphic equalizer controls bass, mid-range, and treble frequency bands, while the Class-D amplifier uses PWM switching to drive the output efficiently. The OLED display provides a welcome page and a three-band spectrogram that visually represents the amplitude of each equalizer band.

The minimum project requirements were to cover the 100 Hz to 8 kHz audio range with three band-pass filters, allow each frequency band to be boosted or cut using potentiometers, generate a PWM carrier above 80 kHz for the Class-D amplifier, avoid shoot-through current in the switching stage using dead time, filter the switching output with an RLC low-pass filter, and display smoothly changing bass, mid-range, and treble bars on the OLED. The project description also required the final system to be tested as an integrated prototype using an audio signal and headphones. Beyond the minimum requirements, our team also explored PCB design and system packaging improvements to reduce wiring noise, improve performance, and make the prototype easier to assemble and debug. These improvements were especially useful because the breadboard version of the system was sensitive to parasitic effects, impedance introduced by long wires, loose connections, and power-supply noise.

2. High Level Design

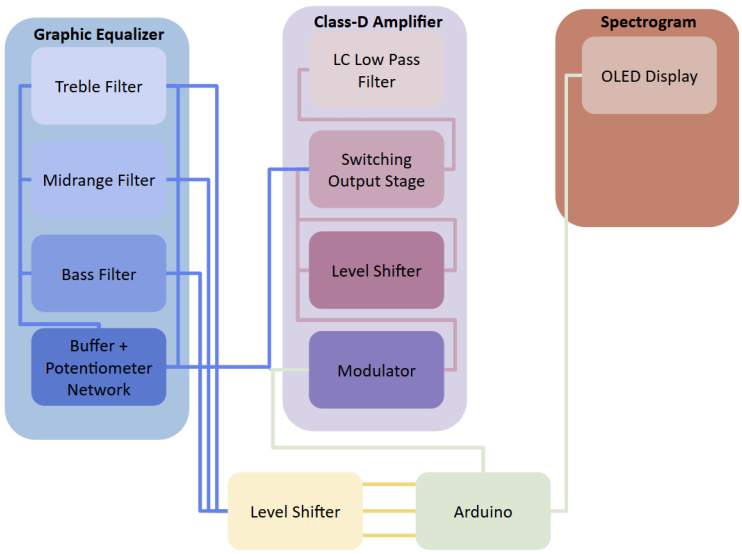

Figure 2.1: *The block diagram of the Home Audio System*

Figure 2.1 shows the high-level block diagram of the Home Audio System. Our design consists of three main structures: the graphic equalizer, the Class-D amplifier, and the spectrogram. When the audio signal enters the system, it will first pass through the graphic equalizer, entering a potentiometer

network with three distinct bandpass filters. Each of these filters is used to isolate and amplify or attenuate distinct frequency bands in the audio signal. Through the use of the graphic equalizer, we may choose to amplify or attenuate low frequencies, midrange frequencies, high frequencies, or any combination of the three. The output of each filter is passed to a level shifter, which ensures the audio signal is within an acceptable voltage for the Arduino, i.e. it doesn't contain negative voltages. The arduino then analyzes these signals and uses them to update an OLED display, allowing us to visualize the amplification and attenuation of the different frequency bands. The output of the entire graphic equalizer is passed to the Class-D amplifier. The Class-D amplifier is unique from other amplifier designs, such as the class-A and class-B amplifiers, on account of its use of switching MOSFETs to minimize power consumption. The Arduino passes a high frequency square wave to the modulator, which integrates it to create a triangle wave. This triangle wave is then level shifted to create a new triangle wave. Both waves then pass through separate comparators and are compared to the graphic equalizer's output, creating new square waves that are used as gate voltages for a CMOS pair. This causes the drain of the pair of transistors to switch back and forth between outputting 5V and -5V, effectively amplifying the input square wave signal. The original triangle wave was shifted to create "deadtime," a period of time where neither the PMOS nor NMOS transistor in the pair is turned on, ensuring that we don't have current spikes from connecting the 5V supply to the -5V supply. A shifted triangle wave was used in place of a time-delay circuit due to its simpler circuitry, as it only required a level shifter and comparator network instead of the precise configuration of two RC and diode networks, which would have been limited by the components available in our kit. The output of this switching stage is then passed to the LC low pass filter, which attenuates the high frequencies from the amplified square wave and returns an amplified version of the graphic equalizer's output.

3. Detailed Design & Verification

Figure 3.1 shows the full system schematic used to connect the graphic equalizer, Class-D amplifier, and OLED spectrogram interface. The graphic equalizer shapes the frequency content of the input audio signal, the Class-D amplifier converts the equalizer output into a PWM switching signal and reconstructs it through a low-pass filter, and the Arduino/OLED subsystem measures the strength of each equalizer band and displays it as a three-band spectrogram.

Each subsystem was designed analytically, simulated in LTspice, built on hardware, and verified using WaveForms/AD2 measurements. The detailed subsystem schematics are included in Appendix A so that the main report can focus on the design choices and verification results. The full graphic equalizer schematic is shown in Figure A.1, the Class-D modulator schematic is shown in Figure A.2, the triangle level-shifter/dead-time circuit is shown in Figure A.3, the switching stage is shown in Figure A.4, and the RLC low-pass filter is shown in Figure A.5. The hand calculations used to select the filter bandwidths, component values, voltage-divider values, Class-D modulator values, and RLC low-pass filter values are shown in Figures A.6 through A.10.

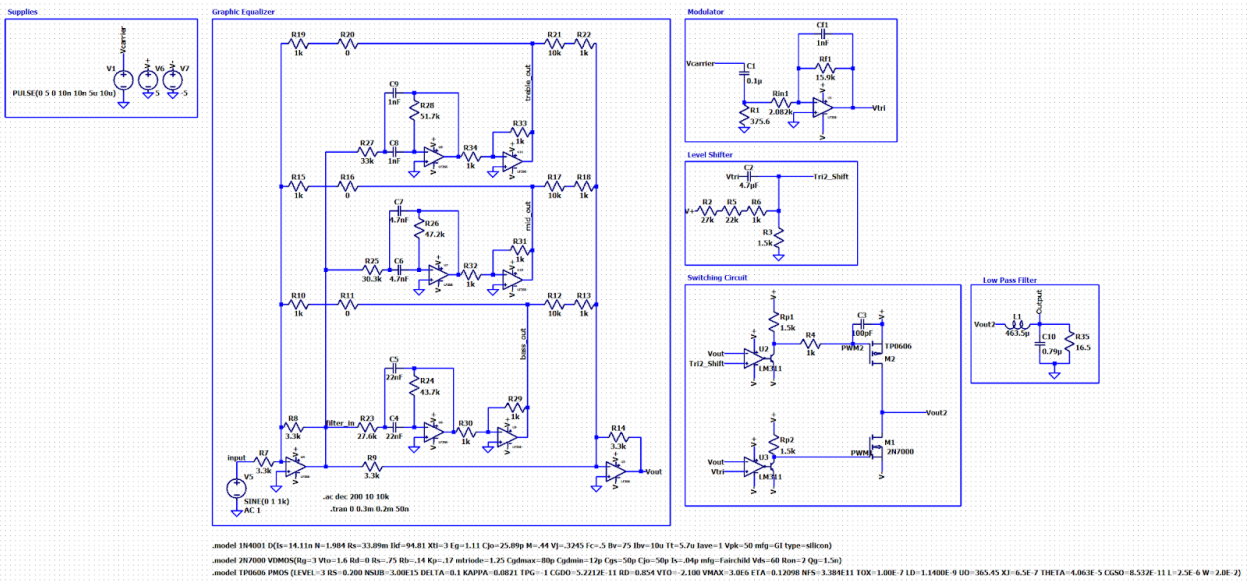

***Figure 3.1**: Full LTspice schematic of the integrated home-audio system, including the graphic equalizer, Class-D modulator, switching output stage, and RLC low-pass filter.*

a) Graphic Equalizer

The graphic equalizer was designed to divide the input audio signal into three adjustable frequency bands: bass, mid-range, and treble. Since the required audio range was approximately 100 Hz to 8 kHz, the band-pass filters were chosen using logarithmic spacing rather than equal linear spacing. This was selected because audio perception is frequency-ratio based, so logarithmic spacing creates a more natural division of the audio spectrum. As shown in Figure A.6, this produced final band ranges of approximately 100 Hz to 431.6 Hz for bass, 430 Hz to 1864.8 Hz for mid-range, and 1853.8 Hz to 8010.3 Hz for treble. These bands provide continuous coverage of the required range with slight overlap near the crossover points, which helps prevent gaps in the equalizer response.

Each band used a multiple-feedback active band-pass filter because this structure allowed the center frequency and Q value to be set using practical resistor and capacitor values from the lab kit. The component-value calculations are shown in Figure A.7, and the full graphic equalizer schematic is shown in Figure A.1. The calculated values were adjusted to available parts while keeping the center frequencies and Q values close to their targets, allowing the filters to be realistic to build while still meeting the intended frequency coverage.

The gain-control portion of the equalizer uses a potentiometer network around the band-pass filters so each band can be boosted, cut, or held near neutral. The transfer function in Equation A.1 shows that the gain depends on the band-pass filter response and the boost/cut resistance values. When the potentiometer is centered, the boost and cut paths balance to produce an approximately 0 dB gain. Adjusting the potentiometer changes the resistance balance, allowing each band to be independently amplified or attenuated.

The simulated band-pass filter responses are shown in Figure 3.2, where the bass, mid-range, and treble filters peak near their intended center frequencies and overlap near the crossover points. The measured responses in Figures 3.3 through 3.5 confirm that the physical filters followed the same overall behavior

as the simulations. Small differences between simulation and measurement were expected due to component tolerances, op-amp nonidealities, breadboard parasitics, and measurement noise.

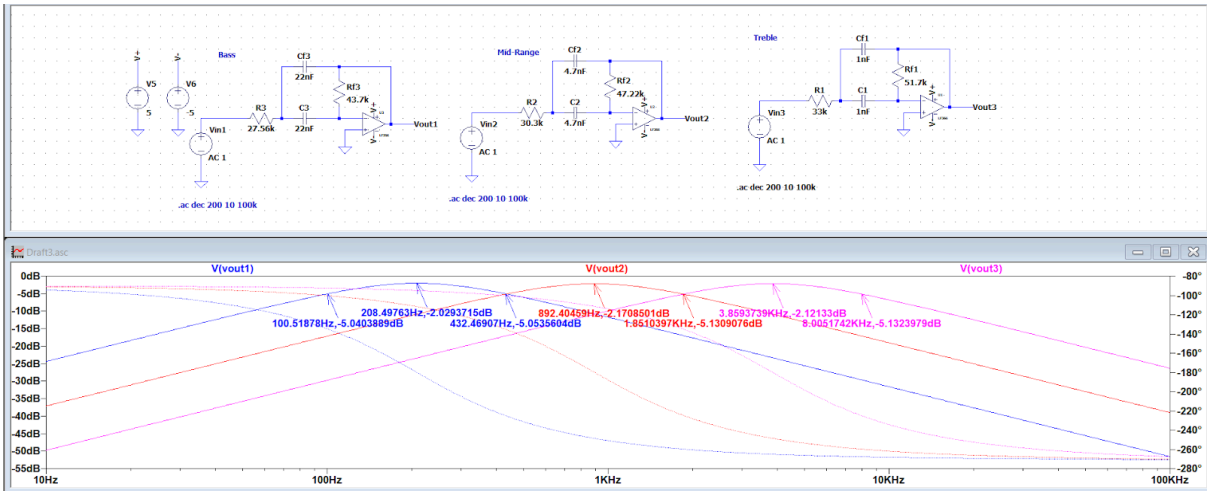

Figure 3.2: *Simulated LTspice magnitude responses of the bass, mid-range, and treble BPF to show crossover behavior.*

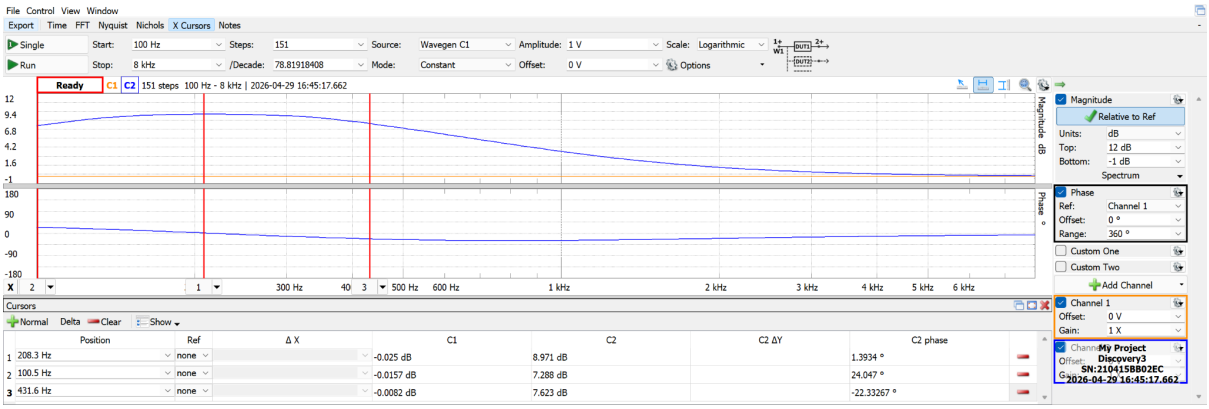

Figure 3.3: *Measured graphic equalizer frequency response of the bass band-pass filter with the bass band boosted and the mid-range and treble bands neutral.*

