

UCF Senior Design 1 (Final Report)

Smart Auto-Equalization Sound System



Group 31 Authors:

Xander Levine
Computer
Engineering

Noah Bixby
Electrical
Engineering

Jordan Riley
Electrical
Engineering

Matias Segura
Computer
Engineering

Review Committee:

Mark Maddox
Instructor

Dr. Sonali Das
Lecturer

Dr. Maedeh Sadat Fasihi
Adjunct Lecturer

Mentor:

Dr. Sreeram Sundaresh

Table of Contents

Chapter 1 – Executive Summary	0
Chapter 2 – Project Description.....	1
2.1 Motivation and Background.....	1
2.2 Goals and Objectives.....	1
2.2.1 Basic Goals	1
2.2.2 Advanced Goals.....	1
2.2.3 Stretch Goals.....	2
2.2.4 Basic Objectives	2
2.2.5 Advanced Objectives	2
2.2.6 Stretch Objectives.....	2
2.3 Features and Functionalities.....	2
2.4 Related Work.....	3
2.4.1 Commercial Systems	3
2.4.2 Prior Academic Projects	3
2.5 Key Engineering Specifications	4
2.5.1 Highlighted Demonstrable Specs:	4
2.5.2 Hardware Block Diagram	5
2.5.3 Software Flowchart.....	7
2.5.4 Prototype Illustration	9
2.5.5 House of Quality.....	11
Chapter 3 – Research	11

3.1 Power Amplifier Technologies	11
3.1.1 Class A Amplifiers	12
3.1.2 Class B Amplifiers.....	15
3.1.3 Class AB Amplifiers.....	17
3.1.4 Class D Amplifiers	18
3.1.5 IC Audio Power Amplifiers.....	19
3.1.6 IC and Discrete Combination Amplifiers	19
3.1.7 Comparisons	20
3.1.8 Summary.....	20
3.1.9 Senior Design 2 Implementation	20
3.2 Amplifier Comparisons and Selection	21
3.2.1 Mids/Highs Amplifiers	21
3.2.2 Subwoofer Amplifiers	24
3.3 Speaker Consideration, Comparison, and Selection	27
3.3.1 Considerations for the High Speaker.....	27
3.3.2 Comparison for the High Speaker	27
3.3.3 Selection for the High Speaker.....	28
3.3.4 Senior Design 2 Implementation	28
3.3.5 Considerations for the Mid Speaker	28
3.3.6 Comparison for the Mid Speaker.....	29
3.3.7 Selection for the Mid Speaker	29
3.3.8 Senior Design 2 Implementation	30

3.3.9 Considerations for the Low Speaker	30
3.3.10 Comparison of the Low Speaker	30
3.3.11 Selection for the Low Speaker.....	31
3.3.12 Senior Design 2 Implementation	31
3.4 Speaker Enclosure Design.....	31
3.4.1 Importance of Speaker Characteristics	32
3.4.2 Open Baffle.....	32
3.4.3 Sealed Enclosure.....	32
3.4.4 Ported Enclosure.....	32
3.4.5 Bandpass Subwoofer Enclosures.....	32
3.4.6 Enclosure Design Comparison	33
3.4.7 Enclosure Material.....	33
3.4.8 Summary.....	34
3.4.9 Senior Design 2 Implementation	34
3.5 Crossover Filters	34
3.5.1 What are Crossover Filters?	34
3.5.2 Comparison of Passive, Hybrid, Active Crossover Filters	34
3.5.3 Cross Over Filter for Mid and High Speakers.....	35
3.5.4 Crossover Filter for Low Speaker	35
3.5.5 Senior Design 2 Implementation	36
3.6 Distance Sensor	36
3.6.1 Considerations	36

3.6.2 Conclusion	37
3.7 Microphones.....	37
3.7.1 Speaking Microphone vs. Audio Measurement Microphone	37
3.7.2 Considerations for System Microphone	38
3.7.3 Comparison of System Microphones	38
3.7.4 Selection for System Microphone	39
3.7.5 Senior Design 2 Implementation	39
3.8 Processor Technology Comparisons	40
3.8.1 SoC (DSP Chip).....	40
3.8.2 MCU	40
3.8.3 FPGA	41
3.8.4 Processor Comparison and Selection	41
3.8.5 MCU Comparisons	41
3.8.6 Summary & Selection.....	45
3.8.7 Senior Design 2 Implementation	46
3.9 Phone Communication with MCU	46
3.9.1 Communication Technologies	46
3.9.2 Bluetooth Connection	47
3.9.3 USB Connection	50
3.9.4 Senior Design 2 Implementation	52
3.10 DAC/ADC vs CODEC.....	52
3.10.1 16-bit vs 24-bit Resolution	52

3.10.2 CODEC.....	53
3.10.3 DAC.....	54
3.10.4 Summary & Conclusion	54
3.10.5 DAC Module Comparison and Selection	54
3.10.6 Further Considerations	55
3.10.7 Senior Design 2 Implementation	56
3.11 Power Supply Unit (PSU)	56
3.11.1 PSU Requirements.....	56
3.11.2 Power Supply Options	57
3.11.3 Summary.....	58
3.11.4 Senior Design 2 Implementation	58
3.12 Technology Research and Comparison.....	58
3.12.1 Cross-Platform Frameworks.....	58
3.12.2 Native Mobile Development	59
3.12.3 GUI Libraries and Design Standards.....	59
3.12.4 Backend Integration Strategies	59
3.12.5 Security Considerations	60
3.12.6 Testing Tools	60
3.12.7 Technology Comparison.....	60
3.12.8 Summary.....	61
3.12.9 Senior Design II Implementation	61
3.13 Test Signal.....	61

3.13.1 White Noise	62
3.13.2 Pink Noise.....	62
3.13.3 Sine Sweep (Log vs Linear)	62
3.13.4 Senior Design 2 Implementation	62
Chapter 4 – Standards and Design Constraints	62
4.1 Design Standards.....	62
4.1.1 Bluetooth Standard	62
4.1.2 USB Standards.....	64
4.1.3 Flutter Standards.....	65
4.1.4 PCB Standard	67
4.2 Design Constraints	69
4.2.1 Power Constraints.....	69
4.2.2 Audio Constraints	70
4.2.3 Financial Constraint.....	72
4.2.4 Time Constraint	73
Chapter 5 – Application of ChatGPT and Other Platforms	73
5.1 Overview	73
5.2 Brief History of AI	73
5.3 Advantages & Disadvantages of AI	75
5.4 Rules and Expectations of Group AI Usage.....	76
5.4.1 Purpose of Guidelines.....	76
5.4.2 Accepted Uses of AI.....	76

5.4.3 Unacceptable Uses of AI	77
5.4.4 Verification and Accountability	77
5.4.5 Team Ethic and Learning Value	77
5.5 Xander Levine's AI Use.....	77
5.5.1 Flutter Build/Debug Assistance.....	77
5.6 Noah Bixby's AI Use	78
5.6.1 Case [2a] – MCU Buck Regulator Selection & Sizing	78
5.6.2 Case [2b] – Class-D Mid/High Amp Supply Details (TPA3116D2 IVDD & Edge Conditions)	79
5.6.3 Case [2c] – 36 V Rail Strategy from 34.2–39.6 V Source (Buck vs. Fixed-Output PSU).....	79
5.7 Jordan Riley's AI Use	80
5.8 Matias Segura's AI Use	80
5.8.1 [3a] Parallel DAC Implementation.....	80
5.8.2 [3b] DAC Recommendation.....	80
Chapter 6 – Hardware Design.....	81
6.1.1 Regulators	81
6.1.2 Digital Circuitry Regulator.....	81
6.1.3 Senior Design 2 Implementation – Analog and Digital 3.3V Regulator.....	81
6.1.4 +5V and -5V Regulators.....	83
6.1.5 Senior Design 2 Implementation - +/- 5V Regulator	83
6.1.6 +12 Volt Regulator	85
6.1.7 Senior Design 2 Implementation - +12V Regulator.....	85

6.1.8 Mid/High Amplifier Regulator	86
6.1.9 Senior Design 2 Implementation – Mid/High Regulator.....	87
6.2 Amplifier Schematics	87
6.2.1 Subwoofer Amplifier	87
6.2.2 Mid/High Amplifier.....	88
6.2.3 Senior Design 2 Implementation - Amplifiers	89
6.3 Digital Schematics.....	89
6.3.1 Main MCU Schematic	90
6.3.2 DAC Schematic	92
6.3.3 Senior Design 2 Implementation	93
Chapter 7 – Software Design	95
7.1 Introduction to GUI and Mobile Application.....	95
7.1.1 Role of the GUI in the Project System	96
7.1.2 Significance of the Mobile Application.....	96
7.1.3 Integration with Embedded System.....	98
7.2 App Architecture and Flow	99
7.2.1 System Overview.....	99
7.2.2 Use Case Flow	99
7.2.3 Activity Flow	100
7.2.4 Data Handling and Synchronization.....	101
7.2.5 Security and Reliability	101
7.2.6 Summary.....	101

7.2.7 Senior Design II Implementation and Changes	102
7.3 Auto-EQ Algorithm.....	103
7.3.1 1 st Stage: Signal Generation and Data Gathering	104
7.3.2 2 nd Stage: Signal Processing	104
7.3.3 3 rd Stage: Filter Design	107
7.3.4 4 th Stage: Playback & Correction Application	108
Chapter 8 – System Fabrication/Prototype Construction.....	109
8.1 Analog Signal PCB	109
8.1.1 Senior Design 2 Implementation – Analog PCB.....	110
8.2 Digital Signal PCB	113
8.2.1 Senior Design 2 Implementation – Digital PCB	116
8.3 Speaker Cabinet Fabrication	118
8.3.1 Mids/Highs Enclosure	118
8.3.2 Subwoofer Enclosure.....	118
8.3.3 Senior Design 2 Implementation	119
Chapter 9 – System Testing and Evaluation	120
9.1 Microphone Testing	121
9.1.1 Test Equipment and Components.....	121
9.1.2 Electrical Setup.....	121
9.1.3 ESP32 Setup	123
9.1.4 Tests.....	124
9.1.5 Conclusion	125

9.2 Embedded System Prototype	125
9.2.1 Flutter Application to ESP32 Integration	125
9.2.2 ESP Firmware Functionality	125
9.2.3 Fabrication Constraints	127
9.2.4 Flutter App / ESP32 BLE Summary and Prototype Conclusion	127
9.3 Audio Testing.....	128
9.3.1 Test Equipment.....	128
9.3.2 Electrical Setup.....	128
9.3.3 Software Setup.....	129
9.3.4 Issues Faced.....	130
9.3.5 Test Results.....	131
9.3.6 Conclusion	131
9.4 Senior Design 2 System Verification	131
Chapter 10 – Administrative Content	132
10.1 Budget and Financing.....	132
10.2 Project Milestones	133
10.2.1 Senior Design I	133
10.2.2 Senior Design II.....	133
10.3 Work Distributions.....	133
Chapter 11 Conclusion.....	134

List of Figures

Figure 2.1 Hardware Block Diagram.....	5
Figure 2.2 Software Flow Chart.....	7
Figure 20.3 User Interface	8
Figure 20.4 System prototype illustration, Angled View	9
Figure 20.5 System prototype illustration, Front View	10
Figure 20.6 House of Quality.....	11
Figure 3.1 Class A Amplifier Topology	12
Figure 3.2 Class A Load Line	14
Figure 3.3 Class B Amplifier Topology	15
Figure 3.4 Class AB Amplifier Topology.....	17
Figure 3.5 Class D Amplifier Block Diagram	18
Figure 6.1 +3.6V Regulator Schematic	81
Figure 6.6 Subwoofer Amplifier Schematic	87
Figure 6.7 Mid/High Speaker Amplifier Schematic	88
Figure 6.8 Digital Board Schematic 1.....	90
Figure 6.9 DAC Schematic	92
Figure 7.1 Mobile App Home Page View	97
Figure 7.2 Mobile App Control Page View	97
Figure 7.3 Mobile App Settings Page View	98
Figure 7.4 App Functional Layer Block Diagram	99

Figure 7.5 System Interaction Flow	100
Figure 7.6 Activity Flow Block Diagram	101
Figure 7.7 Test Signal Generation and Data gathering	104
Figure 7.8 Impulse Response Representation	105
Figure 7.9 Digital Signal Processing Step	107
Figure 7.10 Datapath Overall Flow	108
Figure 8.1 Analog PCB 2D View	109
Figure 8.2 Analog PCB 3D View	109
Figure 9.1 Basic Phantom Power Circuitry	122
Figure 9.2 Full Microphone Test Setup	123
Figure 9.3 Microphone Circuitry Close Up	123
Figure 9.4 Audio Test Setup	129

List of Tables

Table 2.1 Engineering Specifications Table	4
Table 3.1 Amplifier Topology Comparisons	20
Table 3.2 Mids/Highs Amplifier Comparison	23
Table 3.3 Subwoofer Amplifier Comparison.....	26
Table 3.4 High Speaker Comparison	28
Table 3.5 Mid Speaker Comparison	29
Table 3.6 Low Speaker Comparison.....	31
Table 3.7 Enclosure Design Comparison.....	33

Table 3.8 Material Comparison Table	33
Table 3.9 Distance Sensor Comparison	37
Table 3.10 Microphone Comparison Chart	39
Table 3.11 Processor Technology Comparison	41
Table 3.12 MCU Comparison Summary	45
Table 3.13 Bluetooth vs Wi-Fi Comparison	47
Table 3.14 Bluetooth Technology Comparison	48
Table 3.15 Bluetooth Module Comparison.....	50
Table 3.16 - USB Technology Comparison.....	51
Table 3.1717 DAC Module Part Comparison.....	55
Table 3.1818 Power Requirements	56
Table 3.1919 PSU Comparison Table	57
Table 3.2020 Comparison of GUI & Mobile App Development Technologies.....	60
Table 8.1 Input and Output Connections	115
Table 9.1 List of Components.....	121
Table 9.2 GATT Service Definitions.....	126
Table 9.3 List of Components.....	128
Table 10.1 Bill of Materials	132
Table 10.2 Senior Design 1 Milestones	133

Chapter 1 – Executive Summary

The Smart Auto-Equalization Sound System is a Senior Design project developed by electrical and computer engineering students at the University of Central Florida. The objective of this project is to design a 2.1 audio system capable of automatically analyzing the listening environment and adjusting the frequency response of the system to match the acoustics of such environment. Traditional speaker systems require manual equalization, tuning experience, or external calibration hardware. In contrast, this system integrates embedded signal processing, custom amplification hardware, and a mobile application to deliver an automated, user-friendly tuning experience.

At the core of the system is the ESP32-S3 microcontroller, which performs real-time digital signal processing (DSP). A calibrated measurement microphone captures the room's acoustic response while the system plays a controlled audio test signal. Using this recorded data, the ESP32-S3 executes mathematical operations using the ESP-DSP library, including Fast Fourier Transform (FFT), windowing, smoothing, and adaptive filtering, to identify peaks, dips, and resonances across the audible spectrum. This analysis allows the system to generate a customized EQ profile designed to flatten or optimize the room's frequency response.