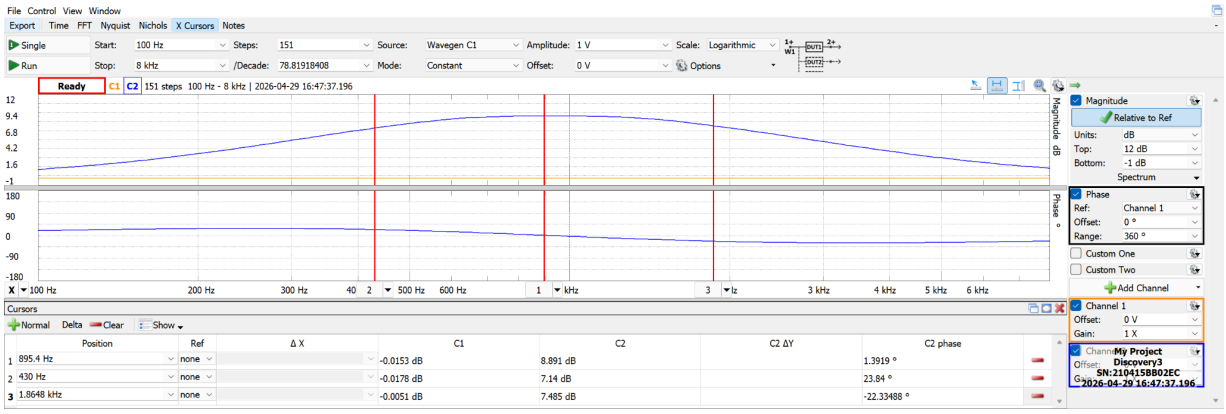

Figure 3.4: *Measured graphic equalizer frequency response of the mid-range band-pass filter with the mid-range band boosted and the bass and treble bands neutral.*

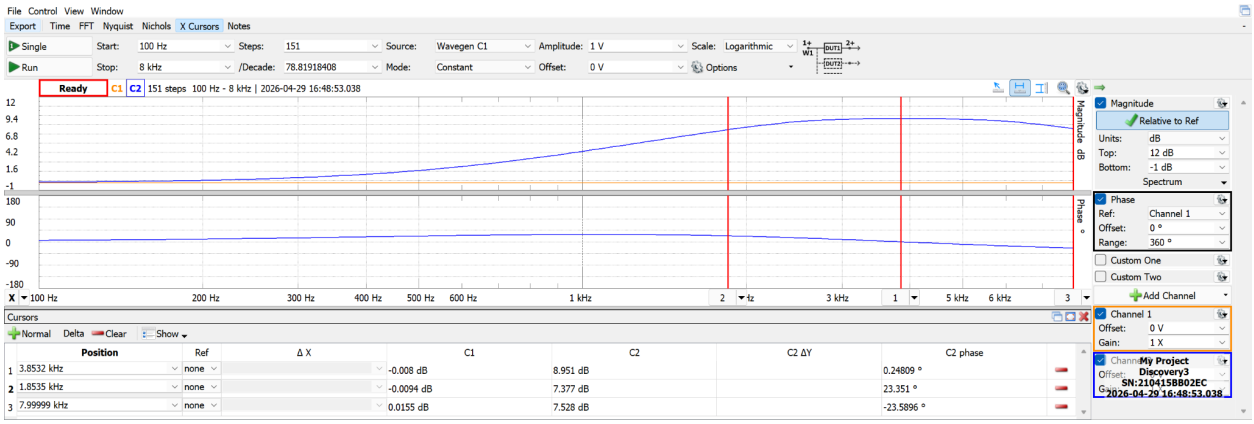

Figure 3.5: *Measured graphic equalizer frequency response of the treble band-pass filter with the treble band boosted and the mid-range and bass bands neutral.*

After verifying the individual filters, the full equalizer was tested in neutral, boost, and cut conditions. Figure 3.6 shows the simulated response when all potentiometers are in the neutral position. The response stays close to 0 dB over the audio band, which confirms that the equalizer does not significantly boost or attenuate the signal when the user controls are centered. Figure 3.7 shows the measured neutral response of the physical equalizer. The measured response remains very close to 0 dB, which is a strong result because it confirms that the real circuit preserves the audio signal when no tonal adjustment is desired.

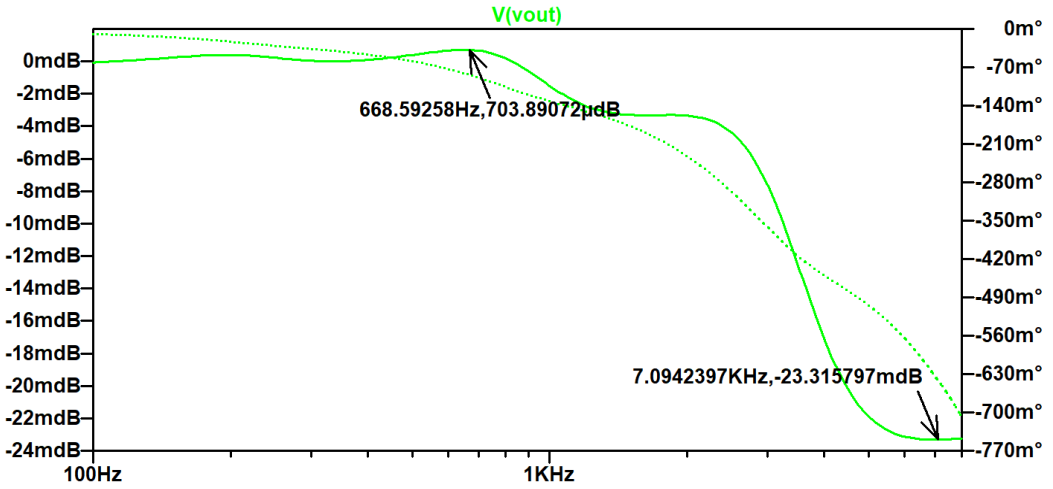

Figure 3.6: *Simulated graphic equalizer frequency response with all potentiometers in the neutral position.*

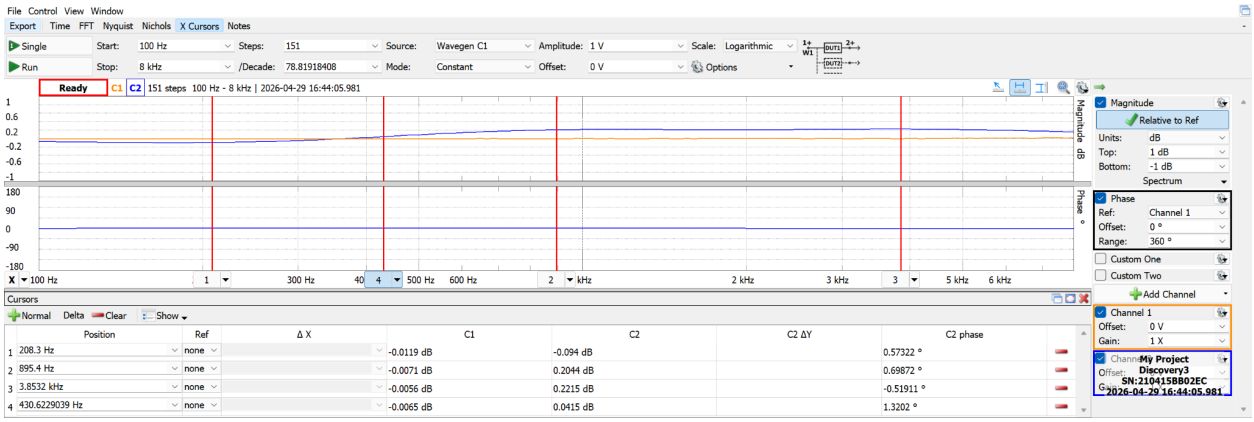

***Figure 3.7**: Measured graphic equalizer frequency response with all potentiometers in the neutral position.*

Figures 3.8 and 3.10 show the simulated frequency responses of the all-boost and all-cut conditions, while Figures 3.9 and 3.11 show the measurements for the all-boost and all-cut conditions. In the measured all-boost condition, the measured ripple was about 0.297dB, staying below the 4 dB project requirement. None of the frequency bands peaked over 12 dB. This is an important result because it shows that the three boosted bands combine smoothly across the full audio range rather than creating large peaks or dips. The all-cut response confirms that the circuit can attenuate all three frequency bands. These results demonstrate that the potentiometer network successfully controls each band independently.

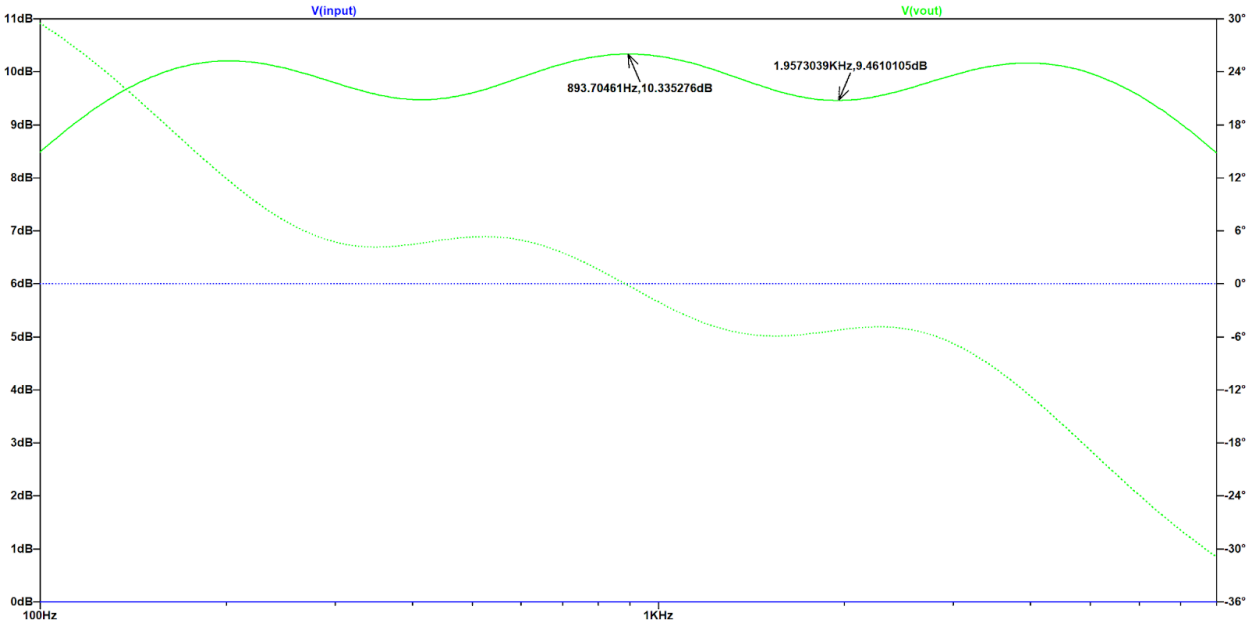

***Figure 3.8**: Simulated frequency response of the graphic equalizer when Cuts are 10k and Boosts are 0 Ohms.*

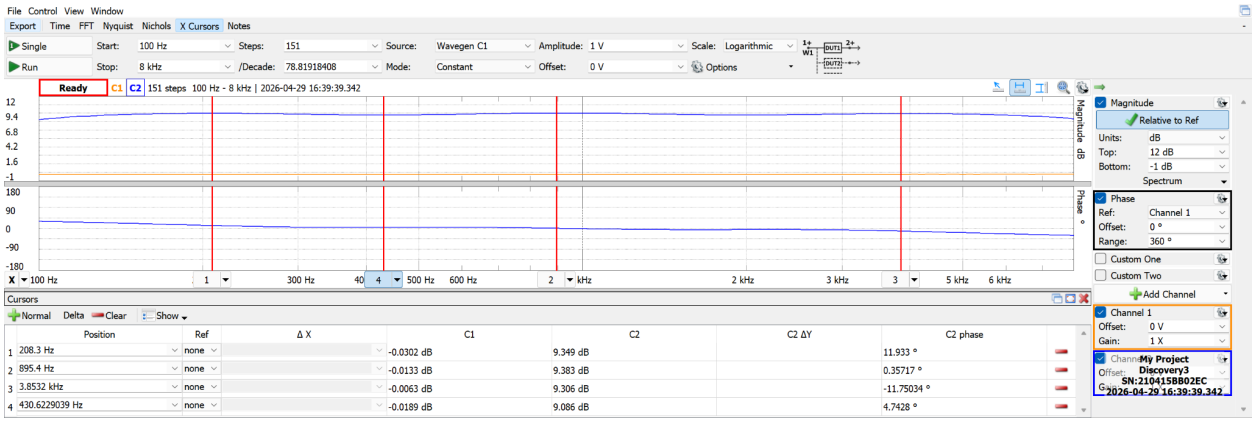

***Figure 3.9**: Measured graphic equalizer frequency response with all bands boosted*

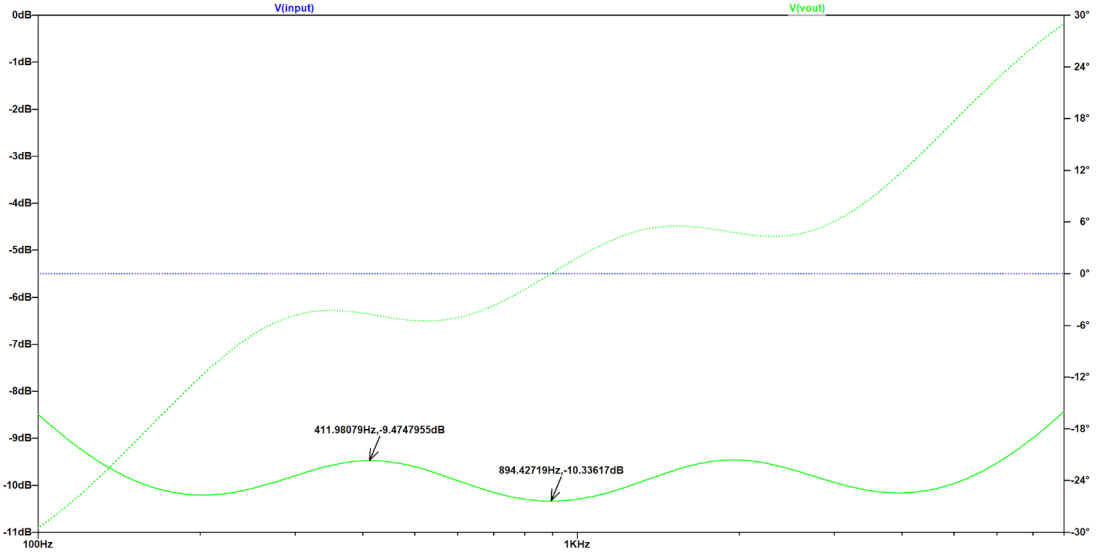

***Figure 3.10**: Simulated frequency response of the graphic equalizer when Cuts are 0 Ohms and Boosts are 10k.*

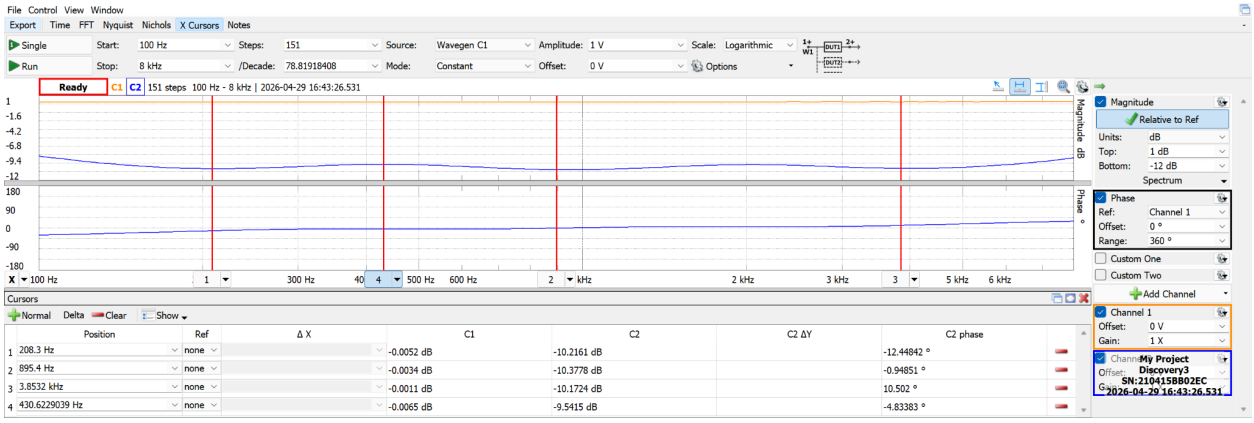

***Figure 3.11**: Measured graphic equalizer frequency response with all bands cut*

b) Class-D Amplifier

The Class-D amplifier was designed to efficiently amplify the equalizer output by converting the analog audio signal into PWM, switching the output between the supply rails, and then filtering the switching

waveform back into an analog signal. This architecture was chosen because Class-D amplifiers use switching transistors instead of linear output devices, which reduces power loss and improves efficiency when driving a load.

The first stage is the modulator. The project required a 1 V, 1 kHz sinusoidal input and a carrier frequency above 80 kHz, so the Arduino Uno was configured to generate a 100 kHz square wave using Timer/Counter1 in Fast PWM mode 14 with no prescaling and ICR1 = 159. This frequency was safely above the minimum requirement and high enough for the output low-pass filter to attenuate the switching component. The modulator design calculations are shown in Figure A.9, and the LTspice modulator schematic is shown in Figure A.2.

The Arduino square wave was converted into a triangle carrier using an op-amp integrator. A high-pass filter was placed before the integrator to remove the Arduino signal's 2.5 V DC offset, preventing integrator drift and op-amp saturation. The integrator component values were selected so the 100 kHz square wave produced a triangle carrier large enough to compare with the 1 kHz audio input. This triangle waveform was then compared with the sine wave using an LM311 comparator, producing a PWM signal whose duty cycle followed the audio amplitude.

The simulated modulator output is shown in Figure 3.12, and the measured PWM output is shown in Figure 3.13. The measured PWM output had a frequency of approximately 100 kHz, which meets the requirement that the carrier frequency be above 80 kHz. This is a strong result because the measured hardware output confirms that the Arduino timing configuration, integrator, and comparator worked together as intended.

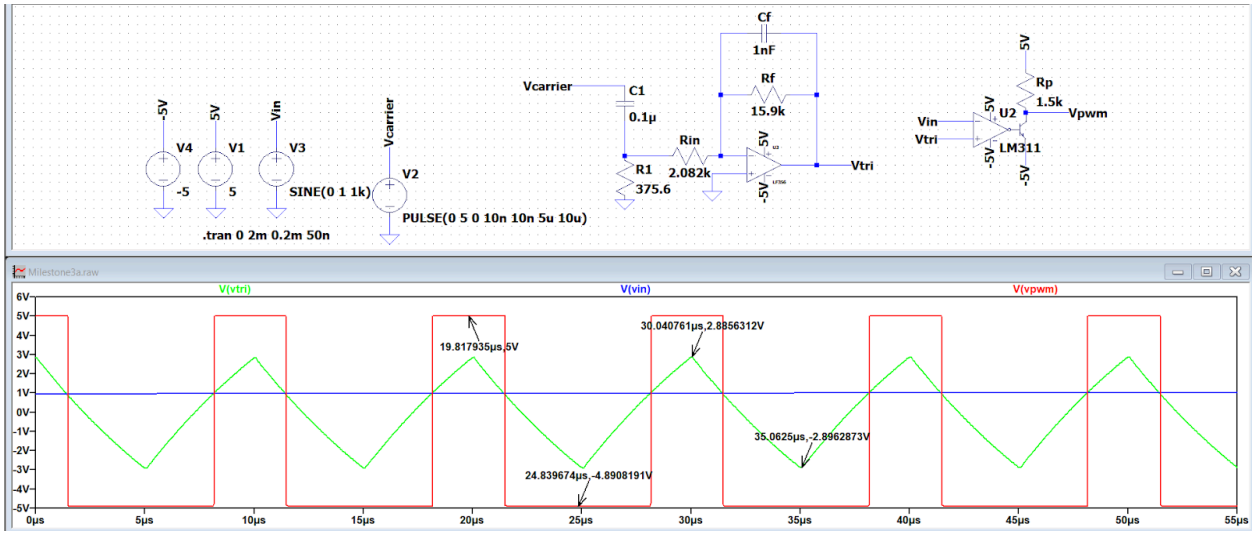

Figure 3.12: *LTspice transient simulation of the Class-D modulator, showing the 1 kHz sinusoidal input, triangle carrier, and resulting PWM waveform.*

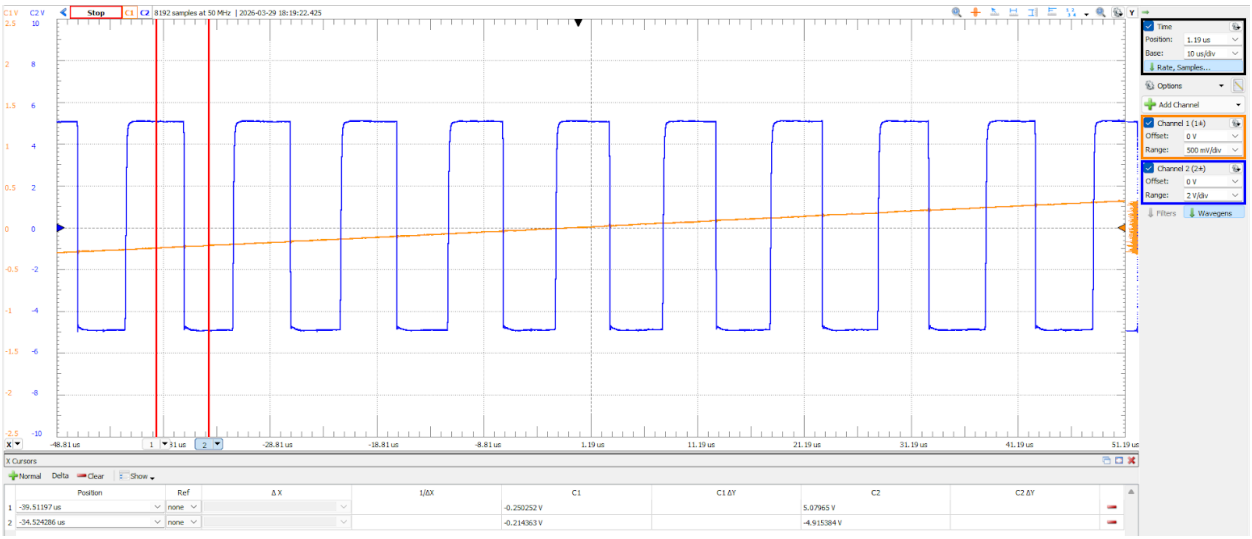

Figure 3.13: *Measured PWM output from the breadboard Class-D modulator*

The switching output stage shown in Figure A.4 used the PWM signal to drive a PMOS/NMOS transistor pair so the output switched between approximately +5 V and −5 V as shown in Figure 3.15. A PMOS/NMOS pair was chosen because it allows the output node to be actively driven toward both supply rails, creating a larger voltage swing than the comparator alone could provide. However, this stage also creates the risk of shoot-through current if the PMOS and NMOS are on at the same time. Shoot-through would briefly short the positive and negative supply rails together, causing large current spikes, distortion, and possible transistor damage.

To prevent this issue, a shifted version of the triangle waveform was generated using the level-shifter circuit shown in Figure A.3. By comparing the input audio signal with two slightly different triangle waveforms, the circuit produced two switching-control signals with intentional non-overlap. This non-overlap is the dead time. The switching-stage output is shown in Figure 3.14, and the switching-control waveforms are shown in Figure 3.15. Figure 3.16 shows the measured dead time between PMOS and NMOS switching transitions. The presence of dead time confirms that the transistors were not turned on at the same time, which protects the circuit from shoot-through current.

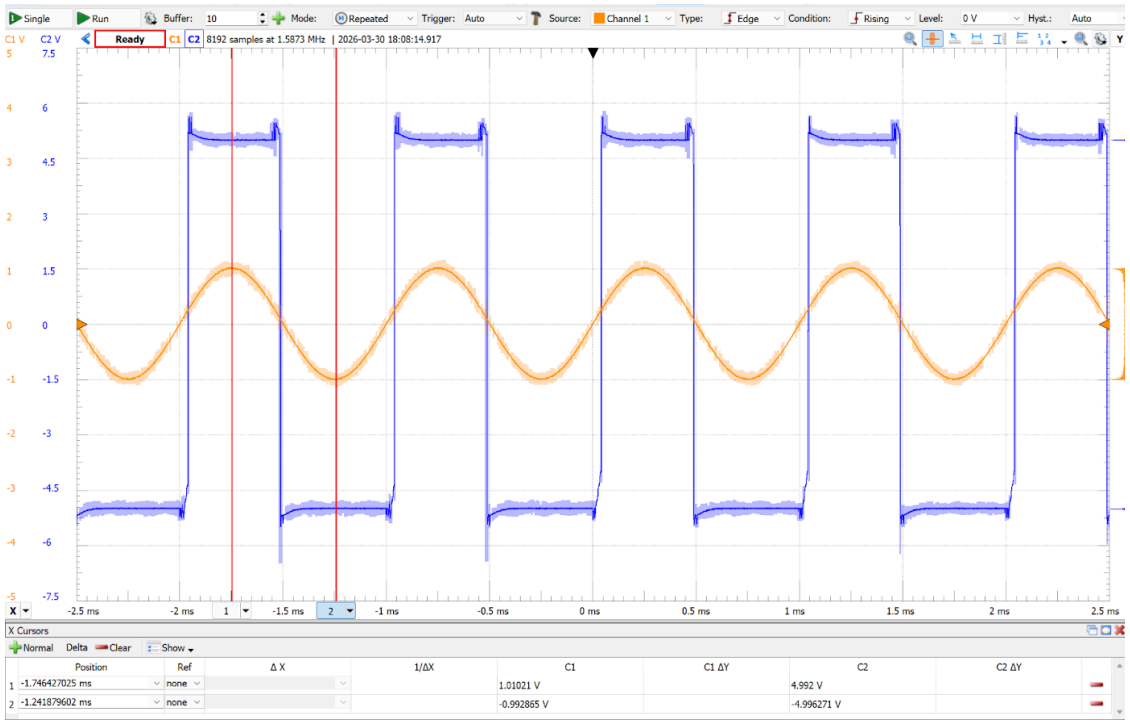

Figure 3.14: *Measured output of the switching stage (blue) and the input sine wave (orange).*