To deliver high-quality audio output, the system includes two custom-designed analog amplifiers: a dual-channel amplifier for mid/high speakers and a dedicated mono amplifier for the subwoofer. These amplifiers are implemented on a custom PCB alongside power regulation circuitry, the microphone interface, the DAC/ADC modules, and the ESP32-S3. Together, the analog and digital subsystems form a fully integrated 2.1 audio solution with real-time DSP correction capabilities.

The system also incorporates a mobile application that enables users to control core functions, including initiating the Auto-EQ routine or manually adjusting EQ bands. The ESP32-S3 communicates with the mobile app through wireless connectivity via the ESP's BLE.

This report documents the entire design process: motivation, goals and objectives, engineering research, hardware and software design, system analysis, fabrication, and preliminary testing. It also explains the theoretical principles behind digital equalization, amplifier design, microphone calibration, and embedded DSP implementation. The Smart Auto-Equalization Sound System demonstrates the feasibility of combining embedded computing, custom analog hardware, and acoustic analysis into a unified consumer audio product.

Chapter 2 – Project Description

2.1 Motivation and Background

Audiophiles all over the world spend thousands of dollars on audio equipment and multiple hours of tuning to achieve the best performance for their respective systems and environments. In many cases, these consumers are buying multiple subsystems to manually tune the system. Each tune is only good for the exact environment and system the tune was made for, meaning that any small change to the environment or system calls for the entire system to be re-tuned. This can be quite a headache for audiophiles who want a plug and play system. This is where the Smart Auto-Equalization Sound System idea strives.

The Smart Auto-EQ Sound System aims to integrate a microcontroller (MCU), audio amplifier, and a calibrated microphone to measure the environment's natural frequency response to automatically adjust the audio file's equalization (EQ) and feed the signal to the audio amplifier. While implementing a mobile app interface or web-based connection, we can provide the user with a more tailored sound EQ experience.

2.2 Goals and Objectives

2.2.1 Basic Goals

- Design and build a functional 2.1 speaker system.
- Develop a custom 2 channel amplifier using the MCU as the input for the mid/highs.
- Develop a mono channel amplifier using the MCU as the input for the subwoofer.
- Integrate a microcontroller (ESP32-S3) for DSP-based EQ adjustment.
- Devise an algorithm that uses analog input signals to construct an auto-EQ system.
- Develop a mobile app that works alongside the system that provides a manual override for customer preferences and/or monitoring.

2.2.2 Advanced Goals

- Perform a logarithmic sine sweep to assist the MCU in ensuring a flat EQ response to start with when tailoring the adjustment for each location the speakers are in.
- Combine DSP with analog circuitry for low latency situations.
- Implementing adaptive filters such as Least Mean Squared, or Recursive Mean Squared algorithms to stabilize given noisy or crowded environments.

2.2.3 Stretch Goals

- Support Bluetooth audio streaming that incorporates EQ.
- Implementation of voice control/assist, such as Siri, Alexa, Bixby, to help with EQ control settings.
- Prevent clipping or distortion by actively monitoring the Total Harmonic Distortion and adjusting in real time.
- Prevent latency when switching between a Wi-Fi or Bluetooth connection.

2.2.4 Basic Objectives

- Utilize a calibrated microphone to pick up the speakers' response when exposed to a test tone. This data will help us with signal processing.
- Create an algorithm that is fast, reliable, and that will ensure the best performance, with the use of the ESP-32's DSP library.
- Implement an LED that will blink to let the user know it is working on Auto-EQ. When it finishes, the LED will stop blinking, which will indicate to the user that the run is complete.
- Investigate the correct amplification factor to achieve sufficient power to the system.
- Achieve a base equalization setting. This will help us and the algorithm have a starting point, to make the equalizer settings easier for the user and the MCU.

2.2.5 Advanced Objectives

- Code an app using Java/Python where the user can choose between running Auto-EQ and manually changing the settings to their convenience.
- Provide a fast and reliable connection between the app and the system. The ESP32-S3 comes with built-in Bluetooth (BLE) and WI-FI modules that will help achieve this task. A separate module could be considered if necessary.

2.2.6 Stretch Objectives

- Achieve a runtime of $N \cdot \log(N)$, where N is the number of samples. We believe that the ESP-32S3 can handle the computational power required for this task. We can achieve this by using FFTs, as well as the ESP's dedicated library for DSP.
- Provide the user with real-time data. Histograms and charts are helpful for displaying data.

2.3 Features and Functionalities

- Equalizer calibration using a calibrated microphone.
- Custom amplifiers are built for the subwoofer and mid/high drivers.
- Real-time DSP EQ on an embedded MCU.
- Custom PCB with modules including MCU, DAC/ADC, Bluetooth module, USB-A connector, amplifiers, LEDs, ON/OFF buttons, and voltage regulators.
- Mobile app interface for user control.

- Digital Crossover Filters to avoid high frequencies on the subwoofer side and low ones on the mids & highs.

2.4 Related Work

2.4.1 Commercial Systems

- DSPEaker Anti-Mode X2 ⁽¹⁾
- DSPEaker Anti-Mode 2.0 Dual Core Room Correction Floor Sample ⁽²⁾
- IK Multimedia ARC Studio Room Correction System ⁽³⁾

These commercial systems contain similar functionality to the project we are designing here, however on a smaller scale. These products are designed for car stereo systems while we are designing a system for a much larger scale with rooms. In addition, these products are no longer sold through their website, giving us an edge in the market.

- REW: Room Equalizer Wizard ⁽⁴⁾

This software aims to help the user to find the most suitable equalizing settings based on their room's acoustic response. The user must have access to a computer to download the software. In addition, the program requires a microphone that is used to pick up the signals. Our project differs from this software by first integrating an entire sound system together with the software; this way, the user does not need to get any other equipment.

2.4.2 Prior Academic Projects

- Pegasus: A Multi-Effect Audio Processor ⁽⁵⁾

This was a project that was completed last fall, 2024, in the Senior Design Capstone course, here at UCF. Like our project, this group is adjusting their audio outputs to create a new signal. The difference, however, is that this project is creating different effects, such as adjusting the pitch, tempo, and reverb of the output signal, which would mean the sound quality is fixed and is the same from one location to the next.

2.5 Key Engineering Specifications

Table 2.1 Engineering Specifications Table

Specification	Target Value	Notes
Frequency Response Range	20 Hz – 20 kHz	Full audio spectrum
Total Harmonic Distortion (THD)	$\leq 0.5\%$	At rated power
Amplifier Output Power	50W RMS (subwoofer), 25W RMS (mid/high)	Per channel
MCU Processing	≥ 240 MHz dual-core with FPU/DSP	ESP32-S3 candidate
LiDAR Accuracy	$\leq \pm 30$ cm for ToF	Room measurements
Microphone Accuracy	$\leq \pm 1$ dB	Acoustic measurements
Power Supply	$\pm 12V$ to $\pm 24V$ (audio), $+5V/+3.3V$ (logic)	Linear + switching regulation
Latency (Mic-to-DSP-EQ)	≤ 20 ms	Real-time correction

2.5.1 Highlighted Demonstrable Specs:

- Achieving a frequency range of 20Hz – 20kHz for a fully enjoyable audio spectrum for the user.
- Having an output power of 50W RMS for our subwoofer and 25W RMS for the mid/high speakers.
- With customer satisfaction in mind, we want to ensure our speakers produce clear crisp sound, hence highlighting the limit of distortion.

2.5.2 Hardware Block Diagram

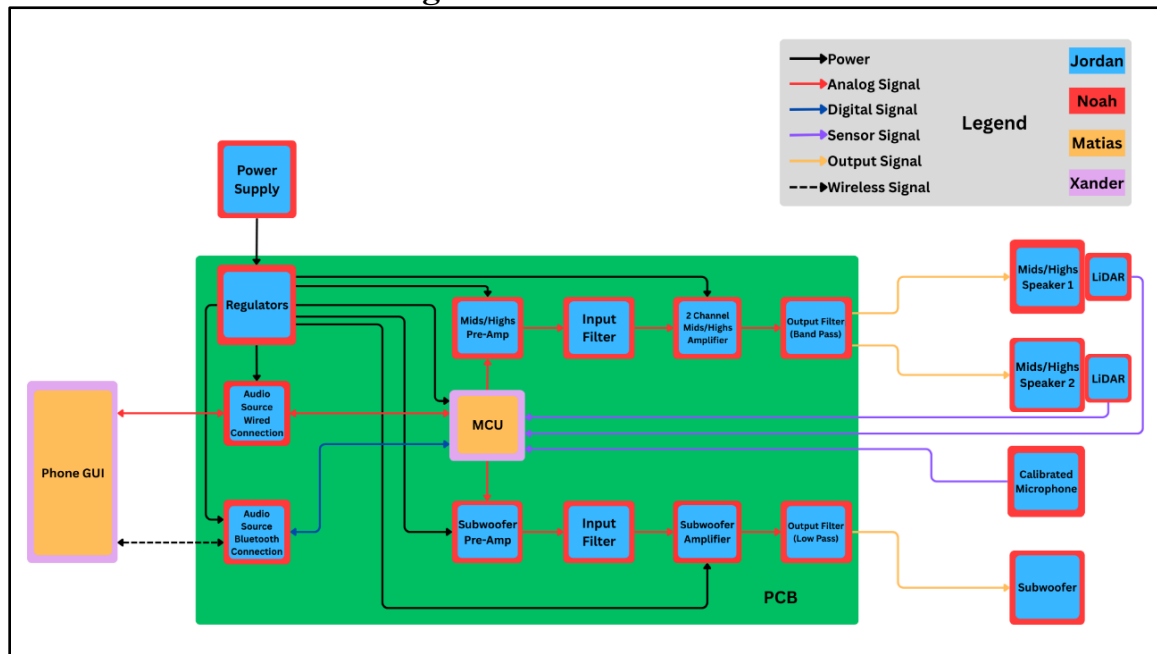


Figure 2.1 Hardware Block Diagram

Pictured above is the hardware system diagram. The diagram illustrates the main hardware components required to process an audio signal from a source and provide an output to a 2.1 stereo system.

The input to the system includes three total sensors, Bluetooth, wired device connectivity, and power. Two sensors are LiDAR, that should be positioned on the speakers to detect the room's dimensions and send the information to the MCU for processing. The third sensor is a calibrated microphone that will record the room's natural acoustics when a sine sweep is played from the speakers. The recorded data is sent to the MCU for processing and filtering of the audio file to create a flat frequency response across the audio band. Additionally, the system will accept either Bluetooth or wired phone connection to communicate with the GUI and send the user's preferred audio to the device. The power supply should supply the power for the entire system from a standard wall outlet.

The system also has five total outputs, three of which are to drive the speakers using the onboard amplifiers. This output should accept speaker wire gauges ranging from 12 to 16-gauge to ensure adequate compatibility with a variety of speaker combinations. The final two outputs are for connectivity to the phone GUI, whether it be Bluetooth or a wired connection. These outputs were also mentioned as inputs as the phone GUI should be able to send and receive data from the system.

On the PCB, there are a variety of subsystems. The MCU is at the heart of the PCB as it oversees receiving commands sent from the phone and three sensors, as well as the reception of audio files from the phone. Additionally, the MCU must process the data sent from all inputs, aside from the power, and execute a DSP algorithm to filter peaks and valleys in the frequency response. The MCU must also be capable of sending the audio, whether digital or analog, to the amplification section on the board.

The amplification section of the board has multiple stages and channels to achieve an efficient, high quality, powerful output. All channels in the amplification section have the same flow but are separated in the hardware diagram as the mids/highs and subwoofer because they are amplifying separate sections of the audio band and with different amplification factors due to the larger power requirements to accurately and effectively reproduce low frequency signals. At the forefront of these sections is the pre-amp, this section ensures the audio signal has a large enough amplitude for the rest of the amplifier to work efficiently. Next, the input filter will be a low pass filter to remove unnecessary signals that cannot be heard by human ears or frequencies that will cause damage to the drivers. The channels' respective amplifier sections do most of the amplification to prepare to drive the speakers. The final stage is responsible for filtering out any unwanted frequencies that passed through the input filter and got amplified in the amplification stage.

2.5.3 Software Flowchart

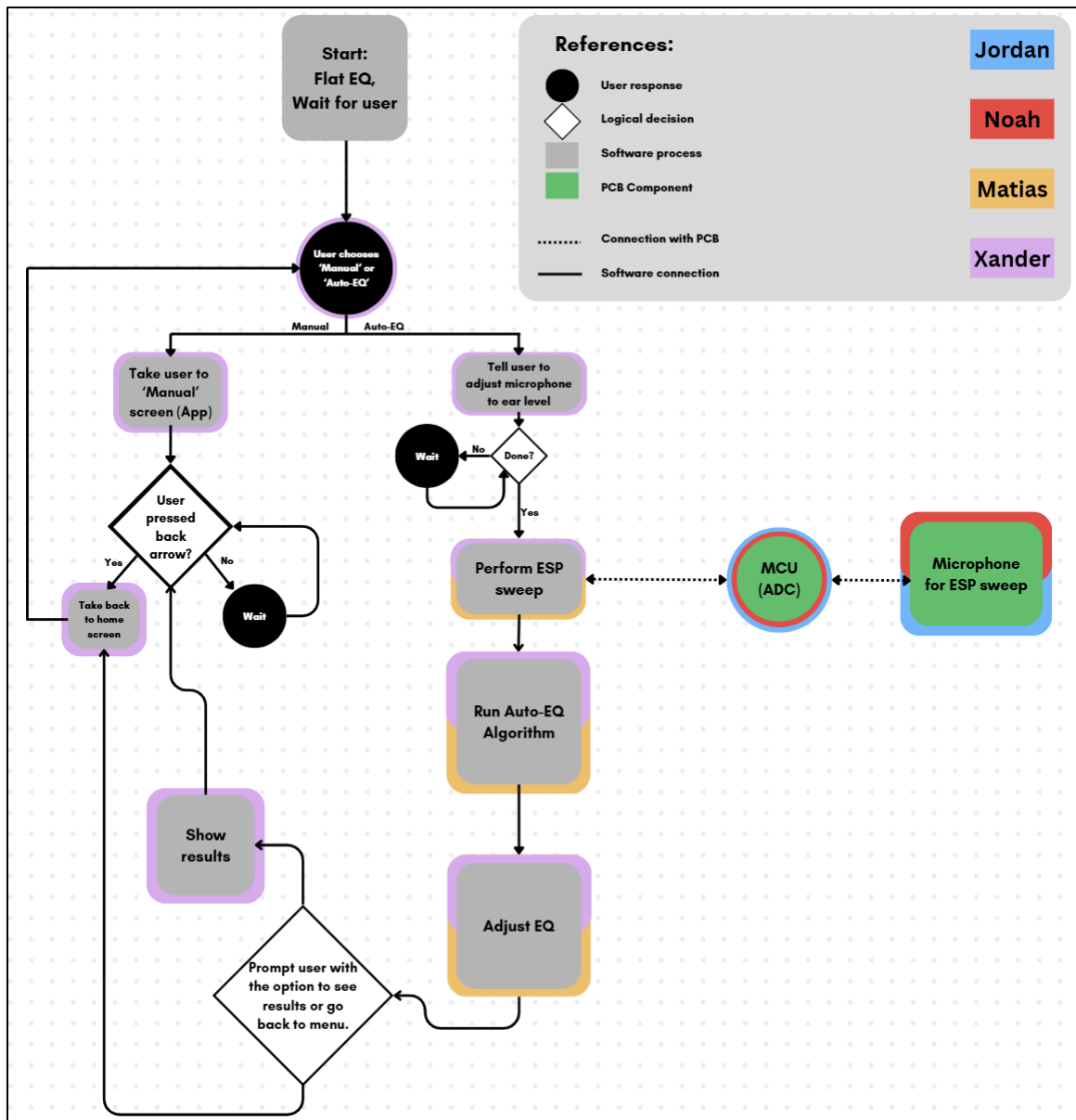
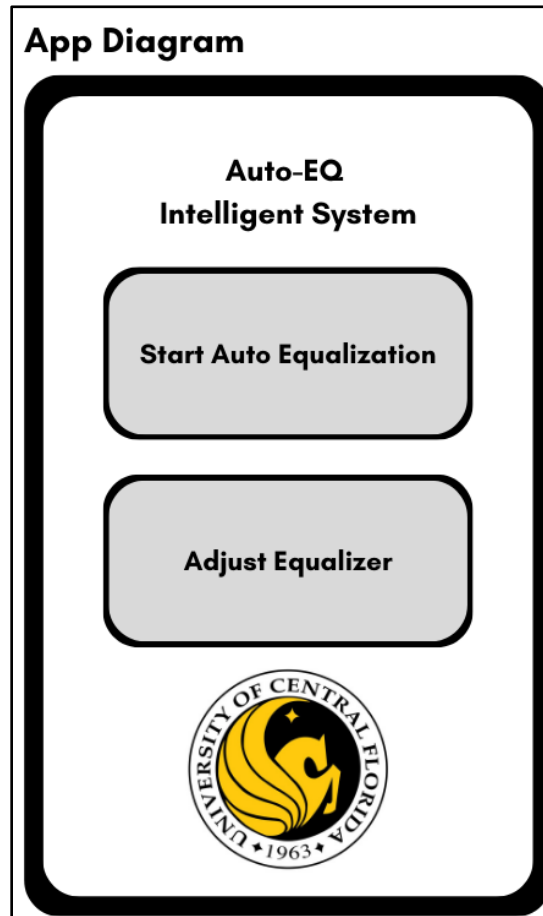