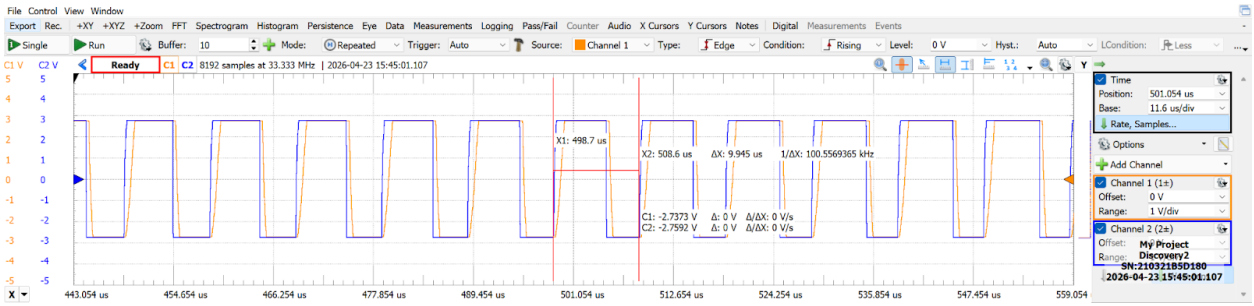

Figure 3.15*: Measured PWM gate signals*

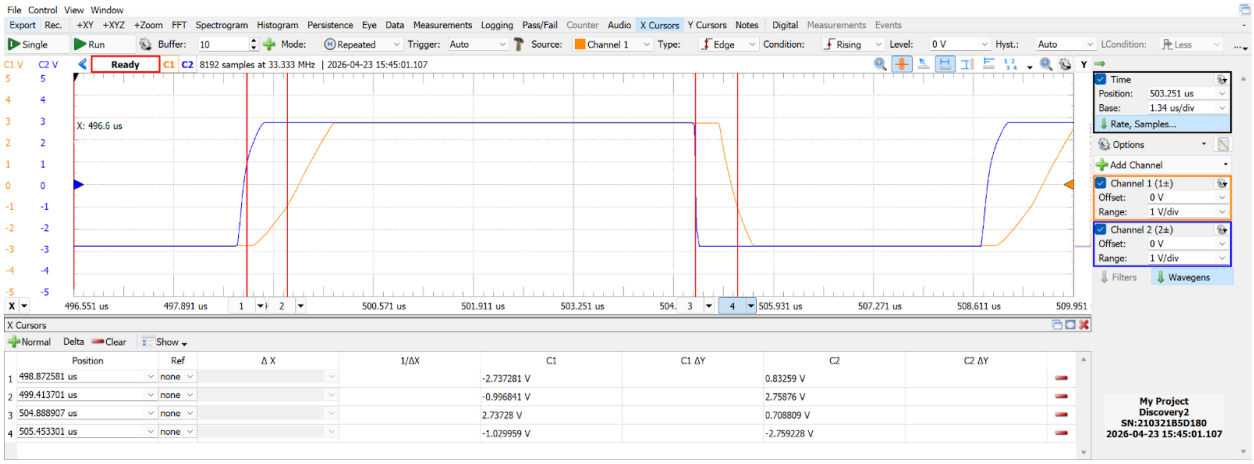

Figure 3.16*: Measured dead time between PMOS and NMOS switching transitions*

The final stage of the Class-D amplifier was the RLC low-pass filter. This filter was required because the switching-stage output contains the desired audio information plus high-frequency PWM carrier

components. A second-order Butterworth low-pass filter was chosen because it provides a flat passband while attenuating the high-frequency carrier. The design used an effective load resistance of 16.5 Ω, corresponding to two 33 Ω headphone channels in parallel. The RLC hand calculations are shown in Figure A.10, and the LTspice low-pass filter schematic is shown in Figure A.5. The calculated ideal values were adjusted to match available lab components, and the final design used the measured value of the hand-wound inductor. The simulated frequency response of the RLC low-pass filter is shown in Figure 3.17, and the measured frequency response is shown in Figure 3.18. The measured cutoff frequency was approximately 9.035 kHz, which is close to the intended design range and well below the 100 kHz switching frequency. This verifies that the filter passes the desired audio-band signal while attenuating the PWM carrier.

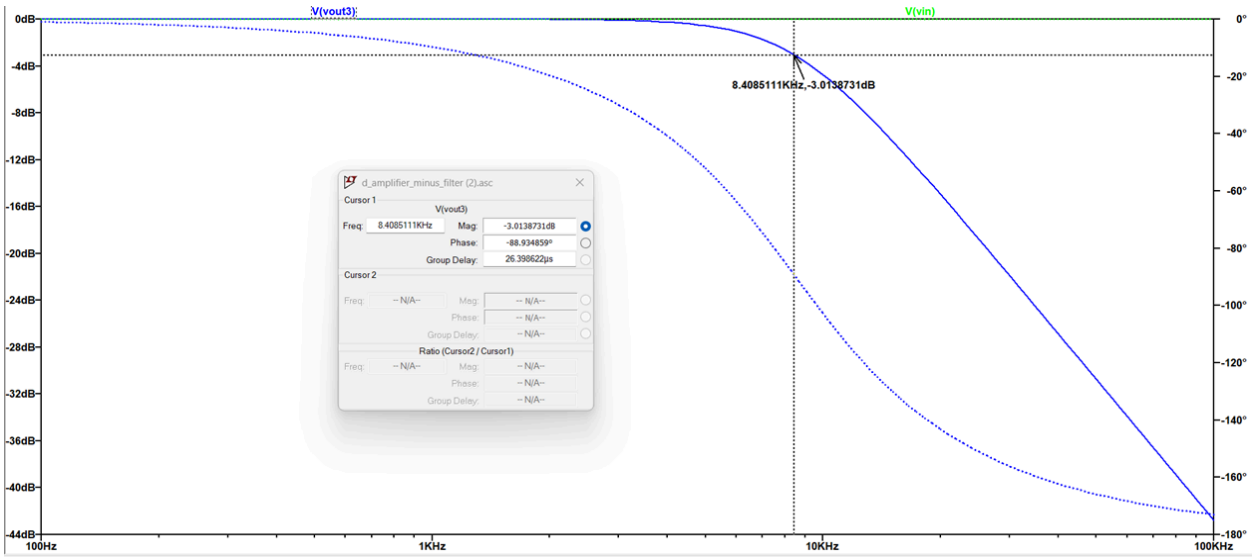

Figure 3.17: *Simulated LTspice frequency response simulation of the RLC low-pass filter.*

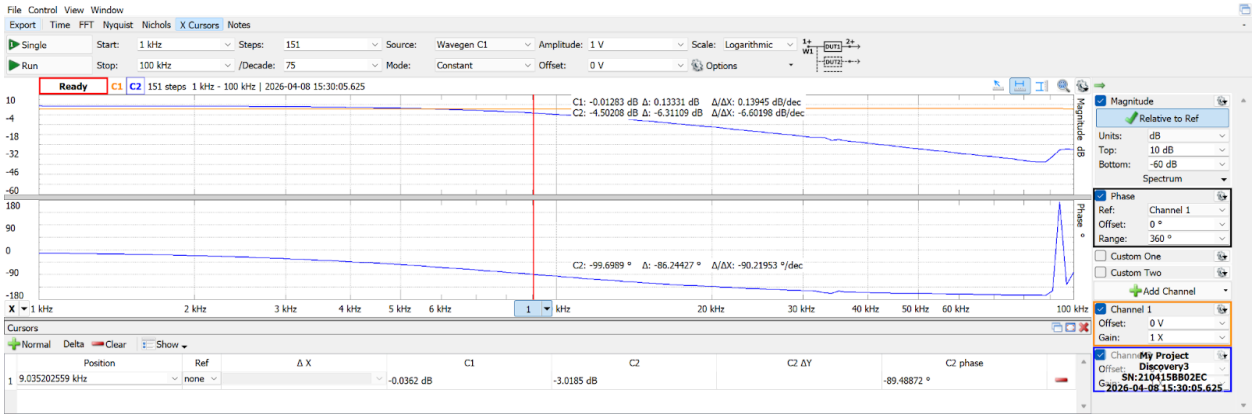

Figure 3.18: *Measured frequency response of the RLC low-pass filter.*

The complete Class-D amplifier was then tested in the time domain. Figure 3.19 shows the filtered Class-D amplifier output compared with the 1 kHz input waveform. The filtered output follows the shape of the input sine wave closely, confirming that the low-pass filter successfully reconstructs the analog audio signal from the switching waveform. This is one of the most important validation results because it shows that the modulator, switching stage, and low-pass filter work together as a complete amplifier.

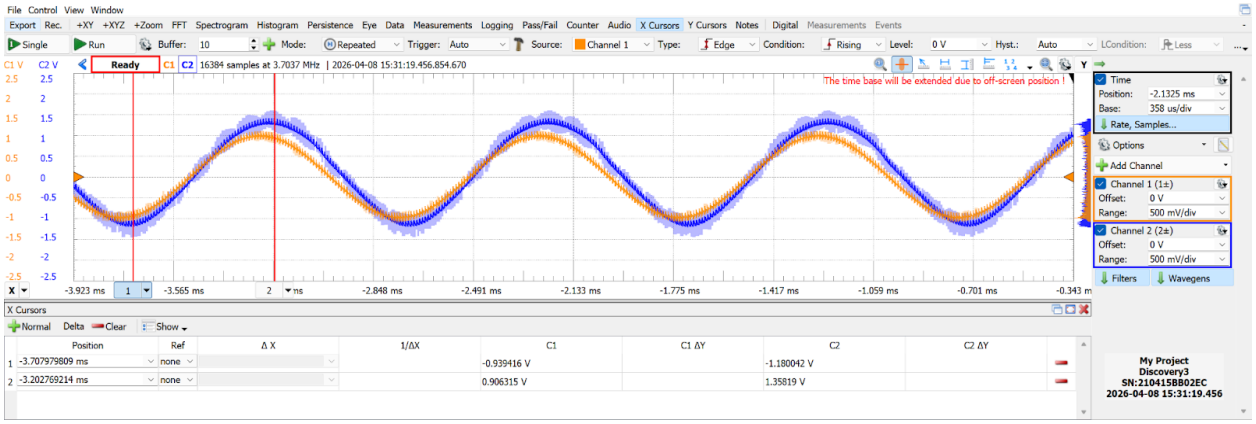

Figure 3.19: *Measured Class-D Amplifier output (blue) compared with the 1 kHz input waveform (orange) in the time-domain.*

Ripple voltage and DC bias were tested by grounding the Class-D amplifier input and measuring the filtered output. This test is important because it shows whether the Class-D amplifier produces unwanted output when no input signal is applied. As shown in Figure 3.20, the measured ripple was 81.4 mV peak-to-peak, which is below the 100 mV requirement, and the measured DC bias was 145.2 mV, which is below the 0.4 V requirement. These results are strong because both specifications were met with margin. The small difference between simulation and measurement is reasonable because the physical circuit includes breadboard parasitics, component tolerances, transistor nonidealities, and power-supply noise.

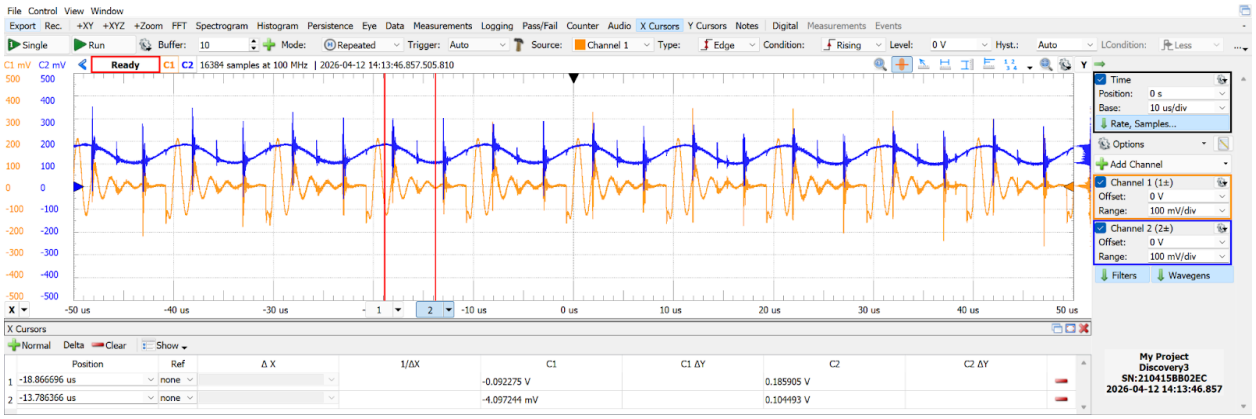

Figure 3.20: *Measured ripple voltage and DC bias of the low-pass filter output (blue) with the Class-D amplifier input (orange) grounded.*

Finally, the input and output spectra were measured using the WaveForms spectrum tool, as shown in Figure 3.21. The measured input THDp was 0.3044%, and the measured output THDp was 4.308%. These are strong results because the output distortion remained low even after PWM modulation, transistor switching, and low-pass reconstruction. A THDp of 4.308% indicates that the Class-D amplifier preserved the main audio waveform with relatively little distortion compared with the amount of switching activity present in the circuit. The low input THDp also confirms that the original test signal was clean, so most of the measured output distortion came from the amplifier hardware rather than the signal source.