Figure 2.2 Software Flow Chart

The figure above represents the Software Diagram. Here we show the complete software flow to acquire the necessary data to run the Auto-EQ algorithm, as well as the required hardware components.



Above is the User Interface intended to form a nexus between the user and the system. This app gives the user a simple and accessible way to connect and enjoy the Auto-EQ system.

Figure 20.3 User Interface

The software model intends to integrate an app that the user can use to start the system. At first, we load a flat EQ file into that will give us a starting point from where we then adjust the EQ settings when we run the algorithm, or if the user desires to change them manually. This is the first logical decision our flow takes: the user can either do it themselves or let the machine do its job. If the user decides to manually adjust the settings, then we give them the option and wait for them to go back by pressing the back arrow.

If, instead, they choose to run the Auto-EQ, then we first let them know that they need to adjust the calibrated microphone to where they will be listening. We will also add a multi-color LED to indicate it; at this point the LED will be blue. Once they do that, we then acquire the necessary signals (using the ToFs and microphone) to run the algorithm. When the algorithm is running, we will make the LED blink in yellow to let the user know that the system is working. This algorithm will be implemented using the ESP-32's dedicated

ESP-DSP library ⁽⁶⁾, as well as complex mathematical analysis such as Fast Fourier Transforms (FFT), which are still under investigation.

One of our end goals is to give the user information while the system is running. This means, with the use of diagrams such as histograms, being able to show the user what is happening inside the system. We aim to achieve this, however, at the end of the run, so that the user can look deeper into the results. Otherwise, they can choose to go back to the main screen on the app and enjoy their intelligently equalized sound system.

2.5.4 Prototype Illustration

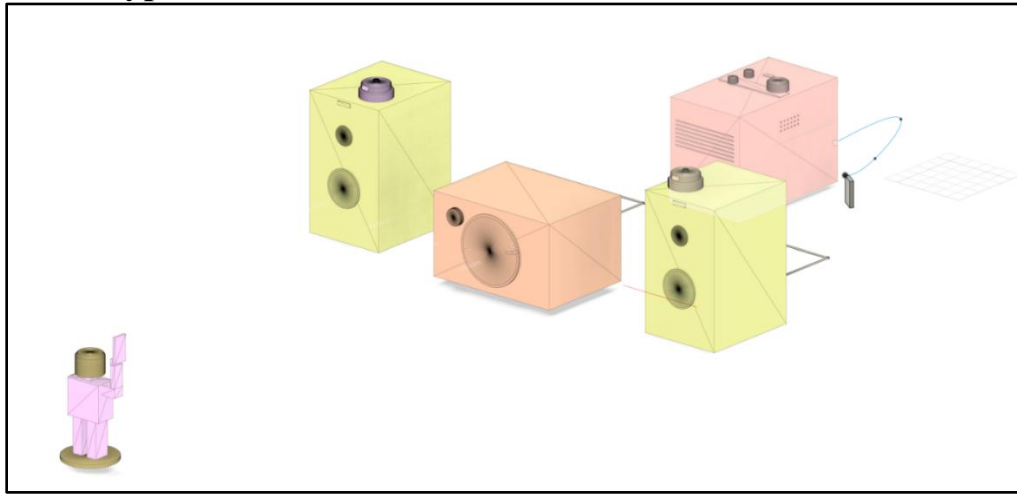


Figure 20.4 System prototype illustration, Angled View

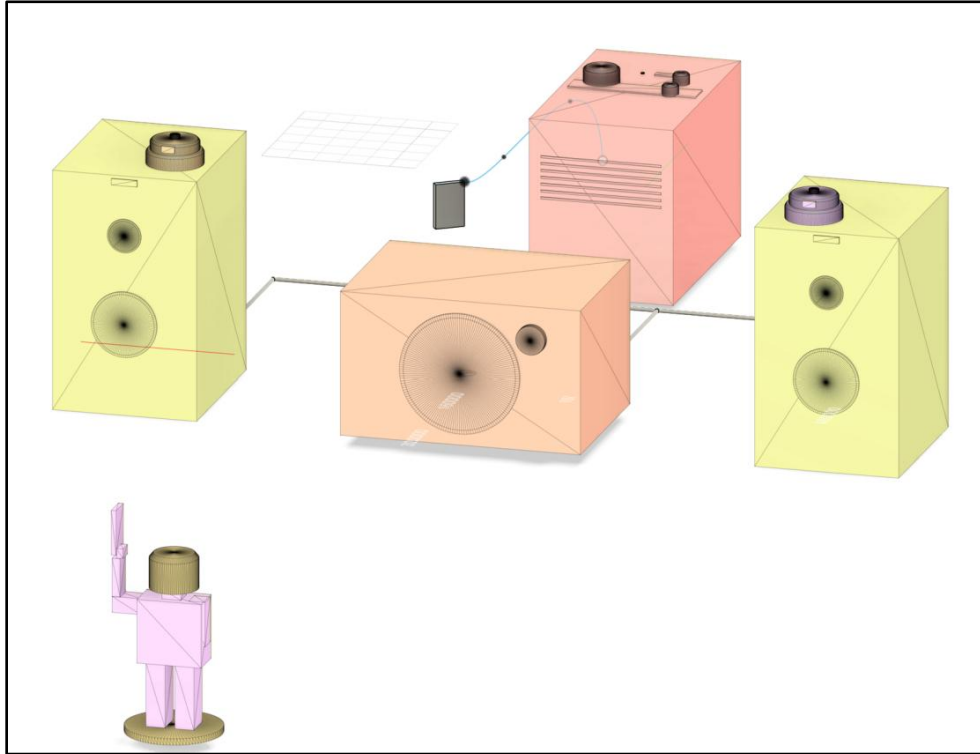


Figure 20.5 System prototype illustration, Front View

	Total Harmonic Distortion -	Capability +	Digital Signal Processing +	Cost -	Calibration Time -	Manufacturability +
Audio Quality +	3	2	3		1	
Ease of Use +		1	2		3	1
Affordability +				3		2
Connectivity +		3	1		1	1
Size -	1	1		1		3

Figure 20.6 House of Quality

3.1 Power Amplifier Technologies

The system has three different types of speakers, midrange, tweeters, and a subwoofer, that require two different amplifier design considerations. The midrange speakers and tweeters

can be driven with the same amplifier, the mid/high amplifier, as they do not require nearly as much driving power as the subwoofer will, however, filtering will be required to send frequencies of roughly 80Hz to 3000Hz to the midrange speakers and frequencies of above about 3000Hz to the tweeters. The mids/highs amplifier will also need to have two channels as there are two sets of midrange speakers and tweeters. The subwoofer amplifier is separate from the mid/high amplifier as the power requirement to accurately produce high quality and loud bass frequencies, less than 80Hz, is much larger than frequencies for the higher frequency bands.

There are many different amplifier topologies that can be used to achieve the goals and objectives of the system. These amplifiers are class A, B, AB, D, integrated circuit (IC) audio power amplifiers, and a combination of IC amplifiers and discrete amplifiers. These amplifiers will be discussed in more detail in the subsequent sections.

3.1.1 Class A Amplifiers

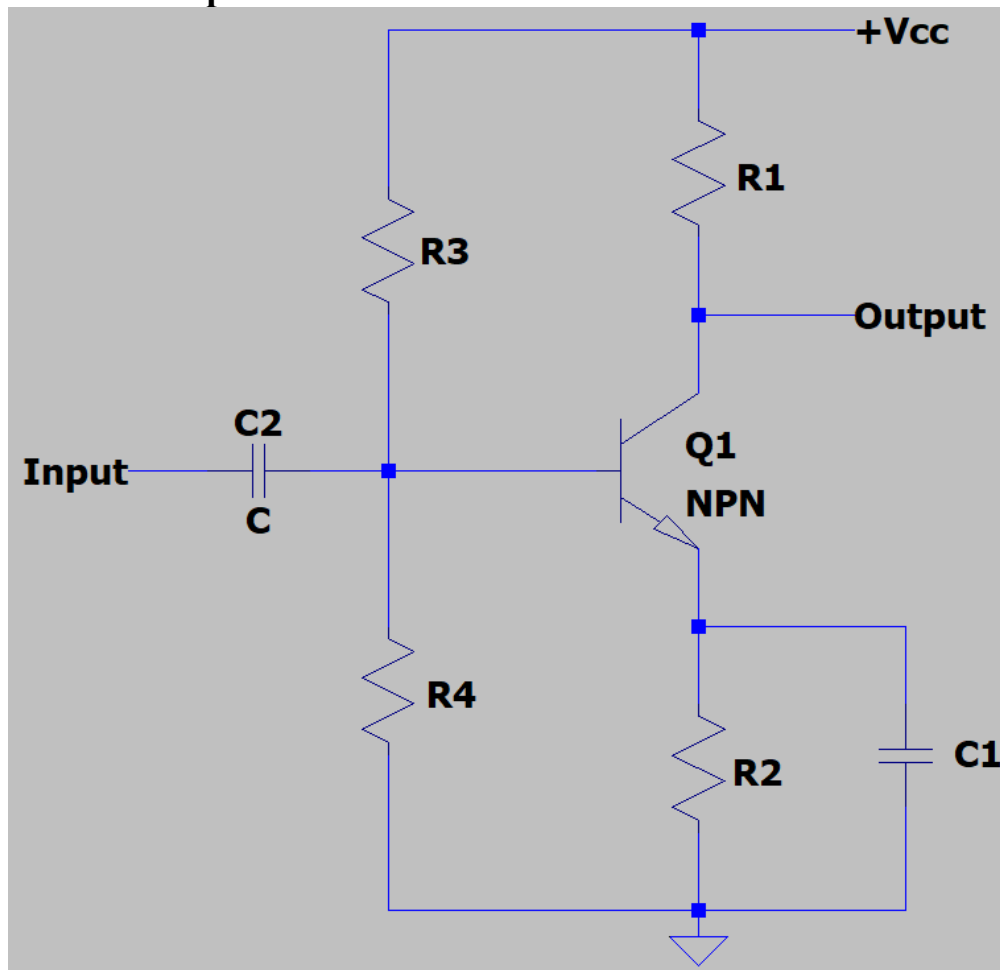


Figure 3.1 Class A Amplifier Topology

The class A amplifier, also known as the common-emitter amplifier, is one of the simplest amplifier designs and is one of the first amplifier designs introduced to UCF ECE students in the required Electronics I course. This amplifier is the foundation to many amplifier topologies so this specific topology will be explained more in depth than the following topologies. The advantages of this amplifier are its linearity, simplicity, and immunity to crossover distortion. The main disadvantage of this amplifier is the inefficiency, which creates a need for large power supplies and heat sinks to power and cool the amplifier. These amplifiers are typically used for radio frequency (RF) and outdoor audio applications [1].

The amplifier works with just one switching transistor that is always powered on and will conduct all 360 degrees of a sinusoidal wave. The transistor in the circuit is biased to a Q-point in the middle of its load line to provide maximum conduction. If the Q-point is not biased in the center of the load line, the amplifier will clip the signal on one side of the wave prematurely and not conduct all 360 degrees of the sine wave.

To center the Q-point on the load line, the load line needs to be modeled by creating a plot with the collector current on the y-axis and the transistor's collector-emitter (drain-source for FETs) voltage on the x-axis. The points where the load line intercepts the x and y-axes are where the saturation points are located. The x-axis intercept is just the rail voltage, while the y-axis intercept is the maximum current that can pass through R_1 . This value is found with the following equation,

$$I_{C(max)} = \frac{V_{CC}}{R_1}$$

The load line is just a linear line between these two points so Q-point should lie on the load line at $(\frac{V_{CC}}{2}, \frac{V_{CC}}{2R_1})$. An example of a load line is shown in the figure below.