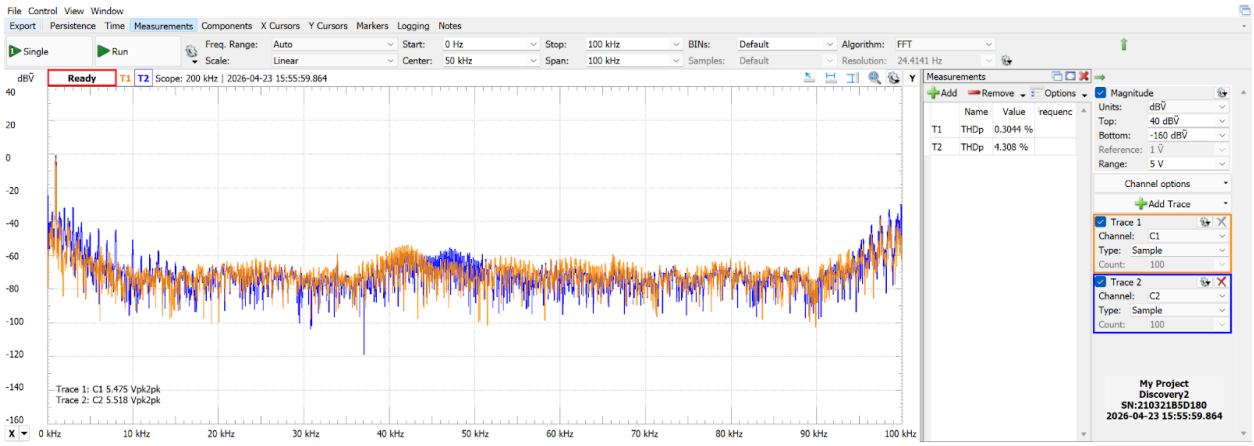

***Figure 3.21**: Measured total harmonic distortion percentage (THDp) of the input (orange) and output (blue) waveforms.*

c) OLED Spectrogram

The OLED spectrogram was designed to display the signal strength of the bass, mid-range, and treble bands in real time. This subsystem used an Arduino Uno and an SSD1306 OLED display. The OLED was controlled through I²C communication, while the three equalizer band outputs were connected to Arduino analog inputs so the microcontroller could estimate the amplitude of each band. Because the OLED logic pins operate at 3.3 V while the Arduino uses 5 V logic, voltage-divider protection was added to the SDA and SCL lines. The divider used a 4.7 kΩ resistor and a 10 kΩ resistor, as calculated in Figure A.8. The purpose of this divider was to reduce the Arduino's 5 V logic level to approximately 3.3 V so the OLED inputs were protected. This was an important design choice because it allowed the 5 V Arduino to communicate safely with the lower-voltage OLED display. The display first shows a welcome page with the project name and team member names, then transitions to the spectrogram screen after a short delay.

Each equalizer band output was connected to an Arduino analog input through a 0.1 µF coupling capacitor and a bias network made from two 100 kΩ resistors. This signal-conditioning stage was necessary because the equalizer outputs are AC signals centered around 0 V, but the Arduino analog inputs can only safely read voltages from 0 V to 5 V. The coupling capacitor removed DC, and the resistor divider biased the signal around 2.5 V so the Arduino could sample both halves of the waveform without clipping. The Arduino code uses software peak detection rather than a hardware peak detector. Software peak detection was chosen because it reduces the number of external components and allows the display response to be adjusted directly in code. The Arduino code in Appendix C samples each band over a short time window, finds the peak-to-peak voltage, estimates the amplitude as half of the peak-to-peak value, and maps that amplitude to the height of a bar on the OLED. The bars are smoothed in software so they increase and decrease gradually rather than jumping abruptly between values. The OLED first displays a welcome screen, shown in Figure 3.22, that includes the project name and team member names.

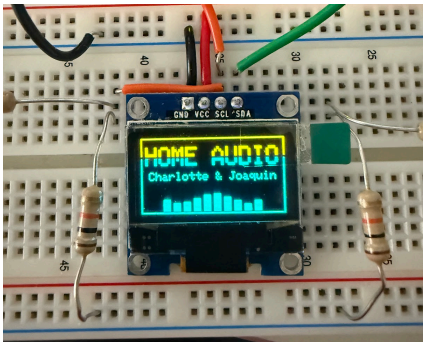

Figure 3.22: *OLED welcome screen displayed during system startup before the spectrogram begins.*

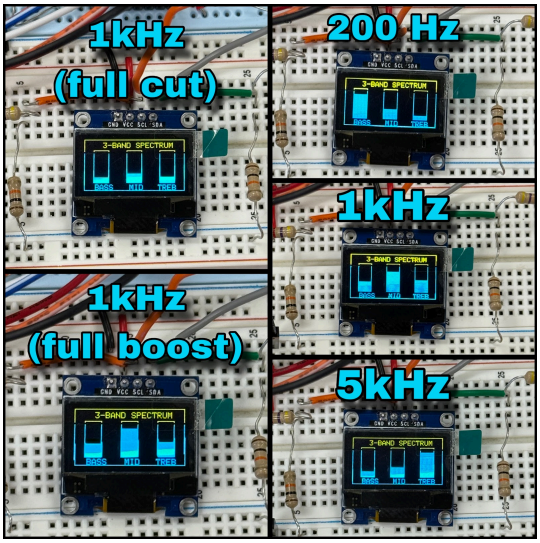

Figure 3.23: *OLED spectrogram display for 200 Hz, 1kHz, and 5kHz sine-wave inputs, along with full-cut and full-boost equalizer settings.*

The spectrogram display was verified using sine-wave inputs at representative frequencies, as shown in Figure 3.23. A 200 Hz input produced the highest bass bar, a 1 kHz input produced the highest mid bar, and a 5 kHz input produced the highest treble bar. This matches the expected frequency ranges of the three-band equalizer and confirms that the OLED is displaying the amplitude of the correct band. Full-cut and full-boost conditions were also tested. In full cut, the displayed bars are reduced as expected, and in full boost, all three bars increase, but the same frequency-dependent behavior remains. For example, with a 1 kHz input in full boost, the mid bar is still the highest while the bass and treble bar rise slightly but are significantly lower, which confirms that the spectrogram continues to reflect the dominant band of the input signal.

4. Conclusion

Throughout this project, we learned important lessons regarding how to break down a complex system into smaller subsystems and build it up in parts, how to effectively troubleshoot system integration errors, and technical information regarding how to build and utilize summing opamps, comparators, level shifters, integrators, inverters, and filters. For example, when it comes to building a complex system in parts, we learned that it's important to not ignore unexpected behavior, even if it doesn't directly affect what you're intending to use the system for. When working on Milestone 2, we found that we were able

to get all the graphs required by the milestone, but that one of the graphs that wasn't required by the milestone, the frequency response of the system when all frequencies were being attenuated, looked very strange. In LTSpice we found that the time domain graphs for this attenuation setting also looked very strange, producing noisy and complicated waves as opposed to an attenuated version of the input signal. However, those graphs weren't required by the milestone, and we didn't see why it would affect the system later on. When performing the full integration test in Milestone 4, it turned out that those graphs were illustrative of a key issue in our graphic equalizer design: the attenuation setting was moving all of our signals out of phase, and the lack of inverting buffers in our filters was causing them to recombine in unexpected ways. Had we paid more attention to the strange graphs in Milestone 2, we could have avoided this issue and saved money on redesigning the PCB, which had been ordered and soldered under the assumption that everything in Milestone 2 was working fine. We also improved our troubleshooting skills, learning to strategically disconnect and test smaller networks in our subsystems to see where our output first began to not align with our simulation. This proved to be much more efficient than taking apart and rebuilding the entire circuit.

Appendix A

Appendix A contains the detailed equations, subsystem schematics, and hand calculations used to support the design choices discussed in Section 3. These figures are referenced throughout the main report so that the design process can be followed without overcrowding the main verification section.

$$\frac{\frac{H}{R_3 boost + R_5} + \frac{1}{R}}{\frac{H}{R_3 cut + R_5} + \frac{1}{R}} = \frac{Vout}{Vin}$$

Equation A.1: *The transfer function for one stage of the graphic equalizer*

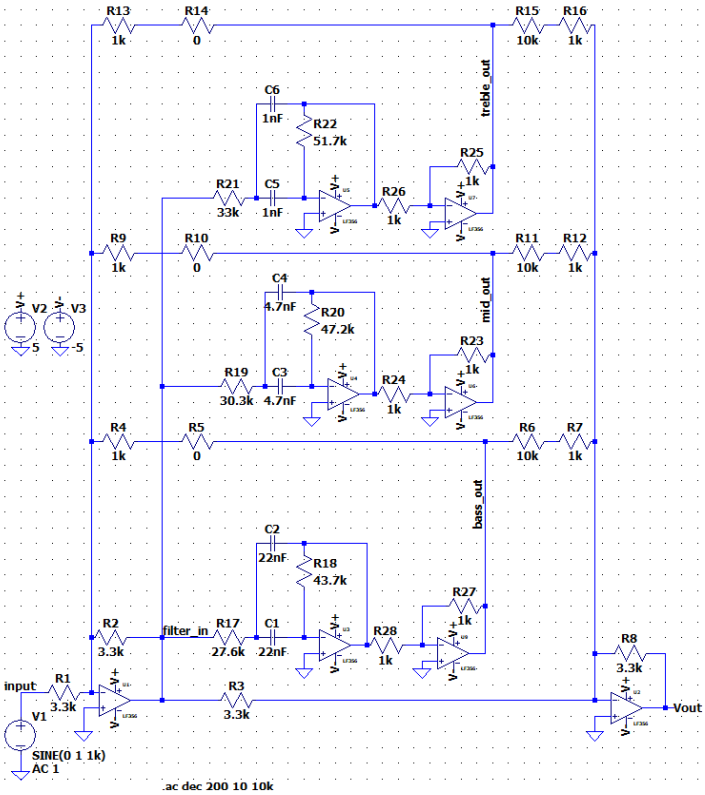

***Figure A.1**: Full Graphic Equalizer LTspice Schematic*

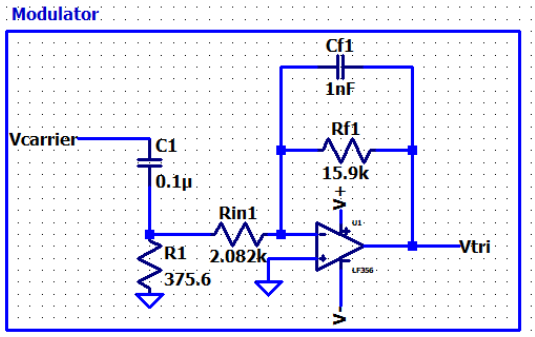

***Figure A.2**: Modulator LTspice Schematic*

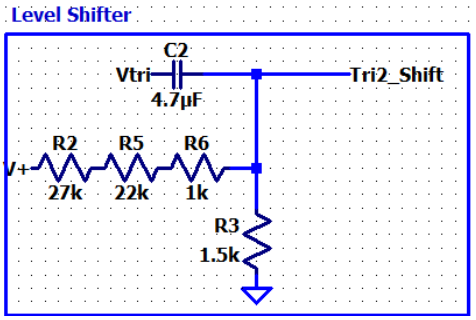

***Figure A.3**: Level Shifter LTspice Schematic*

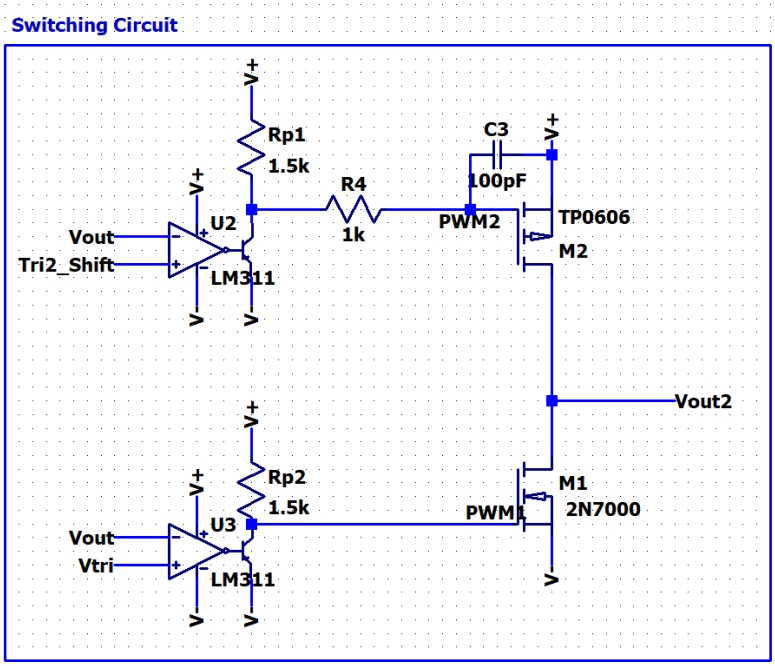

***Figure A.4**: Switching Circuit LTspice Schematic*

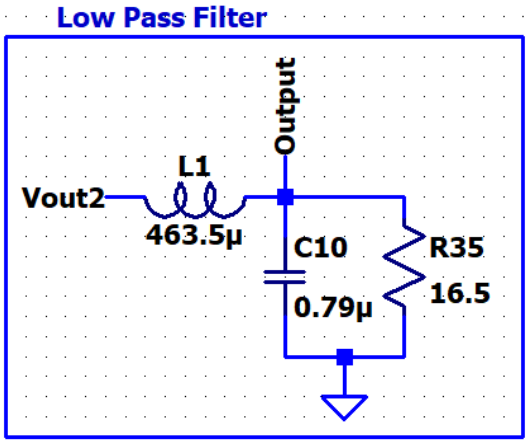

***Figure A.5**: Low Pass Filter LTspice Schematic*

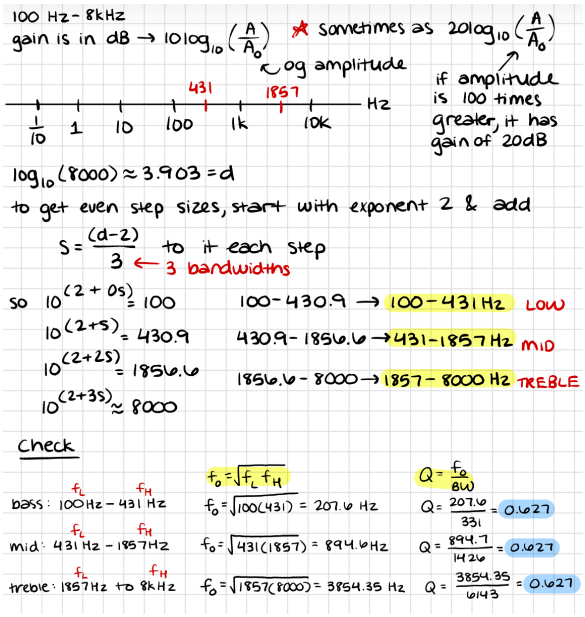

Figure A.6: *Hand calculations showing the log-spaced bandwidths and the computed f_0 and Q values for bass, mid-range, and treble*

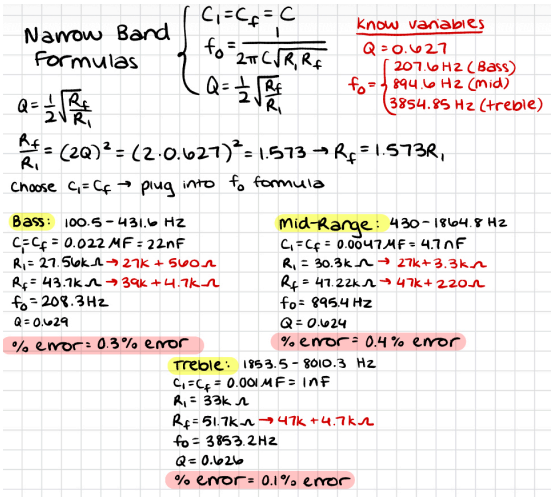

Figure A.7: *Hand calculations with selected practical component values from the lab kit for the bass, mid-range, and treble band-pass filters, showing the revised band edges, center frequencies, Q values, and percent error after redesigning around available resistor/capacitor values*

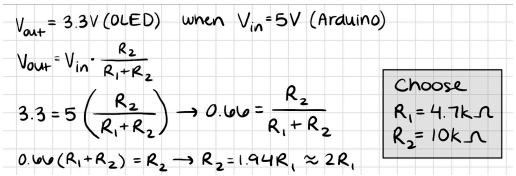

Figure A.8: *Hand calculations for a 5V → 3.3V voltage divider, showing the resistor ratio needed to protect the OLED's 3.3V inputs*

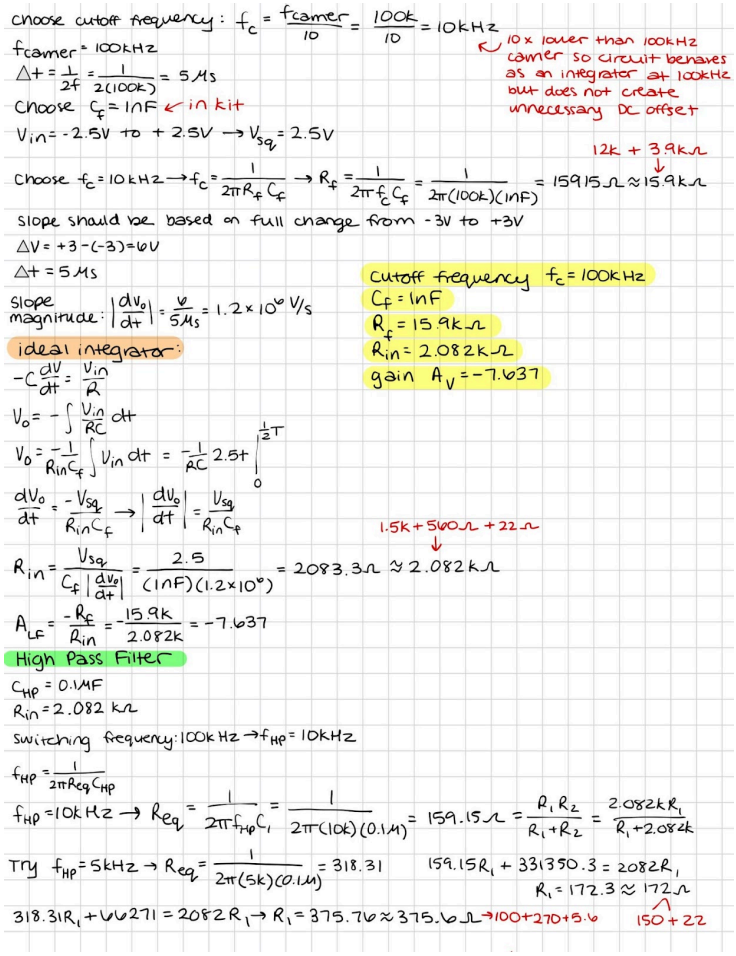

Figure A.9: *Hand calculations for the Class-D modulator design*

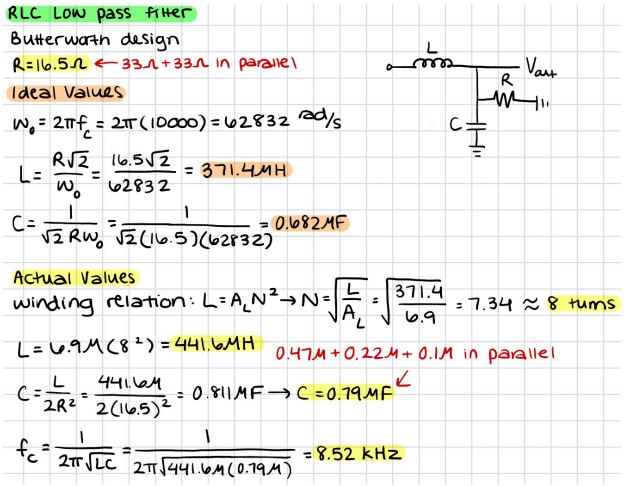

Figure A.10: *Hand calculations used to design the RLC low-pass filter.*

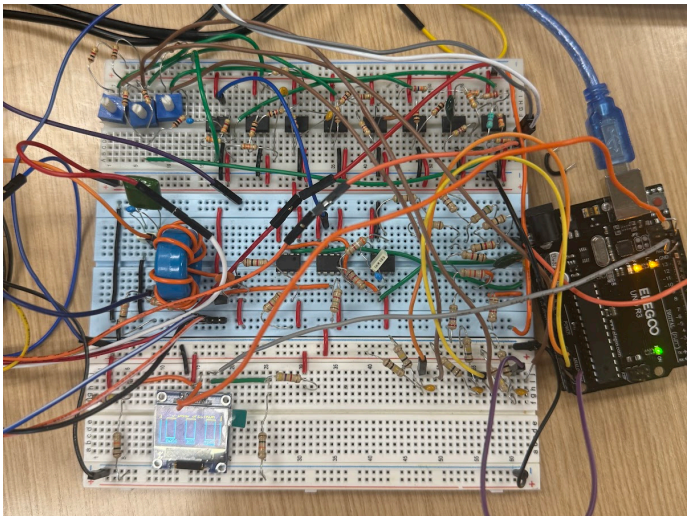

***Figure A.11**: The final home audio system on a breadboard*

Appendix B

As an added opportunity to teach ourselves more about mixed signal PCB design, microsoldering, and to make the circuit easier to assemble and use, we designed and soldered a PCB for the graphic equalizer and another PCB for the Class-D amplifier. We used SMD components in place of THT-based components for all resistors and capacitors. THT components were used for ICs, the inductor, the audio jacks, the header pins, and the MOSFETs to avoid the extra costs associated with purchasing those specific components in an SMD form. Using SMD components instead of the standard breadboard components provides several advantages: the significant reduction in size allows for a more compact board design, which leads to shorter traces. Shorter traces means that there will be less voltage drop across a trace, and the traces will be less capable of acting as antennas and picking up EMI from external sources. The purchased SMD components also have a tolerance rating of 1% instead of 5%, ensuring that our measured results will match the simulation much more accurately. The lack of long pins on SMD components also means that the components themselves will experience less parasitic effects, namely equivalent series inductance (ESL). We made use of a ground plane and a thick power trunk. The ground plane allows currents to have very short return paths to their source, as they're able to take the path of least resistance through the plane instead of being forced to travel through a dedicated ground wire that may not take the optimal route. The power trunk is a much thicker trace that distributes power to components throughout the PCB, reducing impedance at times of high current draw and ensuring that there isn't a significant voltage drop as the current travels from the source to the individual components. The corners of the PCB were given a 1mm fillet to avoid accidentally injuring oneself on sharp corners. To reduce visual clutter, silkscreen names for components were kept to their defaults (U7, R3, etc) instead of giving them more descriptive names. The silkscreen was also used to visually separate distinct groups of components, such as separating the opamps from the potentiometers and the audio jack. The 3D model and wiring diagram of the graphic equalizer can be found in Figures B.1 - B.3