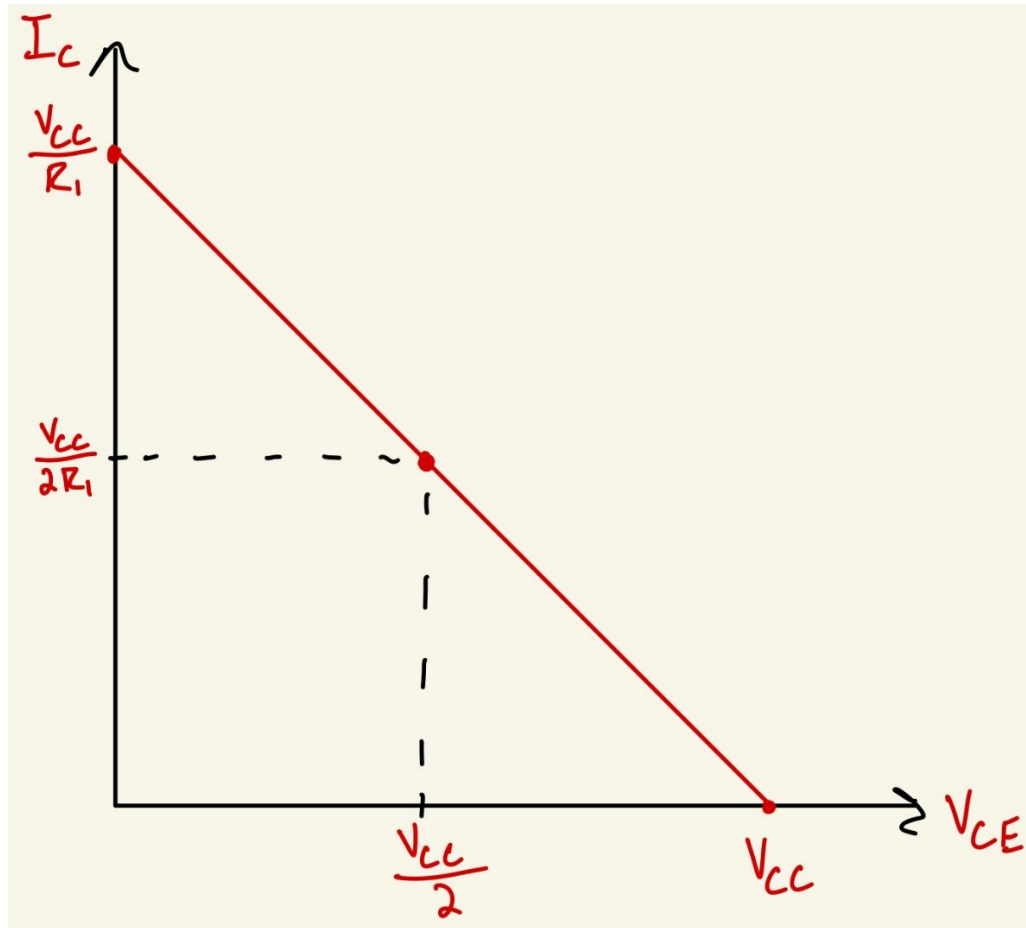


Figure 3.2 Class A Load Line

By knowing the load line and Q-point, the circuit's resistors can be strategically determined to maximize amplifier performance, however, this will not be discussed here as it will be discussed later, in the design section, if a class A amplifier is chosen for the design. It is also important to note here that the Q-point cannot be perfectly centered on the load line due to the emitter resistor and the transistor's turn on voltage.

The load line also illustrates the transistor's ability to conduct the entire sine wave cycle, as the line represents the AC signal. As the signal is input into the transistor the Q-point is moved along the load line and reproduces the input signal at higher amplitudes, with respect to the supply voltage.

The prior discussions explain the benefits of the amplifier; however, they also show the disadvantage of the amplifier. Since the amplifier must be biased at DC to provide maximum conduction, there is a lot of wasted power, resulting in a very inefficient design (around 40% efficiency) [1]. This extra power is wasted as heat and must be actively cooled to prevent the heat from changing the transistor's properties and performance.

What makes this amplifier attractive to use for this project is its linearity and low distortion, resulting in Hi-Fi audio. This will mainly be beneficial in achieving the low THD specification described in Table 2.5.1.

3.1.2 Class B Amplifiers

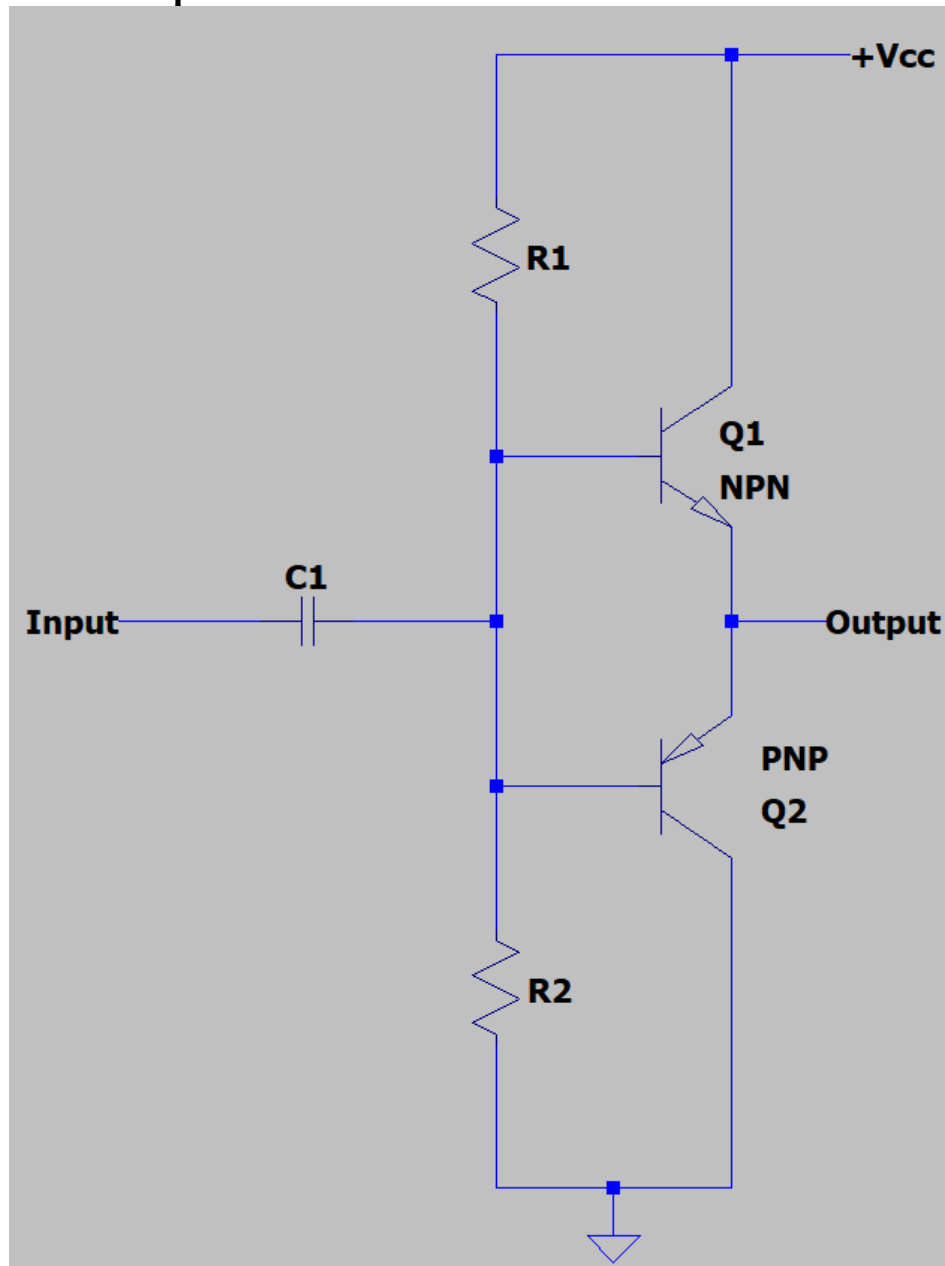


Figure 3.3 Class B Amplifier Topology

The class B amplifier, a push-pull amplifier, is an amplifier that aims to address the inefficiency of class A amplifiers. This is done by having two transistors, each conducting

180 degrees of the input sine wave, instead of one transistor conducting all 360 degrees [2].

This is achieved by using complementary transistors on the output, one NPN and one PNP transistor. The NPN transistor is used to conduct the positive half of the sine wave and the PNP conducts the negative half of the sine wave. This is where the push-pull amplifier name comes from.

This amplifier topology does not require any biasing to reproduce the sine wave. This means that both transistors, when no input is applied, are turned off. In terms of the load line, both transistors' Q-points are set at the x-axis intercept. This shows that there will be no quiescent current flowing through the transistors, resulting in a higher efficiency than the class A amplifier. The theoretical maximum efficiency is 70% [2].

The disadvantage to this amplifier topology is the crossover distortion. This is due to the transistors not being biased; therefore, they require a voltage of 0.7V to overcome the turn on voltage between the base-emitter junction [2].

3.1.3 Class AB Amplifiers

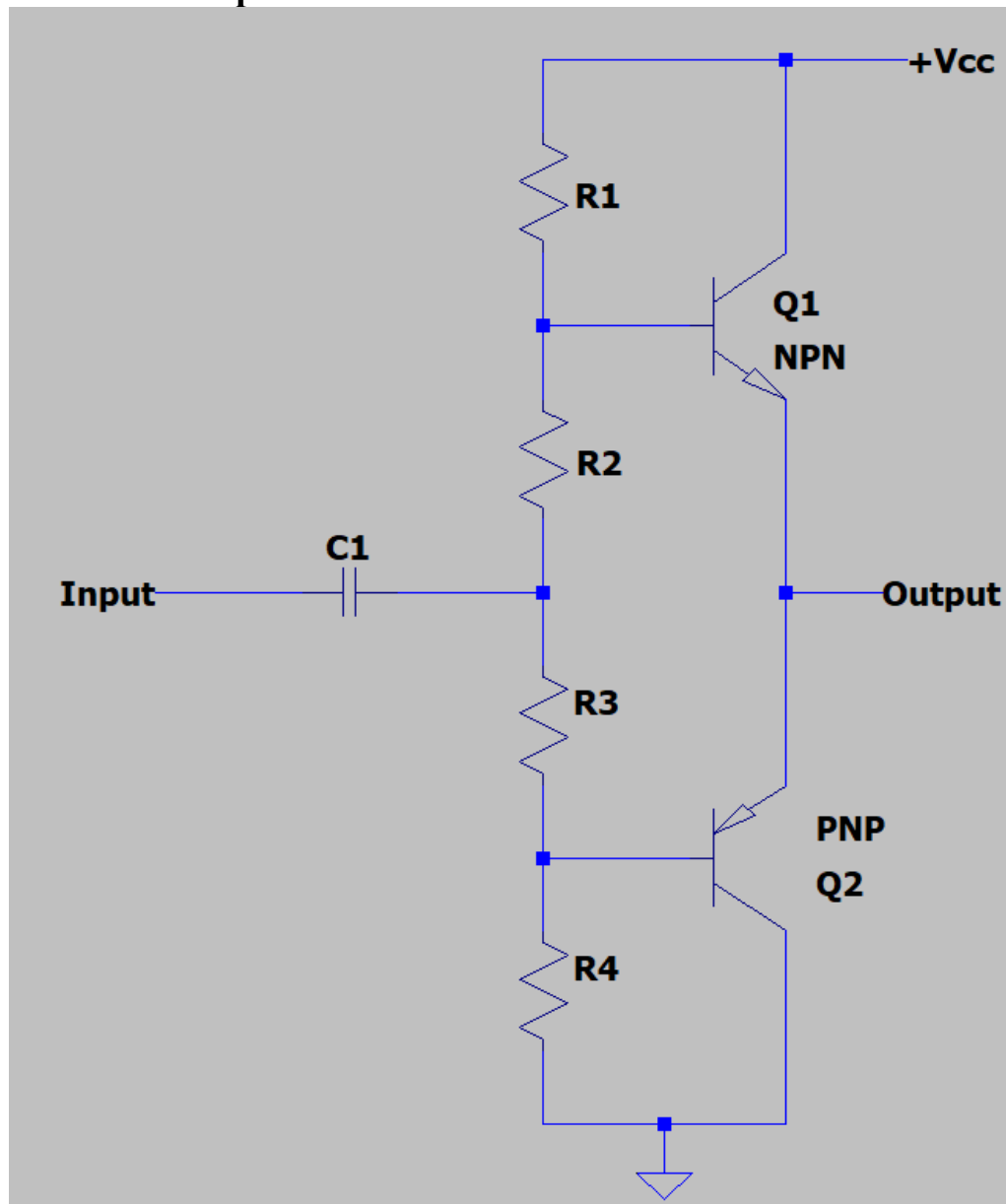


Figure 3.4 Class AB Amplifier Topology

The class AB amplifier, like the class B amplifier, is a push-pull amplifier and is a mix between the class A and class B amplifiers. This amplifier aims to take the efficiency of the class B amplifier, while having the linearity and low distortion of the class A amplifier.

The class AB amplifier topology is very similar to the class B amplifier, but the class AB amplifier has a biasing network between the two complementary transistor bases. This network can be made of voltage sources, voltage dividers (shown above), or diodes. The

purpose of biasing the transistors is to remove the crossover distortion that is prominent in the class B amplifier. The bias of the transistors is such that there are about two diode voltage drops between the two bases. This allows the transistors to conduct at the same time in the crossover region. This also means that both transistors will conduct while under while there is no input, decreasing the efficiency of when compared to the class B amplifier. The biasing only keeps both transistors on for a small region of the sine wave, so once the signal is no longer in the crossover region, only one transistor will conduct. Like the class B amplifier, the NPN transistor conducts the positive half of the wave and the PNP conducts the negative half. This allows for higher efficiency than the class A amplifier.

These amplifiers are typically used as an output stage as the transistors are placed in the emitter-follower mode, meaning they are generally current amplifiers, and the voltage gain is very close to one. For this reason, this topology cannot be used alone if both voltage and current amplification are needed. Many times, these amplifiers are used following a network of largely class A amplifiers, as they can perform low power voltage amplification needed to drive the class AB amplifier to a suitable voltage and the class AB amplifier will provide the current amplification.

The class AB amplifier is very commonly used in many audio applications as they provide a mix of the advantages of both the class A and class B amplifiers making it a very viable option for the system.

3.1.4 Class D Amplifiers

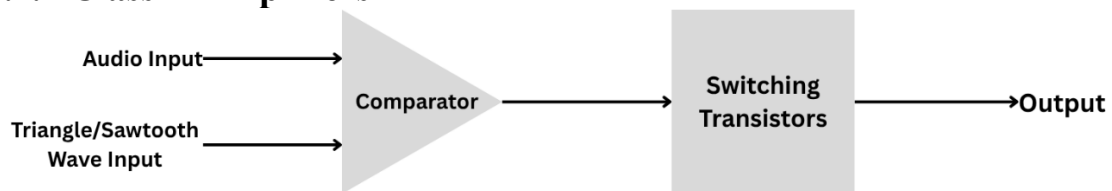


Figure 3.5 Class D Amplifier Block Diagram

The class D amplifier is a relatively new amplifier topology in the audio world as audiophiles have been worried about the high THD that class D amplifiers can create. However, over the years this worry has been minimized through the advancement of transistors and modulation techniques. The class D amplifier is also highly efficient and has a small footprint compared to the other amplifier classifications. For these reasons, class D amplifiers have become a popular choice for amplifier design [3].

Since most audio signals, unless digitized, are analog signals, the signal must be turned into a pulse width modulation (PWM) signal; other modulation techniques are available, but PWM is one of the simplest to implement [3]. This can be generated by a comparator using the audio signal as one input and a triangle/sawtooth wave on the other input. This will create a PWM representation of the audio signal ranging from the comparator's rail voltages.

The amplification is done by switching transistors on and off using the PWM signal. This allows the transistors to conduct little to no current when no input signal is present, resulting in a highly efficient design. Furthermore, when the transistor is conducting, V_{DS} is small, so little power is dissipated in the transistor. The transistors pass nearly all the positive and negative rail voltages to the output [3].

The downsides to the class D amplifier are EMI susceptibility, power supply rejection [PSR], distortion, and speaker pops during turn on/off. Each of these can be minimized with design considerations, but this can add frustration in the design process and create setbacks in the Senior Design flow.

3.1.5 IC Audio Power Amplifiers

IC power amplifiers are a viable option for the design as they offer adequate power output to meet our specifications for both the mids/highs amp and subwoofer amp. These amplifiers are also easy to implement in the system as the transistor design is already complete and integrated on the chip. The biggest task would be to lay out the amplifiers in accordance with their respective data sheets.

These types of amplifiers come in all the previous design topologies discussed in the preceding sections and are readily available to order on DigiKey's website. The available power output of such amplifiers ranges from 4mW to 600W, making them more than capable of driving both the mids/highs and subwoofer drivers.

The issue with these IC power amplifiers is the THD specifications listed in their data sheets. The manufacturers usually list max power output for 10% THD for amplifiers over 25W, which is much larger than the specifications listed in Table 2.5.1. However, the chosen amplifier can be set to a lower power output to meet the specification.

3.1.6 IC and Discrete Combination Amplifiers

In the previous section, IC amplifiers were established as a viable option for the design, with a major tradeoff in the THD specification. One way to overcome this issue is to decrease the gain of the amplifier; however, this may create an issue where the speakers don't get loud enough for the intended room of use. This is where pairing a discrete output stage with the IC may become useful.

The IC amplifier can be used for the preamp and voltage amplification stage (VAS), while the discrete transistors will provide the final current amplification required to meet the power requirements. This will create a signal that has low THD as the IC is not being pushed hard, and the discrete transistors handle most of the strenuous driving capabilities.

The IC choice for this type of amplifier will differ from the pure IC amplifier. This is because the IC will not be responsible for sourcing and sinking large currents; that will be done by the discrete transistors. The chosen IC will only need to provide voltage amplification.

For this type of amplifier, the most viable output topology would be class AB as this topology provides the best combination of efficiency and low distortion when compared to class A and class B topologies. Furthermore, it is more viable than the class D output because the class D requires a PWM signal to drive the MOSFETs. This would require a larger footprint than the class's AB output stage. The idea for this comes from the book listed in Appendix A [4].

3.1.7 Comparisons

Table 3.1 Amplifier Topology Comparisons

Class	A	B	AB	D	IC	IC + Discrete
Efficiency	Very Low	High	Medium	Very High	Varies with Topology	Varies with Topology
THD	Low	Very High	Low - Medium	Low - Medium	Varies (Low is Achievable)	Varies (Low is Achievable)
Footprint	Large	Medium	Large	Small	Very Small	Small
Heatsink	Large	Medium	Large	Small	Varies with Topology	Varies with Topology
Ease of Implementation	Difficult	Difficult	Difficult	Medium	Easy	Medium
Cost	High	Medium	High	Medium	Low	Low

3.1.8 Summary

Since the IC amplifiers are the simplest, cheapest, and abundant, this was chosen for all three amplifiers in the system. DigiKey has a variety of these amplifiers, mostly with class AB and D topologies, which will need to be researched and compared in the following section to choose the best IC available for the design.

3.1.9 Senior Design 2 Implementation

For Senior Design 2, we utilized Class D ICs for our two amplifiers for the subwoofer and the mids/highs speakers.

3.2 Amplifier Comparisons and Selection

3.2.1 Mids/Highs Amplifiers

The mids and highs amplifier must be capable of producing at least 25W of power for both the midrange drivers and tweeters from about 80Hz to 20kHz. There must also be 4 total channels, one for each driver. For example, if the chosen IC has one output channel, then four amplifiers will be needed. If the amplifier has 2 output channels, then two ICs will be needed and so on. Filters will be implemented on the input by the DSP and on the output with LC filters, as the output filters must be nearly lossless to transmit the most power to the speakers. The speakers will be chosen after the amplifier is chosen as the amplifier load, and output power specifications must be known before the speakers can be decided on.

3.2.1.1 TPA3116D2DADR

The TPA3116D2DADR is a class D stereo amplifier made by Texas Instruments (TI). This IC is a two-channel amplifier capable of driving two 4 Ω bridge-tied loads at 50W each (2 x 50W into a 4 Ω BTL load) with a 21V power supply. This is achieved with nearly 90% efficiency, figure 16 of the data sheet, meaning only a small heatsink will be required to cool the device.

The data sheet provides many details to ensure that the device meets our needs. The main points of interest are the THD + Noise (THD+N) vs power output charts, specifically figure 8. Figure 8 shows the THD+N vs output power for a 4 Ω load with a 24V supply at 26dB of gain and ambient conditions. This chart tells us that at 50W output, the typical THD+N value for the IC is less than 0.1% at 1kHz, 6kHz, and 20kHz. This is well within our specifications. However, it is crucial that the output power does not exceed 50W as the THD+N value grows exponentially as output power exceeds this value. Using this orientation, the amplifier can be a great choice in driving both mid/high speakers if each speaker is 4 Ω and a crossover filter is utilized.

Additionally, some features of this IC include multiple switching frequencies to allow the amplifier to be used with another amplifier, a programmable power limiter, overvoltage and undervoltage protection, overtemperature protection, DC-detect, and short circuit with error reporting. These features can be integrated with the MCU to protect the IC under various circumstances that may arise during the system's lifetime.

3.2.1.2 TPA3112D1QPWPRQ1

The TPA3112D1QPWPRQ1 is another class D TI amplifier. This IC is a mono channel amplifier that can drive an 8 Ω load at 25W with a 24V supply at less than 0.1% THD+N. When driving an 8 Ω load, the amplifier has 94% efficiency and does not need a heatsink. The data sheet also lists that the amplifier can drive a 4 Ω load at 20W with a 12V supply at 10% THD+N.

Like the TPA3116D2DADR, TI provides plenty of charts for the IC to understand the typical device characteristics. The main chart of interest is figure 5 as it lists the THD+N

vs output power for the device at 20dB of gain and an 8Ω load with a 24V power supply. The chart shows that the amplifier has less than 0.1% THD+N for all power outputs up to 30W. However, output powers of greater than 25W are thermally limited.

Some of the features of this IC are filter-free operation, adjustable power limiter, pin-to-pin short circuit protection, thermal protection, and four selectable, fixed gain settings.

3.2.1.3 MAX9709ETN+D

The MAX9709ETN+D is a class D amplifier made by Analog Devices. This amplifier can drive 2 x 25W at 8Ω or 1 x 50W at 4Ω . The problem with this is that the specification is made for a THD of 10%, which is far greater than our system specification. The applications listed for this IC are LCD TVs, PDP TVs, and PC/HiFi Audio systems. The temperature range is from -40C to +85C.

Analog devices, like TI, include typical operating characteristics charts in their data sheet. The main chart of interest for comparison and knowing whether this device can meet our design goals is the THD+N vs output power chart for a 4Ω load, as the 8Ω load is already unable to be used for this device as seen on the first page of the data sheet. The THD+N for the 4Ω load is under the 0.5% criteria set for our design up to about 35W output.

3.2.1.4 LM3886 (ChatGPT)

The LM3886 is yet another TI amplifier that can meet our design goals. This amplifier has a class of AB topology, so the efficiency is lower than the other amplifiers discussed. However, the power output is better for both 4Ω and 8Ω loads; 68W and 38W, respectively, with $\pm 28V$ supply. This amplifier also features very good THD+N characteristics as from 20Hz to 20kHz, the typical value is 0.1%.

The downside to this amplifier is that it requires both positive and negative rail, which is not required for the other amplifiers described in the previous sections. Additionally, lower efficiency requires larger heatsinks than the other amplifiers. The features of this amplifier are also fewer than the other amplifiers as this device does not have overvoltage, undervoltage, or thermal protection circuitry.

3.2.1.5 Comparison

Table 3.2 Mids/Highs Amplifier Comparison

Device	Topology	Power (W)	THD	Efficiency	Price
TPA3116D2DADR	Class D	2 x 50W @ 4Ω BTL w/21V	<0.1%	~90%	\$3.10/Unit
TPA3112D1QPWPRQ1	Class D	1 x 25W @ 8Ω w/24V	0.1%	94%	\$2.84/Unit
		1 x 20W @ 4Ω w/12V	10%	Not Considered	
MAX9709ETN+D	Class D	2 x 25W @ 8Ω	10%	Not Considered	\$21.80/Unit
		1 x 35W @ 4Ω	0.5%	~85%	
LM3886	Class AB	1 x 38W @ 8Ω w/±28V	<0.1%	Dissipates ~35W with ±30V supplies	\$7.01/Unit
		1 x 68W @ 4Ω w/±28V	<0.1%	Dissipates ~40W with ±30V supplies	

3.2.1.6 Summary

The TPA3116D2DADR is going to be the best option for our design as it has two channels capable of driving 4Ω loads with lower THD+N than our specification states. Furthermore, the amplifier has the second highest output power that meets our THD+N specification while also being highly efficient. The LM3886 has higher power with better THD+N, however, the heat sink requirements make it a less optimal choice. The LM3886 also only has one channel and does not compare well with the features of the TPA3116D2DADR.

3.2.1.7 Senior Design 2 Implementation

For the Mids/High amplifier, we ended up utilizing the IC mentioned above and found it to work extremely well, when following the datasheets specifications.

3.2.2 Subwoofer Amplifiers

3.2.2.1 TPA3251D2DDVR (ChatGPT)

The TPA3251D2DDVR is a class D amplifier capable of 2 x 140W at 4 Ω BTL, 2 x 175W at 3 Ω BTL, and 1 x 285W at 2 Ω parallel BTL (PBTL), all at 1% THD+N. The typical applications for this amplifier are Blu-ray Disk/DVD Receivers, high end HTiB Systems, AV Receivers, High End Soundbars, mini combo systems, and active speakers and subwoofers. Our system is like active speakers and subwoofers, so our system meets the intended use of this IC.

The data sheet provides plenty of charts for consumers to understand the typical characteristics of the device. The main charts of interest are figures 3 and 16. Figure 3 shows the THD+N vs Output Power for the BTL configuration and shows that the device meets our specification for 8 Ω loads up to 55W, 4 Ω loads up to about 150W, and 3 Ω loads up to 175W. The amplifier is also capable of outputting 100W to the 3 Ω and 4 Ω loads with 0.01% THD+N. Figure 16 shows the THD+N vs Output Power for the PBTL configuration. This chart shows that the THD+N of the device meets our specification for 8 Ω loads up to about 60W, 4 Ω loads up to about 150W, and for 2 Ω loads up to about 250W. The amplifier is also capable of outputting about 125W to a 4 Ω load with about 0.01% THD+N and 200W to a 2 Ω load at about 0.02% THD+N. The PBTL configuration would be ideal for our design as for the subwoofer, only 1 channel is needed, and we can push more power to the subwoofer if a 2 Ω subwoofer is chosen.

The efficiency of this amplifier, given in figure 6, in the BTL configuration, is about 90% for 8 Ω loads, 90% for 4 Ω loads, and 85% for 3 Ω loads at the maximum power we would use. For the PBTL configuration, there are no efficiency charts given, but it can be assumed that the efficiency is greater than 80% since the amplifier has a class D topology.

The features of the TPA3251D2DDVR include multiple configurations, click and pop free startup and stop, undervoltage protection, clipping protection, overtemperature protection, and short circuit protection.

3.2.2.2 TDA7293V

The TDA7293 is a class AB amplifier made by STM. This device is capable of 1 x 100W into an 8 Ω load at 10% THD with a $\pm 40V$ power supply. The datasheet describes the device intended for use in HiFi applications, like home stereos and self-powered loudspeakers.

Figures 14 and 15 of the datasheets illustrate the THD vs Output Power characteristics of the device and show that the IC meets our specification up to about 70W for 4 Ω loads with a $\pm 20V$ supply and about 90W for 8 Ω loads with a $\pm 40V$ supply. As with any class AB amplifier, efficiency is just as important for our design as the THD since the class AB requires significant heatsinking, however, the data sheet does not list efficiency charts to assess the thermal factor but recommends 1.5C/W heatsinking.

The features of this device include clip detection, low distortion, low noise, no switch on/off noise, muting and stand-by functions, and thermal shutdown.

3.2.2.3 TPA3221DDVR

The TPA3221DDVR is a class D amplifier made by TI that can drive 2 x 88W at 4 Ω BTL, 2 x 100W at 3 Ω BTL, and 1 x 170W at 2 Ω PBTL with 1% THD+N. This amplifier can be used at any voltage in the range of 7V to 30V, however, lower voltages result in less power or more THD+N at a specific power level. The intended applications of this IC are Bluetooth and Wi-Fi speakers, soundbars, subwoofers, bookshelf stereo systems, and professional and public address speakers.

In typical TI fashion, plenty of charts are provided to ensure the device can meet our specifications set for the project's design. The main charts of interest are figures 6-1 and 6-17 as both charts plot the typical THD+N for various output powers with a 30V power supply rail. Figure 6-1 shows chart for the BTL configuration with loads of 3 Ω , 4 Ω , and 8 Ω , and the max output power that will meet our design specifications for THD+N is about 100W, 80W, and 45W respectively. The first page of the data sheet lists that these are around the values for 1% THD+N and this is because the region these values are pulled from is exponential, causing some discrepancy in the ability to pinpoint an exact value. Figure 6-17 shows the chart for the PBTL configuration, meaning the amplifier is driving one speaker with both output channels. This chart gives the THD+N spec for loads of 2 Ω , 3 Ω , and 4 Ω , and the max output power that meets our specification for each load is about 150W, 125W, and 100W, respectively. For the 2 Ω and 3 Ω loads, the amplifier can be set to an output power of 100W for less than 0.1% THD.

Furthermore, the efficiency of this amplifier is high since it is a class D. The efficiency of the BTL configuration at the maximum power outputs that would be used for loads of 3 Ω , 4 Ω , and 8 Ω being about 80%, 80%, and 90%, respectively. This information is found in Figure 6-6. For the PBTL configuration, the efficiency for the amplifier at the maximum output would be used for loads of 2 Ω , 3 Ω , and 4 Ω being about 85%, 90% and 90%, respectively. This information is in figure 6-22. With the efficiency being high, only a small heatsink would be required for this IC.

Some of the features of this IC are a closed loop feedback design, standby modes, selectable gains, protection circuitry, and a bottom mounted thermal pad.

3.2.2.4 Comparison

Table 3.3 Subwoofer Amplifier Comparison

Device	Topology	Power (W)	THD	Efficiency	Price
TPA3251D2DDVR	Class D	2 x 55W @ 8Ω BTL	0.5%	90%	\$7.81/Unit
		2 x 100W @ 4Ω BTL	0.01%	90%	
		2 x 100W @ 3Ω BTL	0.01%	85%	
		1 x 60W @ 8Ω PBTTL	0.5%	N/A	
		1 x 125W @ 4Ω PBTTL	0.01%	N/A	
		1 x 200W @ 2Ω PBTTL	0.01%	N/A	
TDA7293	Class AB	1 x 90W @ 8Ω w/ ±40V supply	0.5%	N/A	\$8.49/Unit
		1 x 70W @ 4Ω w/ ±20V supply	0.5%	N/A	
TPA3221DDVR	Class D	1 x 45W @ 8Ω BTL	0.5%	90%	\$4.39/Unit
		1 x 80W @ 4Ω BTL	0.5%	80%	
		1 x 100W @ 3Ω BTL	0.5%	80%	
		1 x 100W @ 4Ω BTL	0.5%	90%	
		1 x 100W @ 3Ω BTL	<0.1%	90%	
		1 x 100W @ 2Ω BTL	<0.1%	85%	

3.2.2.5 Summary

The TPA3251D2DDVR is going to be the best option for our design as the amplifier is capable of large output powers with very little THD+N and high efficiency while not sacrificing the features that come on any of the other amplifiers discussed in the section. The efficiency of the amplifier is also comparable to the other amplifiers, so no sacrifices are made with this choice. With this selection, a subwoofer of 2Ω and 4Ω impedances should be considered with ratings of 125W and 200W, respectively.