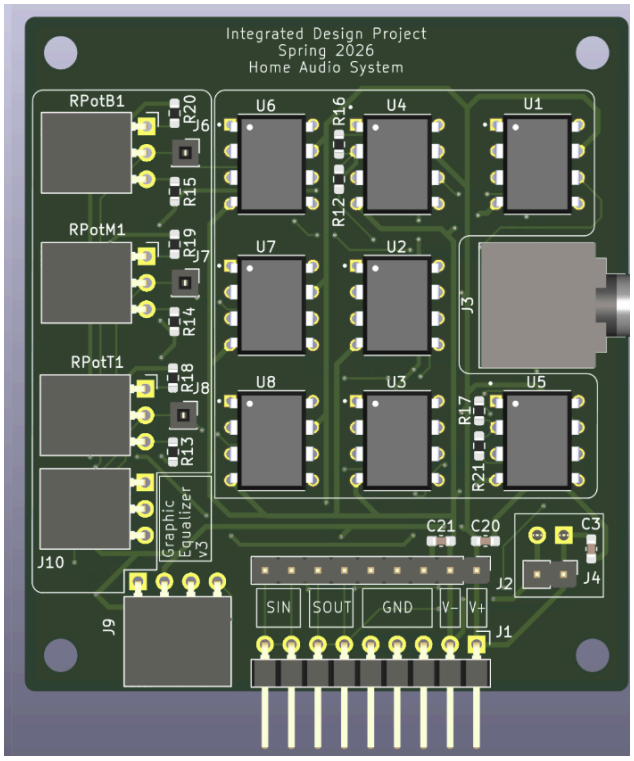

Figure B.1 *The 3D model of the PCB design for the graphic equalizer*

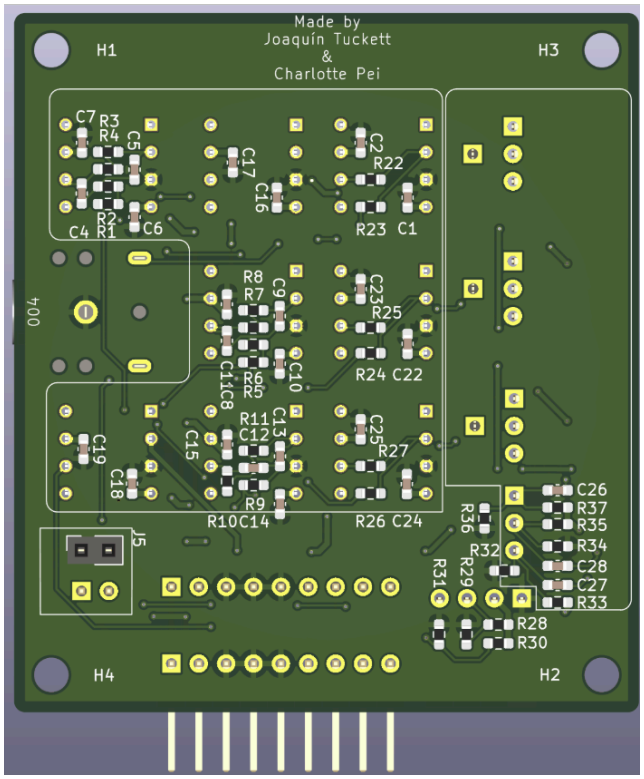

Figure B.2: *The backside of the 3D model in Figure B.1.*

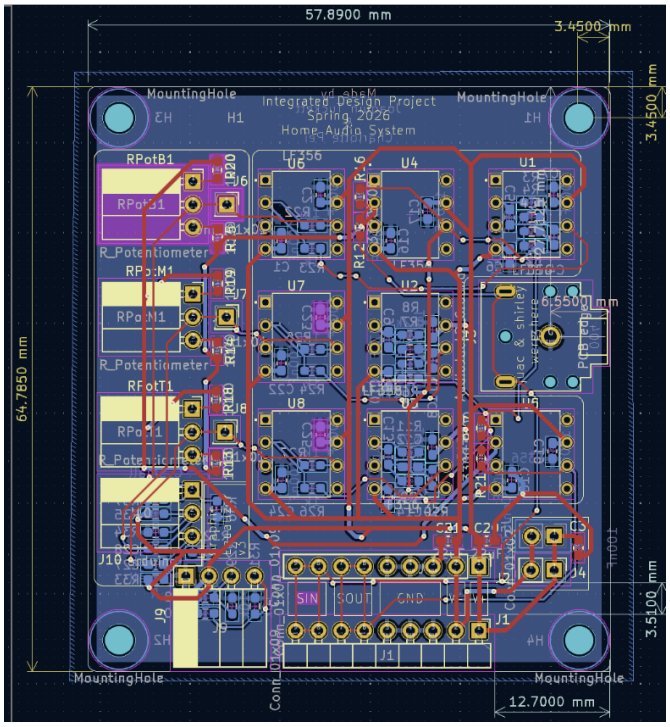

***Figure B.3**: The layout and routing of the PCB design for the graphic equalizer.*

The design of the PCB for the class-D amplifier followed many of the same techniques, making use of SMD components to reduce size, a power trunk, and extra header pins to connect to the graphic equalizer. For all PCBs, we also took care to avoid 90 degree angles in any trace, as 90 degree angles are generally considered to be bad practice. This is due to the extra length and impedance that is introduced by 90 degree angles, which is usually only an issue in high frequency applications, but we opted to abide by these good practices regardless. Though the class-D amplifier had far fewer components, we kept the size the same so that its mounting holes would align with those of the graphic equalizer. The most notable difference is that the class-D amplifier does not use a copper fill to create a ground layer, and instead has individual traces connecting each ground node back to the ground header pins. The constant switching of the transistors on the class-D amplifier introduces significant noise into the circuit, which could be propagated throughout the rest of the system through the ground plane if we were to implement one. To isolate the noise and ensure it doesn't interfere with other signals in the circuit, we use a technique known as "star ground," where there's one central point denoted as ground and each ground pin has its own individual trace connecting it to that central point. This ensures that the return currents don't interfere with one another, thereby isolating the noise introduced by the switching transistors. The design of the class-D amplifier PCB can be found in Figures B.4-B.6.

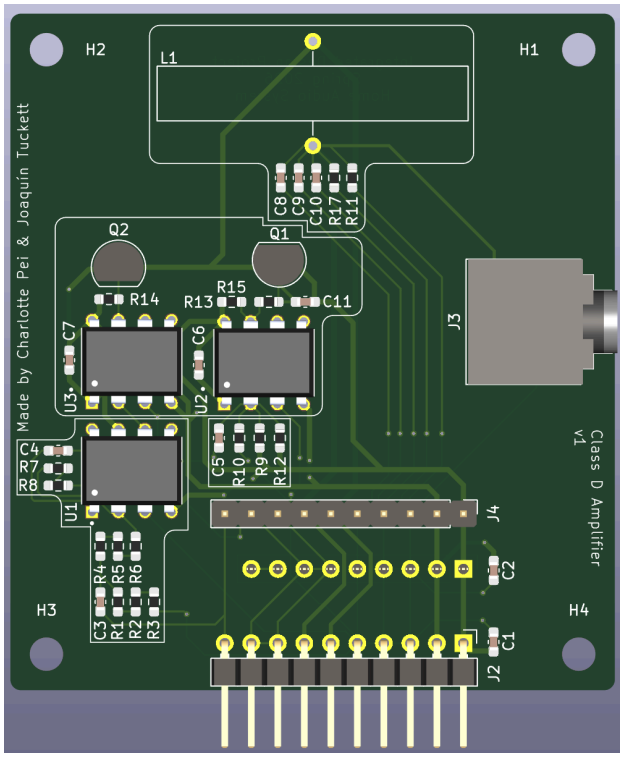

Figure B.4: *The front of the 3D model for the class-D amplifier PCB.*

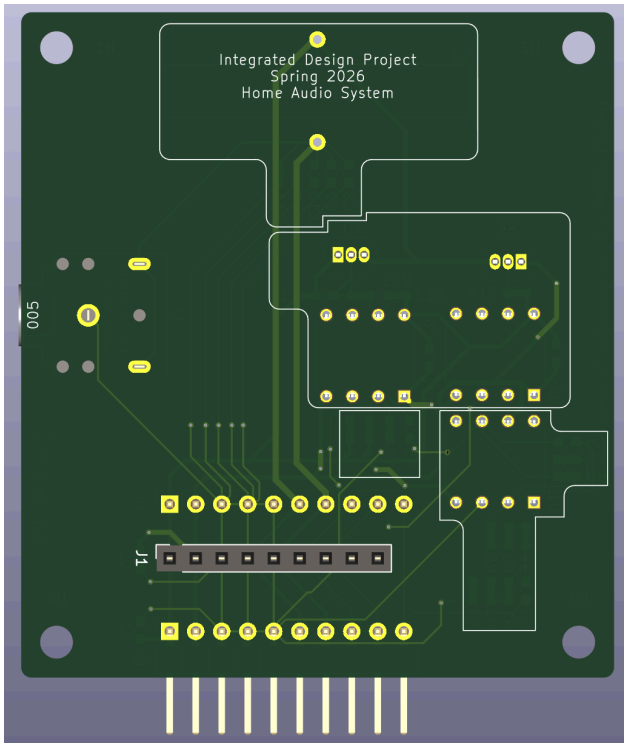

Figure B.5: *The back of the 3D model for the class-D amplifier PCB.*

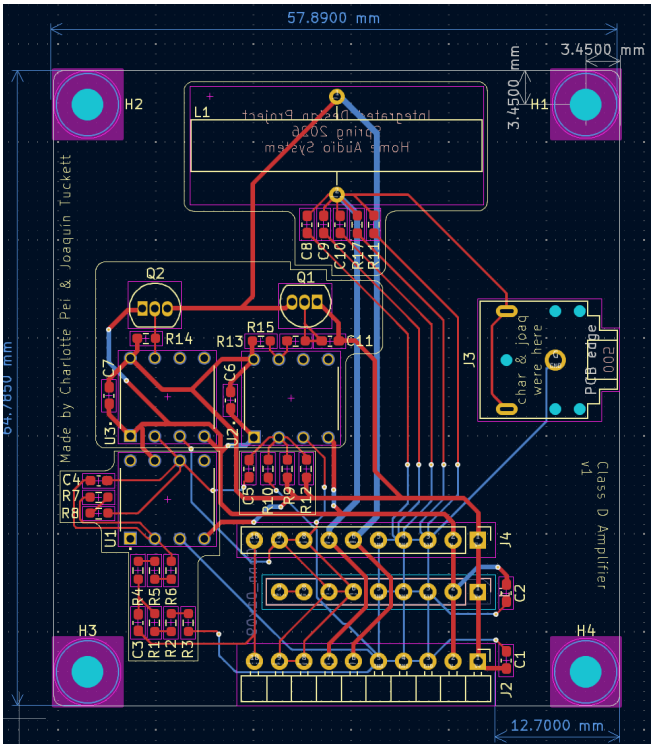

Figure B.6: *The layout and routing of the PCB for the class-D amplifier.*

After assembling, soldering, and troubleshooting, we find that our PCB version of the system is fully functional. It has less noise, a neutral response closer to 0dB, and frequency responses that are more accurately aligned with those of the simulation. Images of the circuit and the individual PCBs can be found in Figures B.8 - B.12 below.