3.2.2.6 Senior Design 2 Implementation

For Senior Design 2 we ended up sticking to the Class D IC amplifier we selected in Senior Design 1, the TPA3251D2DDVR. This part was unable to meet our total power output specification; however, we were still able to implement and use the part as planned. The part was unable to meet our power specification for the subwoofer due to input limitations, as the IC required differential inputs and the Op-Amp selected to convert a single-ended signal to a differential signal did not phase shift its negative output by 180-degrees. A way to overcome this issue in the future is to add a follow up op-amp to invert the negative input.

3.3 Speaker Consideration, Comparison, and Selection

3.3.1 Considerations for the High Speaker

For considering our high speaker, our attention is drawn to tweeters as their frequency range and hardware design are built to produce anywhere from 2,000 Hz to 20,000 Hz. However, in our considerations we needed a tweeter that would have a sensitivity of about 92 dB and power handling of anywhere from 45 W and higher. In addition, for the tweeter and mid speaker, which is discussed in a future section, we wanted to keep the cost of each component to be under 30 dollars. Following these criteria, we came across three different tweeter options that would satisfy our requirements and work within our project. The three options that we came across are the Peerless by Tymphany BC255C08-04, Peerless by Tymphany XT25SC40-04, and the Mr.DJ HDT5005.

3.3.2 Comparison for the High Speaker

The Peerless by Tymphany BC255C08-04 comes in at a price tag of \$23.95, has a power handling of 100W, a frequency response range of 2,500 Hz to 20,000 Hz, and a sensitivity of 98 dB. Of the three options, this tweeter is the costliest of our options but still meets all the specs needed to make our project successful. Another note regarding this tweeter and the other model manufactured by this company is that the low end of the frequency range is 2,500 Hz which is still a little high compared to the average tweeter frequency range being 2,000 Hz so when choosing a mid-speaker, we would need to make sure that we are choosing one that has a higher frequency range or equal to the 2,500 Hz.

The next choice for our tweeter is the Peerless by Tymphany XT25SC40-04. With a \$18.51 price tag, 100 W power handling, 2,500 Hz – 20,000 Hz frequency range, and a sensitivity of 91.1 dB, this option produces many admirable qualities when deciding whether we should move forward with this speaker component. In terms of what we are saving in terms of cost in comparison to the BC255C08-04 we are losing in sensitivity with our goal of 92 dB, however we are talking about a .9 difference. However, it is a tradeoff that must not be overlooked. In addition, as mentioned before, we would still need to choose a mid-speaker that can meet or is greater than the 2,500 Hz frequency range.

Our last consideration for tweeters is the Mr. DJ HDT5005, and this speaker comes in at a price tag of \$29.99 for a pack of two, which puts it at the cheapest price of the three options. With 100 W power handling, and sensitivity of 109 dB, this choice presents itself as the

standout for our project. However, the very deterring fact is that the website that offers this product, nor any other sources say what the frequency response is, so we are unable to find out what its lowest and highest frequency capabilities are and therefore we would not be able to pick an appropriate mid-speaker.

Table 3.4 High Speaker Comparison

Speaker Component	Cost	Frequency Range	Power Handling	Sensitivity	Ohm
Peerless by Tymphany BC255C08-04	\$23.95	2,500Hz – 20,000Hz	100 W	98 dB	4
Peerless by Tymphany XT25SC40-04	\$18.51	2,500Hz-20,000Hz	100 W	91.1 dB	4
Mr. DJ HDT5005	\$29.99/2	???	100W	109 dB	4

3.3.3 Selection for the High Speaker

Of all the choices presented, the best suited candidate would be the Peerless by Tymphany BC255C08-04. Although it carries the highest price tag, it still fits within our \$30 budget. In addition, it meets the sensitivity needs for our components and can handle the power provided by our amplifiers with no issues.

3.3.4 Senior Design 2 Implementation

When ordering the drivers to realize our speaker enclosures, the speaker that was selected, BC255C08-04 was unfortunately not available, as a result we chose our second best option which was the XT25SC40-04.

3.3.5 Considerations for the Mid Speaker

For the mid speaker we looked at two different speakers that fall under the mid speaker category: woofers and midranges. Within these two classifications of a mid-speaker, the normal frequency range produced is around 250 Hz to 2000 Hz. Following the criteria presented in the high speaker section, we needed a speaker that had a sensitivity of about 92 dB, power handling of about 45W or higher, and had to be under 30 dollars. Our research led us to these three speakers that we felt were great candidates for our project. The first was the Dayton Audio DC130B-4, the Goldwood GM-65/4, and the Goldwood GM-85/4.

3.3.6 Comparison for the Mid Speaker

The Dayton Audio DC130B-4 costs \$28.98, has a power handling of 40 W to 80 W, a frequency response of 50 Hz to 5000 Hz, and has a sensitivity of 92.1 dB. What makes this option stand out for selection is that it meets the budget, sensitivity, and frequency response requirements set by our project. Although the power handling is rated for between 40 W and 80 W, we also know that most companies are conservative in these ratings as they don't want average consumers to "blow up" their speakers. With this in mind, we can safely say that the DC130B-4 would meet the power rating criteria as well; we would just need to be mindful.

The next option is the Goldwood GM-65/4 which has a price of \$11.98, a power handling of 50 W to 100 W, a sensitivity of 92 dB, and a frequency range of 550 Hz to 15,000 Hz. This is the cheapest of our options and fits very well in line with the power rating in comparison to the Dayton Audio choice. However, like our choice with the tweeter we would need to decide on a low speaker that can compensate for the high, lower end frequency of 550 Hz as most low speakers can only produce up to 200 Hz. Regardless, this option still meets the cost, power rating, and sensitivity requirements.

Our last, but not least option is the Goldwood GM-85/4. This component costs \$15.98, has a power rating of 60 W to 120 W, a 500 Hz to 15,000 Hz frequency range, and a 92 dB sensitivity level. This option is very similar to other Goldwood model; however, it has a higher power rating and covers a larger frequency range. This model would allow us to ensure feel confident that the component would be able to receive the power supplied by the amplifier safely, while also giving us more selection opportunities for our low speaker since it is the closest to the normal 200 Hz of the Goldwood models discussed.

Table 3.5 Mid Speaker Comparison

Speaker Comparison	Cost	Frequency Range	Power Handling	Sensitivity	Ohms
Dayton Audio DC130B-4	\$28.98	50Hz-5000Hz	40W – 80W	92.1 dB	4
Goldwood GM-65/4	\$11.98	550Hz-15,000Hz	50W-100W	92 dB	4
Goldwood GM-85/4	\$15.98	500Hz-15,000Hz	60W-120W	92 dB	4

3.3.7 Selection for the Mid Speaker

Through careful consideration and deliberation, the best choice for our project is the Goldwood GM-85/4 model. By providing us with a safer buffer in terms of power handling and although high, but manageable frequency response, this mid-speaker is more than capable of delivering on all the aspects needed to help this project succeed. By choosing

this mid-speaker it also allows extra room in our team's budget to fund the tweeter we chose since it is a lot closer to our \$30 budget cap for speaker component selections.

3.3.8 Senior Design 2 Implementation

Unlike the tweeter we were able to order and successfully use the speaker we chose above for our project.

3.3.9 Considerations for the Low Speaker

When considering our low speaker, we needed to change some of the technical specifications we had in comparison to our mid and high speaker selections. First off, due to our amplifier design we were able to look at low speakers that had a 2-ohm tolerance as opposed to the 4-ohm tolerance that we were forced to use when deciding on the high and mid-speakers. With this, we had to adjust the power rating of the speakers using the 2-ohm low speakers, which led us to a new rating of 200W. However, when dealing with 4-ohm low speakers we would need to use 125W. In addition, our budget for the low speaker was expanded to components that were up to \$100 or less, compared to the \$30 previously decided upon for the mids and highs. When deciding on what low speaker, or subwoofer, we should choose our criteria listed from the most important to least important are as follows, impedance, power handling, sensitivity, and frequency response. After following these criteria, we found three speakers that would fit our design are, the Goldwood GW-212/4, the Goldwood GW-10PC-4, and the Kicker 43CVT102.

3.3.10 Comparison of the Low Speaker

The Goldwood GW-10PC-4 comes in at a price tag of \$79.98, with a range of 25 Hz to 3000 Hz frequency response, a 92.7 dB sensitivity, and 200 W to 400 W power handling level. As a 4-ohm speaker we meet the requirements in terms of power handling by a wide 75 W margin which makes it easier to implement and not worry about power issues. The other nice aspect of this subwoofer is that it has 92.7 dB sensitivity, which is the highest of the three options, plus, at a \$79.98 price we are saving \$20.02 in budget costs. With a range of 25 Hz to 3000 Hz we also avoid any gaps in frequency between the mid and low, as we decided on a mid-speaker with a lower frequency of 500 Hz.

Our next option is the Goldwood GW-212/4. With a price tag of \$42.98, 120 W to 240 W power handling, 28 Hz to 4000 Hz frequency range, and a sensitivity of 92.3 dB. Cheaper than the GW-10PC-4, this component also has an extended 997 Hz frequency range. However, one of the downsides compared to the previous Goldwood model is that we are losing approximately .4 dB in sensitivity. Another drawback of this Goldwood model is that the power handling level again relies on the generosity of the manufacturers. For a 4-ohm speaker, which this component is, we need a minimum power level rating of 125 W, and this specific model has a level minimum level 120 W, however, it has a maximum power level of 240 W. This could lead to issues if we find the speaker is performing at a power level too low for our design.

Our final consideration for the subwoofer needed to complete the speaker design is the Kicker 43CVT102. This component costs \$99.95, a 350 W to 700 W power handling, a 25 Hz to 500 Hz frequency range, and 87.2 dB sensitivity. This specific speaker consideration is a 2-ohm, which indicates we would need to ensure that the Kicker meets the power handle level of 200 W which it does with a 150W margin. In addition, what makes this speaker a great candidate is that it does reach the lower-level frequency set by the mid-speaker we chose of 500 Hz. What makes this speaker not the best option though is its sensitivity level. With a sensitivity of 87.2 dB, it is well below the 92 dB goal set for each of our speakers, and with the heftiest price tag of the three options the tradeoff is not worth the price. Unfortunately, this speaker does not size up very well compared to the other two Goldwood options.

Table 3.6 Low Speaker Comparison

Speaker Component	Cost	Frequency Range	Power Handling	Sensitivity	Ohms
Kicker 43CVT102	\$99.95	25Hz – 500Hz	350W – 700W	87.2 dB	2
Goldwood GW-212/4	\$42.98	28Hz – 4000Hz	120W – 240W	92.3 dB	4
Goldwood GW-10PC-4	\$79.98	25Hz – 3000Hz	200W – 400W	92.7 dB	4

3.3.11 Selection for the Low Speaker

After considering all three options, we found that the Goldwood GW-10PC-4 was the best option for implementation. With its wide margin of power handling, budget-friendly pricing, and excellent sensitivity this product really stood out amongst the others. However, when it comes to implementation if this model didn't seem to work, we have considered the Goldwood GW-212/4 as a back-up option.

3.3.12 Senior Design 2 Implementation

For the subwoofer driver we were able to order and successfully implement the component discussed above for our selection.

3.4 Speaker Enclosure Design

The design of the speaker enclosure is just as important, if not more important, than the driver selection. This is because the enclosure controls the air movement dynamics when the speaker is moving. This creates different sound profiles from the speaker. The enclosure can also be used to raise the mechanical resistance of a speaker, or it can be used to minimize the mechanical resistance. Furthermore, the enclosure acts as a passive filter for the speaker drivers. This section will be used to discuss the different types of enclosures and the choices we make for our specific drivers.

3.4.1 Importance of Speaker Characteristics

The goal of enclosure design is to maximize speaker performance and to achieve this, the speakers Thiele/Small (TS) Parameters must be known. The TS parameters is a “series of characteristics based on a model of electrical and mechanical attributes of individual loudspeaker drivers” organized by Neville Thiele and Richard Small, independent of one another, that collectively describe the properties of a loudspeaker [5]. One of the most important parameters is the Q factor, a sum of the electrical damping, Q_{es} , mechanical damping, Q_{ms} , and the total damping, Q_{ts} , where the Q_{ts} describes the type of enclosure a speaker will thrive in [5]. Q_{ts} of less than 0.6 benefit from ported enclosures, and a sealed enclosure is best suited for Q_{ts} of between 0.6 and 1.0 [5].

3.4.2 Open Baffle

The open baffle technically is not an enclosure at all as the speaker is mounted to just a flat board and no surrounding parts. This type of enclosure maximizes the distance the rear sound waves must travel to interfere with the sound waves coming from the front of the speaker cone [5].

3.4.3 Sealed Enclosure

Sealed enclosures are exactly what the name suggests; an enclosure completely closed off from external air. All forward air is allowed to pass to the listeners while the rear air stays inside the enclosure and does not interfere with any front sound waves. This type of enclosure acts as a 2nd Order high pass filter at the enclosure’s resonant frequency. These enclosures are common in all types of speaker drivers, especially tweeters and midrange speakers. The downside to this is that more power is required to produce low frequencies at the same SPL as higher frequencies [5].

3.4.4 Ported Enclosure

Ported Enclosures utilize a vent leading from the inside of the enclosure to the outside. This allows for better frequency response below both the speaker driver’s and enclosure’s resonant frequency [5]. This effect is achieved by allowing the rear sound waves to be released from the enclosure and mixing with the sound waves produced by the front of the speaker's cone. The enclosure volume, port length, and port area all play a role in the tuning frequency of this enclosure [5]. The port parameters must be designed carefully as air oscillates in this section of the enclosure and can cause distortion if the air traveling inside the port does not move cleanly around the edges [5]. This can be minimized by not using sharp edges in the port and using slanted or curved sections where necessary. These enclosures are a 4th order high pass filter in nature with the attenuation beginning at the enclosure’s resonant frequency.

3.4.5 Bandpass Subwoofer Enclosures

Bandpass enclosures are mainly used for subwoofers and are highly efficient but must be meticulously crafted as any defects can cause distortion. The enclosures are very common in car audio SPL competitions for bass and, as the name describes, they are bandpass filters

by nature with various orders available to the designer. These enclosures allow for very high-power levels to be pushed to the subwoofers.

The 4th order bandpass enclosure is designed with both a sealed section and ported section. The sealed section is used to recreate the lower frequencies, giving the subwoofer a punchy sound, and the ported section is responsible for the higher frequencies.

Another type of design used is the 6th order bandpass enclosure. This utilizes a similar design to the 4th order; however, the sealed section is vented into the ported section to create a more efficient enclosure.

3.4.6 Enclosure Design Comparison

Table 3.7 Enclosure Design Comparison

Enclosure Design	Open Baffle	Sealed	Ported	Bandpass
Difficulty	--	--	+	++
Low End Response	--	-	+	++
Filter Type	None	2 nd Order High Pass	4 th Order High Pass	4 th Order and Higher Bandpass
Size	--	-	+	++

3.4.7 Enclosure Material

The most common materials for speaker enclosures are plastic and wood, specifically MDF. The benefits of wood are that it is durable, lightweight, and acoustically sound. Woodworking has also been around for some time so finding resources to craft the perfect enclosure should be easy. Plastic has the benefit of being cheaper, easier to build with 3D printers, and more resistant to the outside elements, however, they may take a few design cycles to get the material thickness correct. Also, it may be difficult to find a 3D printer capable of supporting our enclosure size.

Table 3.8 Material Comparison Table

Material	Wood	Plastic
Ease of Manufacturing	+	-
Resources	+	+
Availability	+	-
Durability	+	-
External Factors	-	+
Acoustics	+	-

3.4.8 Summary

For our design it is apparent that using wood as our enclosure material is the best option. For the enclosure design, a sealed enclosure should work well with the midrange driver and tweeters, while a ported design is going to optimize the low frequency response of our subwoofer so we can play louder bass frequencies. Pairs of midrange drivers and tweeters will be in the same enclosure, each section sealed from each other, for a total of two enclosures for the mids/highs. The subwoofer will be in its own enclosure.

3.4.9 Senior Design 2 Implementation

For both our mids/high enclosure and the subwoofer enclosure we made them out of wood as noted above as we found that it provided the best sound quality and safety for our speakers. We ended using a sealed enclosure for the subwoofer as making the enclosure ported would have driven up the cost for the project, along with the original design not considering the thickness of the material.

3.5 Crossover Filters

3.5.1 What are Crossover Filters?

When discussing crossover filters regarding the performance and construction of our system, it is important to ask yourself what a crossover filter is. Crossover filters are either high, low, or bandpass filters that are used to individualize the type of frequency that is being sent through either our high, low, or mid speaker. To further clarify, we would utilize a high pass filter for a high speaker since this type of speaker performs best when exposed to higher frequencies. Furthermore, bandpass is used for mids, and low pass filters are used for subwoofers. [6][7]

Now in terms of the applications of these filters, there are two distinct paths: multi-way cabinets and multi-cabinet setups. Multi-way cabinets describe when there are at least two speaker components within a cabinet, while multi-cabinet refers to having a specific cabinet for each speaker component. For our design we are incorporating both by having our high and mid-speakers enclosed in one cabinet while our low speakers will be enclosed in a separate cabinet.[6][7]

3.5.2 Comparison of Passive, Hybrid, Active Crossover Filters

When dealing with crossover filters, there are three main and distinctive types, passive, hybrid, and active. What separates the three different filters is where the crossover is placed regarding the amplifier and the speakers themselves. The placement of the crossover determines the type of input signal the crossover receives whether it be line-level (active), speaker level (passive), or a mix of both (hybrid). [6][7]

Passive crossovers shine in their easy implementation, cost effectiveness, and low power usage. By placing the crossover between the amplifier and the individual speakers, we are receiving a speaker level signal strength which is higher than the normal line level input seen by the MCU and in this case the amplifier as well. Essentially what this means is that

the filter does not need a power source to work. However, some of the drawbacks of passive crossover include limited frequency accuracy, power loss due to the heat in the electrical components that make up the filter, no tunability unless component changes occur, the fact that it is affected by the speaker's impedance which changes while being used, and the increased difficulty of phase alignment. [6][7]

Active crossovers are known for their high flexibility, efficiency, frequency accuracy, and independence of speaker impedance. These types of filters are normally implemented in high performance, professional setups. However, when looking at the scope of the system we are trying to create, the downside of an active crossover is the complexity of the design and the cost associated with it. [6][7]

This is where the hybrid crossover comes in. What makes the hybrid stand out in terms of selection for our project is that it allows us to take the positive traits of both the active and passive crossovers and join them into one. The hybrid allows for tunability, a proper complexity level for the scope of this project, well enough frequency accuracy, minimized effect of speaker impedance, minimized power loss, and seems to be the best cost/performance option of the three. [6][7]

3.5.3 Cross Over Filter for Mid and High Speakers

Now that we have established the type of network we are using for our crossover filter, we need to distinguish the traits or specs of each filter we are going to use for our different speakers. As mentioned before, our high and mid-speakers will be enclosed in the same cabinet. So, for this case we will need to implement a high pass and band pass filter to separate the two frequencies. By placing the crossover filter after the amplifier but before the signal reaches the two speakers, we can separate the high pass filter response created from the MCU into another high pass filter so the high speaker can function properly, but also creating a band pass with cutoff frequencies equaling the max frequency of which the low speaker can function properly (lower cutoff frequency) and the max frequency of which the mid speaker can function properly (higher cutoff frequency). [6][7]

3.5.4 Crossover Filter for Low Speaker

For our low speakers, we are taking a very similar approach to that of mid and high speakers. Since the MCU will be splitting our initial signal into two components, a low frequency and mid/high frequency range, we would be sending the low frequency signal produced by the MCU straight to the subwoofer, however, on one condition. To ensure that our speakers are playing frequencies that are well suited and tailored to the level of quality we are aiming to create we felt that passing the signal through a crossover filter to “fine tune” the signal, something that was mentioned as a bonus for choosing the hybrid crossover network, would allow us to create a more specific frequency cutoff for the woofer, thus allowing better frequency accuracy. [6][7]

3.5.5 Senior Design 2 Implementation

To implement the crossover filter for senior design 2, we utilized the ESP32 to act as an active filter to separate the mids/highs frequencies from the frequencies the subwoofer required. For the mids/highs crossover filters, we utilized a car audio crossover, or bass blocker, that was available to separate the frequencies we sent to the midrange driver and the tweeter as without a crossover the tweeter would be overloaded by the mid-bass frequencies and cause the tweeter to break.

3.6 Distance Sensor

3.6.1 Considerations

The distance sensor will play the crucial role of measuring the distance from our system to the room's ends. The information gathered from the sensor(s) will not only help us to determine the size of the room and how that affects the sound, but also other factors such as reverberation or echo. The main specification that we are looking for in our sensors is the maximum distance at which it can be measured. This is crucial because we want our system to work in any space of any dimension. Of course, there are many other factors that we should be considering creating the best sound experience for the user, like the material of the walls, or if the room is filled with furniture or other large objects that could block sound waves, but these are out of the scope of this project due to the already complexity of project and the time constraints.

Since we want to measure all dimensions (length, width, and height) we also need to consider if we want many different sensors in our system (two for each dimension would be needed, because the system won't be placed in one corner) or if we want one single sensor that can rotate or move to measure different distances. If we go for the latter, we will need to implement more software to calculate the precise distance from the sensor to the walls or roof. On the other hand, if we decide to integrate many sensors, we will just need to add the distances that each sensor gathers, but this could increase the cost of the build, depending on the type of sensor we choose.

We have summarized all our options in Table 3.x.1, where the maximum measuring distance, accuracy, type, and price of each are listed. All the following options are ToF (Time of Flight) types of sensors, but each uses a different technology. The first option is the SEN0647 which can measure up to 25m (~82ft) and is a Laser sensor. The main advantage of this sensor is the astonishing distance it can measure up to; it is one of the few options that we found that can measure up to that distance. The same model has another version that can go twice as far, measuring up to 50m (~164ft) for only four dollars more. [25] The second option is a 360 LiDAR (DFR0315) that can measure up to 12m (~39ft) in diameter, or 6m in each direction. This option would save us from buying multiple sensors to achieve different measurements, but the module itself is considerably bigger than the rest of the options. This one is also the most expensive on the list, but it offers other great features such as very high accuracy: it only has a $\pm 0.5\text{mm}$ deviation. [26] The third option is another Laser sensor from STMicroelectronics: the VL53L4CX. Although it offers the least maximum distance, it also comes at a reasonable price and in a very compact module

(4.4 x 2.4 x 1 mm) making it the smallest in the list and the easiest to integrate into our system, as our MCU is also an STM product. [27] Our last option is the LIDAR-Lite LED V4 which, despite its name, is an LED sensor. This differs from the other options in that it utilizes an infrared LED, which is safer for the human eye than the sensors, which use lasers. It also comes with other benefits such as Bluetooth integration to use wirelessly, an ARM Cortex -M4 processor, and much more. However, we do not intend to use these for our project, as we would already have all these features in our MCU. We just need the sensor to measure distance.

Table 3.9 Distance Sensor Comparison

	Max. Distance (m)	Accuracy	Type	Price
SEN0647/ SEN0648	25/50	± 3cm	Laser	\$25.00 / 29.00
DFR0315	12	± 0.5mm	360 LiDAR	\$99.00
VL53L4CX	6	± 3%	Laser (w/ FOV)	\$14.95
LIDAR-Lite LED V4	10	± 1cm	LED Sensor	\$59.95

3.6.2 Conclusion

When we started this project our idea with the distance sensors was to use them to aid our acoustic measurements of the room. Our intuition told us that by getting the room's dimensions, we were going to be able to gather valuable information to design our algorithm. We later realized that this is not entirely true. Even though the room's dimensions can be useful for some analysis, it would not give us more information than what the microphone will. These are usually used when implementing a "Room Simulation" environment, which is used to "artificially reproduce the acoustics of a room." [67] This is helpful if we wanted to predict how the audio signals will behave in certain rooms and maybe find some dead spots, but the reality is that it is out of the scope of this project and it does not add valuable features for our design. Therefore, we decided to remove the distance sensors from our design.

3.7 Microphones

3.7.1 Speaking Microphone vs. Audio Measurement Microphone

For our system to equalize based on the room's environment we will need a microphone to measure the acoustics of the location the speaker system is placed in. Knowing this, we need to distinguish what type of microphone we need since a standard talking microphone is not going to work in this case. This is where a condenser measuring microphone comes into play. Essentially this speaker is going to capture the flat frequency response of the speaker's location and will allow the system to equalize based on the measurements the microphone sends back to the system. The reason we would not use a speaking microphone

in this case is that speaking microphones lack the sensitivity and accuracy needed to detect the sound pressure of a given room. Within the measurement microphone technology there are different classes that provide individual advantages based on a given situation. [8]

3.7.1.1 Class 1 Measurement Microphones

The most accurate of the classes in, these microphones are designed to meet high standards and are mostly used for research and lab situations. The strictness of the frequency response and dynamic range is where this class finds its usefulness. [8]

3.7.1.2 Class 2 Measurement Microphones

Although not as accurate as class 1, these microphones are used in workplace environments such as schools, and industrial settings. They meet the requirements for compliance and legal checks; however, they are not accurate enough for research and testing scenarios. [8]

3.7.1.3 Omnidirectional Measurement Microphones

These microphones are designed to capture sound measurements from 360 degrees, making them suitable for room acoustics and environment sound capture. [8]

3.7.1.4 Audio Measurement Microphones

As the name suggests, these microphones are designed to help with sound system calibration and speaker testing. These microphones are mostly used for sound design, music, and broadcasting, as they excel in reproducing audio characteristics [8]

3.7.2 Considerations for System Microphone

When researching different measuring microphones that would be used within our project, we came across four different options that we feel would be easy to integrate and provide the necessary results to ensure proper acoustic tuning for our speakers. These options are the Superflux ECM999, miniDSP UMIK-1, Dayton Audio Li, and the Behringer ECM8000.

3.7.3 Comparison of System Microphones

The first microphone that we looked at was the Superflux ECM999. This microphone costs \$49.99, has a sensitivity of 37 dB, a SNR of 77dB, frequency range of 20,000 Hz, and is omnidirectional. This option is very budget friendly, however, has very low sensitivity that we feel would need to be improved upon. It has a very wide frequency range, and decent max SPL at 132 dB so it would be able to handle loud sound sources without clipping.

Going to a more high-end microphone we have the miniDSP UMIK-1. With the highest price point of \$128.95, a sensitivity of 39.88 dB, SNR of 78 dB, 20,000 Hz frequency range, and is omnidirectional. This microphone also includes a calibration file which allows for the microphone's frequency and sensitivity to be adjusted in measurement software. The microphone also features a plug and play USB interface, allowing for plug-ins on laptops.

Our next option is the Dayton Audio EMM-6. With a price of \$59.98 this microphone's features include 40 dB sensitivity, 70 dB SNR, 20,000 Hz frequency, and is also omnidirectional. This speaker has been known to provide an extremely flat frequency, and like the miniDSP UMIK-1 features an exclusive calibration file to aid in accurate measurements. Instead of using a USB output connector like the miniDSP UMIK-1 this microphone uses an XLR connector. The EMM-6, however, does feature a lower SPL than some of its counter parts.

The last contender of the 4 is the Behringer ECM8000. With a \$35.00 price tag, it is the cheapest of the bunch and has the following specs of 60 dB sensitivity, 35 dB SNR, 20,000 Hz frequency, and has an omnidirectional polar pattern. The most budget-friendly option does have a few drawbacks that prove it might not be as good of a choice as some of the other discussed microphones. Between the Superflux ECM99 and this option it does have the lowest signal to noise ratio, or SNR. The Behringer does feature a frequency range that does hold up compared to the microphone options though.

Table 3.10 Microphone Comparison Chart

Microphone Component	Cost	Sensitivity	Signal to Noise Ratio	Frequency Range
Superflux ECM99	\$49.99	37dB	77dB	20,000Hz
miniDSP UMIK-1	\$128.95	39dB	78dB	20,000Hz
Dayton Audio EMM-6	\$59.98	40dB	70dB	20,000Hz
Behringer ECM 8000	\$35.00	60 dB	35dB	20,000Hz

3.7.4 Selection for System Microphone

After careful consideration, the microphone we felt would best encapsulate the needs we have for this project, while still staying at a reasonable price point was the Dayton Audio EMM-6. For the cost of this component, we are getting a lot of value and features that other higher end speakers such as the miniDSP UMIK-1 have, such as the calibration file. Furthermore, we are not sacrificing any quality issues as the frequency response, SNR, and sensitivity already fit within the ranges we need so no further adjustments would need to be made to ensure this component would work.

3.7.5 Senior Design 2 Implementation

For the microphone that was used to measure the audio signal from the speakers we did end up implementing the EMM-6 which was the speaker we chose in our initial design.

3.7.5.1 Phantom Power for the Dayton Audio EMM-6

Condenser microphones, such as the EMM-6, are normally extremely sensitive and have a diaphragm next to a metal plate with varying capacitances as the diaphragm vibrates. A pre-amp lowers and amplifies this capacitance leading to a need for more power, which is where phantom power comes into play. Phantom power is a way of providing voltage through current, in our case a current through an XLR cable. The standard of for phantom power voltage can range from 11 to 52 volts, however, our microphone only needs anywhere from 15 to 48 volts.

3.8 Processor Technology Comparisons

The brain behind every electronic system is the processor; it is the component that glues everything together, as it can perform different tasks within its core, as well as organize the other components on the system. For our design, we intend to implement the processor as the heart behind the DSP algorithm, one of the most important software components in our system. It will be responsible for gathering the input data (microphone and sensors), processing the sound signals gathered using DSP concepts (like FFTs, deconvolution, etc.), adjusting the Equalizer's bands based on the DSP processing, and outputting the processed signals to our speakers. Let's look at the different types of technologies that we could use to achieve this.