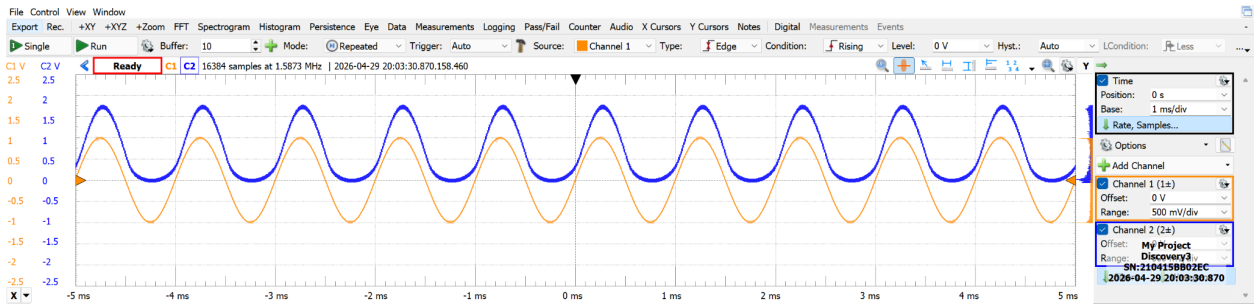

Figure B.7: *The time-domain response of the assembled PCB system.*

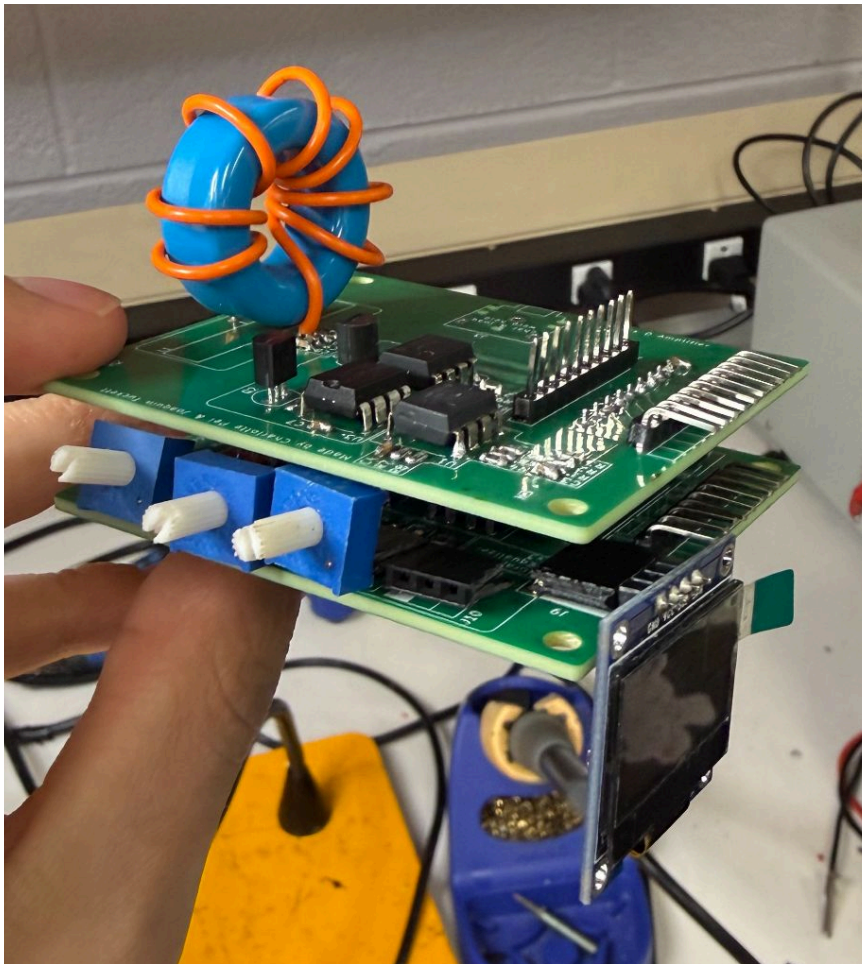

Figure B.8: *A picture of the fully assembled PCB system.*

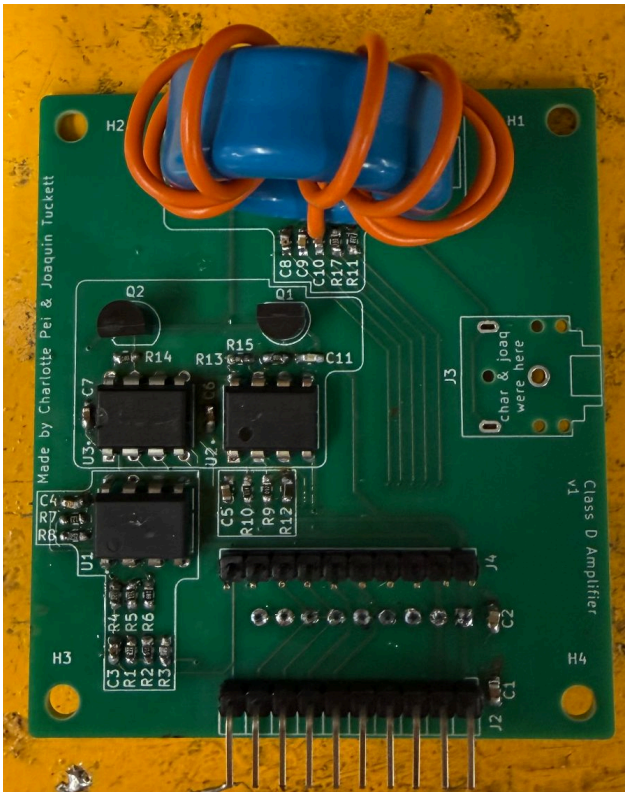

Figure B.9: *The top view of the Class-D amplifier PCB.*

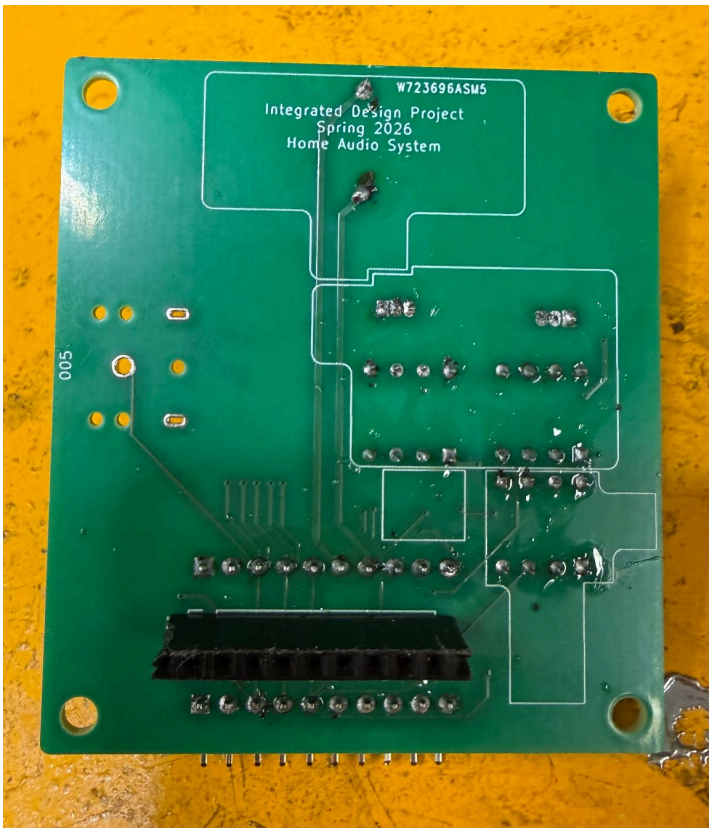

Figure B.10: *A bottom view of the Class-D amplifier PCB.*

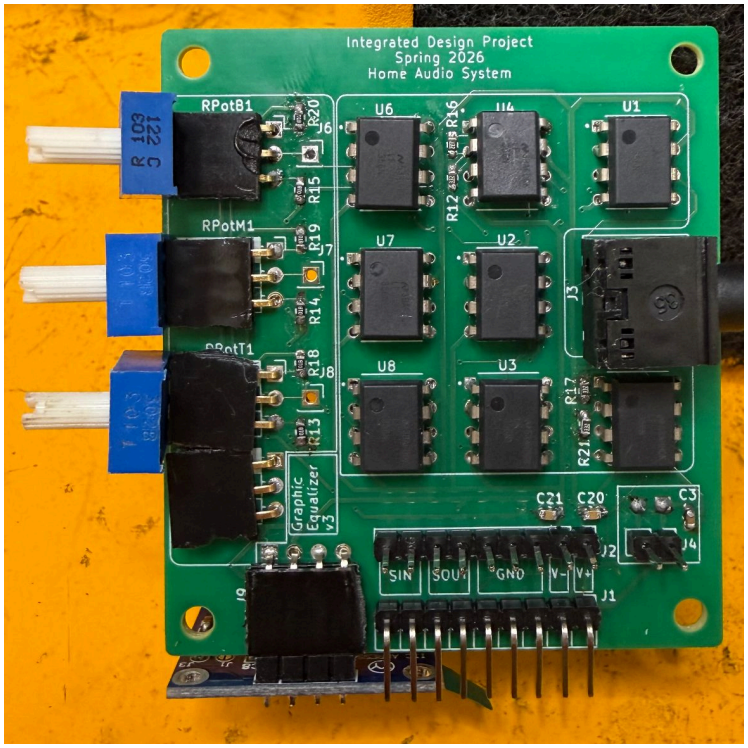

Figure B.11: *A top view of the graphic equalizer PCB.*

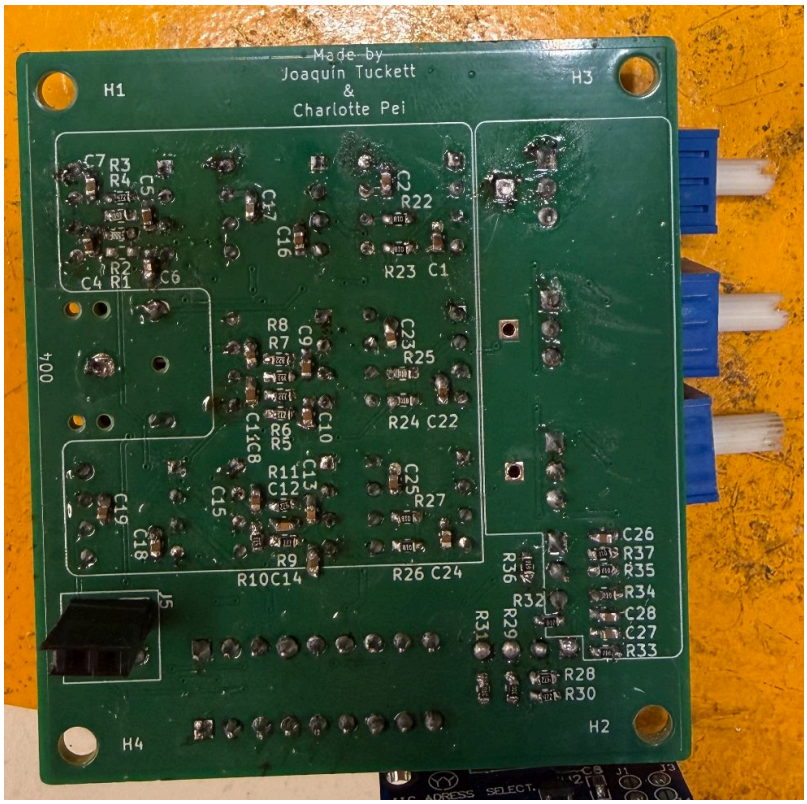

Figure B.12: *A bottom view of the graphic equalizer PCB.*

Appendix C

The code used to program the Arduino is provided below. It was programmed in the Arduino IDE and is capable of controlling the spectrogram and sending the carrier frequency simultaneously.

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

// OLED configuration
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define OLED_ADDR 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Input pins
const int BASS_PIN   = A0;
const int MID_PIN    = A1;
const int TREBLE_PIN = A2;

// 100 kHz square wave output
const int SQUARE_WAVE_PIN = 9;

// Timing
unsigned long startupTime = 0;
const unsigned long WELCOME_DURATION = 3000;

// Smoothed bar heights
float bassBar = 0;
float midBar = 0;
float trebleBar = 0;

// Faster response for music
const float ATTACK  = 0.45;   // bars rise quickly
const float RELEASE = 0.25;   // bars fall reasonably fast

// ADC settings
const float ADC_REF = 5.0;
const int ADC_MAX = 1023;
```

```
// Shorter window so bars respond faster to music
const int SAMPLE_COUNT = 250;

// Per-band display scaling
// Start here, then tune using Serial Monitor
const float BASS_FULL_SCALE   = 0.5;
const float MID_FULL_SCALE    = 0.5;
const float TREBLE_FULL_SCALE = 0.5;

// Small threshold to ignore idle noise
const float NOISE_FLOOR = 0.015;

// Welcome screen helpers
void drawCenteredText(const char* txt, int16_t y, uint8_t size) {
  display.setTextSize(size);
  display.setTextColor(SSD1306_WHITE);

  int16_t x1, y1;
  uint16_t w, h;
  display.getTextBounds(txt, 0, y, &x1, &y1, &w, &h);

  int16_t x = (SCREEN_WIDTH - (int16_t)w) / 2;
  display.setCursor(x, y);
  display.print(txt);
}

void drawEqBars(uint32_t t_ms) {
  const int baseY = 62;
  const int barW  = 6;
  const int gap   = 3;
  const int bars  = 10;

  int totalW = bars * barW + (bars - 1) * gap;
  int startX = (SCREEN_WIDTH - totalW) / 2;

  for (int i = 0; i < bars; i++) {
    int phase = (t_ms / 90 + i * 3) % 20;
    int h = 6 + (phase < 10 ? phase : 20 - phase);
```

```
    int x = startX + i * (barW + gap);
    int y = baseY - h;

    display.fillRect(x, y, barW, h, SSD1306_WHITE);
  }
}

void drawWelcomeScreen() {
  display.clearDisplay();
  display.drawRect(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, SSD1306_WHITE);

  drawCenteredText("HOME AUDIO", 8, 2);
  drawCenteredText("Charlotte & Joaquin", 28, 1);
  drawEqBars(millis());

  display.display();
}

// Read signal amplitude
// Measure amplitude as Vpp/2 over a short sample window
float readAmplitudeVolts(int pin) {
  int minVal = 1023;
  int maxVal = 0;

  for (int i = 0; i < SAMPLE_COUNT; i++) {
    int raw = analogRead(pin);

    if (raw < minVal) minVal = raw;
    if (raw > maxVal) maxVal = raw;
  }

  float vMin = (minVal * ADC_REF) / ADC_MAX;
  float vMax = (maxVal * ADC_REF) / ADC_MAX;
  float vpp = vMax - vMin;
  float vamp = vpp / 2.0;

  if (vamp < NOISE_FLOOR) {
    vamp = 0.0;
  }
```

```
  return vamp;
}

// Smooth motion
float smoothValue(float current, float target) {
  if (target > current) {
    return current + ATTACK * (target - current);
  } else {
    return current + RELEASE * (target - current);
  }
}

// Map amplitude to display height
int mapAmplitudeToHeight(float ampVolts, float fullScaleVolts) {
  if (ampVolts < 0.0) ampVolts = 0.0;
  if (ampVolts > fullScaleVolts) ampVolts = fullScaleVolts;

  return (int)((ampVolts / fullScaleVolts) * 40.0);
}

// OLED spectrogram
void drawSpectrogram(int bassH, int midH, int trebleH) {
  display.clearDisplay();
  display.drawRect(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, SSD1306_WHITE);

  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(24, 4);
  display.print("3-BAND SPECTRUM");

  int baseY = 54;
  int barW = 20;

  int bassX = 12;
  int midX = 54;
  int trebleX = 96;

  display.drawRect(bassX,   baseY - 40, barW, 40, SSD1306_WHITE);
  display.drawRect(midX,    baseY - 40, barW, 40, SSD1306_WHITE);
  display.drawRect(trebleX, baseY - 40, barW, 40, SSD1306_WHITE);
```

```
  display.fillRect(bassX,   baseY - bassH,   barW, bassH,
SSD1306_WHITE);
  display.fillRect(midX,    baseY - midH,    barW, midH,
SSD1306_WHITE);
  display.fillRect(trebleX, baseY - trebleH, barW, trebleH,
SSD1306_WHITE);

  display.setCursor(14, 56);
  display.print("BASS");
  display.setCursor(58, 56);
  display.print("MID");
  display.setCursor(94, 56);
  display.print("TREB");

  display.display();
}

// 100 kHz square wave
void setupSquareWave100kHz() {
  pinMode(SQUARE_WAVE_PIN, OUTPUT);

  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;

  TCCR1A |= (1 << WGM11);
  TCCR1B |= (1 << WGM12) | (1 << WGM13);

  TCCR1A |= (1 << COM1A1);
  TCCR1B |= (1 << CS10);

  ICR1 = 159;
  OCR1A = 80;
}

// Setup
void setup() {
  Serial.begin(9600);
```

```
  Wire.begin();
  PORTC &= ~((1 << PORTC4) | (1 << PORTC5));
  Wire.setClock(400000);

  if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)) {
    for (;;) {}
  }

  pinMode(BASS_PIN, INPUT);
  pinMode(MID_PIN, INPUT);
  pinMode(TREBLE_PIN, INPUT);

  setupSquareWave100kHz();

  display.clearDisplay();
  display.display();

  startupTime = millis();
}

// Main loop
void loop() {
  unsigned long elapsed = millis() - startupTime;

  if (elapsed < WELCOME_DURATION) {
    drawWelcomeScreen();
    delay(30);
    return;
  }

  float bassAmp   = readAmplitudeVolts(BASS_PIN);
  float midAmp    = readAmplitudeVolts(MID_PIN);
  float trebleAmp = readAmplitudeVolts(TREBLE_PIN);

  Serial.print("Bass: ");
  Serial.print(bassAmp, 3);
  Serial.print(" V   Mid: ");
  Serial.print(midAmp, 3);
  Serial.print(" V   Treble: ");
  Serial.print(trebleAmp, 3);
```

```
  Serial.println(" V");

  int bassTarget   = mapAmplitudeToHeight(bassAmp,   BASS_FULL_SCALE);
  int midTarget    = mapAmplitudeToHeight(midAmp,    MID_FULL_SCALE);
  int trebleTarget = mapAmplitudeToHeight(trebleAmp, TREBLE_FULL_SCALE);

  bassBar   = smoothValue(bassBar, bassTarget);
  midBar    = smoothValue(midBar, midTarget);
  trebleBar = smoothValue(trebleBar, trebleTarget);

  drawSpectrogram((int)bassBar, (int)midBar, (int)trebleBar);

  delay(15);
  }
```

Authorship Table

Section	Author	Revised By
Introduction	Charlotte Pei	Joaquin Tuckett
High Level Design	Joaquin Tuckett	Charlotte Pei
Detailed Design & Verification	Charlotte Pei	Joaquin Tuckett
Validation of the Overall Project	Joaquin Tuckett	Charlotte Pei
Conclusion	Joaquin Tuckett	Charlotte Pei
PCB Design & Soldering	Joaquin Tuckett	N/A
OLED Programming	Charlotte Pei	N/A

Table 1: *The authorship table for the final report*