3.8.1 SoC (DSP Chip)

A System on Chip is a type of integrated circuit that implements all the computer's components (such as microprocessors, memory, WI-FI, Bluetooth, etc.) in one chip. These types of processors are usually seen in devices like smartphones, tablets, desktop computers. [9] They are usually in the high-end spectrum of Embedded projects, as they are a lot more capable and powerful than a microprocessor. In the case of our project, we would be looking into a DSP chip, which are designed to specifically help with DSP related work. This would give us a great opportunity to implement a processor that would speed up the signal processing stage. However, these processors usually require more advanced programming, are more expensive, and we would still need to add different modules to achieve all we want in our system. These are usually intended for high-end systems, which is somewhat off the scope of this project.

3.8.2 MCU

Probably the most common choice for most projects, a Microcontroller Unit is a type of processor that is designed to serve one specific task without requiring a complex operating system. [10] They are usually used to implement "simple" projects, such as real-time systems, robots, or most electronic appliances (dishwashers, microwaves, coolers). They give the user the flexibility of choosing where to implement it, as they can be programmed with an object-oriented programming language (C/C++, Python) without the need for an operating system. But it's in their simplicity where MCUs usually fail to deliver very high performance for some projects, as they are built with smaller components than a SoC.

3.8.3 FPGA

The last technology we considered implementing is a Field Programmable Gate Array. Unlike the previous two, FPGAs are integrated circuits that are re-programmable, meaning that they do not necessarily need to serve a limited number of tasks. They run with the advantage of offering huge flexibility when it comes to realizing a digital system, as they can be programmed for one task and later be reprogrammed to serve a different one. [11] Their main players are LUTs, which are a type of programmable logic that can implement different logic functions based on the number of inputs. These LUTs are interconnected with each other with flip-flops and other digital circuits that allow the FPGA to restructure itself. The great advantage of FPGAs is that they offer great throughput thanks to this technology. The downside of using one is that they are usually the most expensive in the list, and become tedious to program, as they must be programmed with an HDL (Verilog, VHDL, SystemVerilog).

3.8.4 Processor Comparison and Selection

The following table summarizes the main advantages and disadvantages of using each different processor based on the scope of the project.

Table 3.11 Processor Technology Comparison

Specification	SoC	MCU	FPGA
Processing Power	High	Moderate-Low	High
Throughput	High	Low	Very High
Programming	Difficult	Moderate	Very Difficult
Project Fit	Moderate	Very good	Low
Price	Moderate	Low	High

After analyzing our options, we will be implementing an MCU in our design. We think that, despite being behind in terms of computational power when compared to an FPGA or a SoC, an MCU will give us enough power to achieve our goals. There are many options available, which will be narrowed down and discussed in future sections, but we think many should offer what we are looking for.

3.8.5 MCU Comparisons

The MCU is most likely the most crucial component on our board. It is the part that connects everything together and coordinates them to make the project possible. The first criteria for choosing the right MCU will be the type of chip it possesses. For this project, DSP capabilities are one of the most important aspects as we will be processing audio signals at high speeds. We will be using techniques such as FFTs (Fast Fourier Transforms), FIRs, or IIRs, which require heavy computation in real time. It will also need to handle incoming data, as we will implement distance sensors to measure the room's dimensions and use a calibrated microphone to capture the natural response to build a base case for easier equalization setting.

A second set of criteria we will need to consider are ADC and DAC capabilities. Modern MCUs usually come with built-in converters, especially ADCs, but these may not be suitable for work the board will go through. Therefore, depending on the MCU we choose, we need to consider adding external modules with higher resolution to make sure that the signal transmission is easily achievable. The most important task they will need to perform is strongly related to converting the incoming signals captured by the microphone and delivering the processed signal to the speakers/subwoofer.

One final, and still very important, consideration is Bluetooth integration. To make the app possible, we will need a chip that can include Bluetooth capabilities to transmit the button presses the user makes in the app and to display the information in real time. This will also be needed to make the experience more comfortable for the user, as it will allow them to play their music wirelessly.

3.8.5.1 ESP-32

The ESP-32 was probably the first choice that we thought of when we started this project. At its very competitive price, it offers strong performance as well as ease of use and integration with other electronic components. The S3 runs at 240 MHz, making it fast enough for DSP and audio implementations. Some of its variants offer up to 384 KB of ROM and up to 512 KB of SRAM. We can also increase both the storage and memory thanks to its multiple-SPI technology; only one chip (ESP32-S3FH4R2) offers both enough flash memory and SRAM. [12]

What makes the ESP-32 so popular is its availability and performance in comparison to its price: the basic ESP-32S3 sits at just \$1.85 (as of October 2025, DigiKey) and offers enough power to deliver good results. A better option in terms of storage would be the ESP32-S3FH4R2, which costs \$4.14, about \$2 more than the base S3. In terms of DSP capabilities, the chip does not possess an integrated chip or dedicated core for this, but it does support a set of functions in the form of a library that can speed up most mathematical operations. It also utilizes a SIMD (Single Instruction Multiple Data) architecture, which “allows a single instruction to process multiple data elements simultaneously.” [13] The S3 utilizes the Xtensa LX7 microprocessor, which “adds a series of extended instruction set in order to improve the operation efficiency of specific AI and DSP algorithms.” [14] These instructions can be optimized using assembly code, but the Espressif family implemented these optimizations in their dedicated DSP library. This library contains functions, modules, classes, and components that make DSP related work run in a more efficient and faster manner. They provided a table [15] comparing different ESP MCUs when used for many of the functions provided by the DSP library. If we look at the example below, we can see many columns: for our project, we are focusing on the ESP32, ESP32S3 (the MCUs that we are considering), and the ANSI Xtensa (the Microprocessor inside these MCUs) columns. The ANSI Xtensa provides benchmark values (in number of total cycles) for the different functions when not optimized, whereas ESP32 columns are for the optimized (for speed) versions for their respective MCU. We can clearly see the advantage of the ESP32-S3 over the traditional ESP32, but even more clearly when compared to the Xtensa column.

For an FIR filter, when optimizing the ESP32-S3 it took only 32.7% of the time it took the non-optimized version to complete the task (See *Appendix B*, (2) for table). This gives us a clear view of how we could take advantage of these optimizations to speed up our algorithm. [15]

The constraint that we find here is that these optimizations are software-only oriented, as the Xtensa LX7 is not purely dedicated to DSP related work. We are aware that the ESP-32 is generally oriented to more basic projects, which could make our implementation slower in contrast to if we were to use a more DSP-friendly MCU.

We said at the introduction to this comparison that ADC and DAC resolutions are vital to achieve high-quality audio processing. The ESP32-S3 line offers 2 12-bit SAR ADCs and no embedded DAC. This is a clear disadvantage that the S3 suffers from, as we are aiming to find a chip that can deliver almost all the features that we are looking for. As for the ADC, the 12-bit resolution would pose a possible bottleneck when converting the incoming signals captured by the calibrated microphone because even though we could work with that, for the purposes of our project, 16-bit would be a better choice. This would allow us to have a better representation of the analog signal as well. Finally, we will consider using 24-bit ADCs, but this would depend on the storage availability on our MCU. Theoretically, it should give an even better, almost desirable, representation of the signal, but the amount of data to be stored would also increase significantly. As for the DAC, none of the S3 variants come with a built-in D/A converter, meaning we would need to get a separate one. This would not only increase the size of our board but also may lead to more integration in terms of software. Luckily, the ESP-32 offers I2S communication that makes this process a lot more manageable.

A final note on the ESP32-S3 worth noticing, and most likely its big advantage over the rest of its competitors, is that it comes with a built-in Bluetooth antenna. The main reason we wanted to consider the ESP was because of its Bluetooth capabilities, which enabled us to make the MCU-to-app connection a lot easier. We want to offer the user the choice of connecting to the system from the comfort of their seat, so Bluetooth is a must for this reason. For similar reasons as DAC, we would not need to worry about adding external modules that would potentially increase the size of our board.

3.8.5.2 STM32H7xG

The second choice on our list of MCUs is the STM32H7xG family. A more DSP-related MCU, the STM32H7 comes with an ARM Cortex -M7 which offers a double-precision FPU processor that makes mathematical functions run a lot faster than a fixed-point processor (like the ESP32-S3) or even a single-precision FPU. It runs at a maximum clock frequency of 550 MHz (more than twice the speed of the ESP32-S3) and offers 1 MB of flash memory and a total of 564 KB of SRAM. The microcontroller also offers different mathematical accelerators like CORDIC or FMAC, which aim to optimize several mathematical functions like trigonometric functions for a faster DSP application. [16]

This MCU comes at a less affordable price than the ESP32-S3 at a minimum of \$12.35 and can go up to \$14.6 (as of October 2025, DigiKey) but offers much more performance. The tradeoff between the different variants comes down to the number of I/O ports which we are not worried about. Diving into the DSP side of the matter, the STM32H7 offers two hardware units that are designed to accelerate mathematical operations such as sine, cosine, arctan, vector multiplication, etc.: CORDIC and FMAC. The first one stands for Coordinated Rotation Digital Computer and uses a “successive approximation algorithm designed for efficiently computing trigonometric, hyperbolic, and logarithmic functions.” [17] The main advantage of CORDIC is that it is implemented as a coprocessor that can run in parallel to the main CPU, alleviating the workload of the latter. It was designed on a fixed-point unit architecture, which means it can run at a faster rate than a floating-point unit, but with less flexibility. The second accelerator is the FMAC which stands for Filter Mathematical Accelerator which “performs arithmetic operations on vectors”. [16] The FMAC is very useful when it comes to DSP because it can allow us to implement IIR or FIR filters and run them in parallel to the main core, speeding up the process of our algorithm. [18] Turning to the software side, ARM-based microprocessors inherit a library of functions called CMSIS DSP that implements common signal processing functions [19] that could be used to build our algorithm. However, for the STM32, we will mainly focus on exploiting hardware accelerators.

Finally, if we were to choose the STM32H7 as our MCU, we would need to consider adding a separate module to provide Bluetooth connectivity, as it does not come with built-in BT or Wi-Fi modules. We have been looking for modules to integrate with the STM32H7 and, thankfully, STM has many options to make this possible; however, this will be further discussed in a different chapter.

3.8.5.3 NXP i.MX RT1170

Our final consideration for our MCU is the RT1170 from NXP Semiconductors. This is the best DSP suited MCU of all our options, as it possesses an ARM Cortex -M7 as its main processor running at a stunning 1 GHz and an ARM Cortex -M4 as a secondary coprocessor running at 400 MHz. The MCU also comes with a total of 2 MB on-chip RAM, which should be more than enough memory for our purposes, but can still be extended. [20] This MCU is less available than the ESP32 or the STM32 and is selling for \$25.82 (111 In-Stock pieces as of October 2025, in DigiKey), making it the most expensive out of the three.

Having the ARM Cortex -M7 means that it shares the same advantages as the STM32H7, but the i.MX MCU does not come with CORDIC or FMAC on board, meaning that any sort of optimization for DSP workload would be directly tied to software, specifically the CMSIS-DSP library dedicated for ARM processors. But the advantage of this MCU is that we could leave the heavier tasks to the -M7 processor and let the -M4 processor handle everything else. The RT1170 comes with two 12-bit ADC and one 12-bit DAC, so adding external converters should be a consideration if we choose this MCU.

In terms of Bluetooth capabilities, the RT1170 stands on the same side as the STM32H7: an external module would be required to achieve wireless connectivity.

3.8.5.4 ESP32-S3-WROOM-1U-N16R8

As a part of the ESP32-S3 family, this model contains features that will help make this project successful. Since it shares the same processor as the standard version of the ESP32-S3 it has the same max frequency. Since it follows the same trend as the ESP32-S3 series it does include the same environment for DSP integration as the standard version as well. However, regarding the different versions of the ESP32-S3-WROOM-1U models the reason why we chose this specific model was because it provided the most ROM, Flash memory, and SRAM. These values are 384 KB, 16 MB, and 8 MB respectively. In addition, this specific model does include a 2 x 12-bit ADC which supplements the microphone usage we will need for acoustic measurement. Finally, unlike the standard ESP32-S3, it meets the minimum pitch spacing requirement needed to solder efficiently.

3.8.6 Summary & Selection

The following table summarizes the different MCUs we have described in detail in the previous sections.

Table 3.12 MCU Comparison Summary

Category	ESP32-S3	STM32H735ZGT6	i.MX NXP RT1170	ESP32-S3-WROOM-1U-N16R8
Processor	Tensilica Xtensa LX7	Arm Cortex - M7	Arm Cortex - M7, Arm Cortex -M4	Tensilica Xtensa LX7
Frequency (max)	240 MHz	550 MHz	1 GHz, 400 MHz	240 MHz
Bluetooth, non-BLE	Yes	No (Module Req.)	No (Module Req.)	No (Module Req.)
DSP	ESP-DSP Library	CORDIC, FMAC, CMSIS library	CMSIS library	ESP-DSP Library
ADC	2 x 12-bit	2 x 16-bit, 1 x 12-bit	2 x 12-bit	2 x 12-bit
DAC	No	2 x 12-bit	1 x 12-bit	No
Programming	C/C++, Assembly	C/C++	C/C++	C/C++, Assembly
Price	\$ 1.85 - 4.14	\$ 12.35 - 14.60	\$ 25.82	\$6.00 - 6.56

After careful consideration, we decided that the ESP32-S3-WROOM-1U-N16R8 would be the best option moving forward for our MCU. Although it doesn't contain the necessary

Bluetooth standard we need, as well as a DAC, these peripherals can be easily implemented to our board. Furthermore, the DSP library is sufficient to create the algorithm needed for equalization. Although not a deciding factor, its 6.56 price tag makes it the second cheapest option.

3.8.7 Senior Design 2 Implementation

In Senior Design 2, we implemented the ESP32-S3-WROOM-1U-N16R8 as the brain of our system, to control all digital components, send commands, and process all incoming digital signals.

3.9 Phone Communication with MCU

Our system has two means of communication with the mobile app, Bluetooth and USB connection. The idea behind this is to give users the freedom of choice to play music in the two most popular connection modalities. Bluetooth modality gives users freedom to play music from a distance while keeping their phone nearby. The USB connection allows users to play higher quality music, such as lossless audio. The USB connection also provides a backup option in the case that the Bluetooth connection fails, since USB connection is hardwired.

3.9.1 Communication Technologies

There are multiple ways we can communicate with the MCU from our mobile app. Of these choices the main three that stand out are Wi-Fi, Bluetooth, and a hardwired connection. The hardwired connection is something the group has unanimously agreed upon as it offers a fast and reliable connection to the system, furthermore, the hardwired connections allow for lossless audio to be sent to the system, the highest quality audio available for music streaming. The real investigation is between a Wi-Fi connection and a Bluetooth connection.

The foundation of Wi-Fi technologies lies in the domain of radio frequency (RF) signals. With RF signals, two or more devices can connect to the same internet and share information with each other wirelessly. The benefits of Wi-Fi compared to Bluetooth are the data transfer rate, as Wi-Fi has a large bandwidth, range, compatibility with other devices connected to the internet, and security [21]. The challenging part of utilizing Wi-Fi would be finding a way to get the MCU connected to Wi-Fi. The ESP32 would be a great option to implement this function as it already has Wi-Fi capabilities, however, since we have chosen the ESP32-S3-WROOM-1U-N16R8 for our design, an external module would be required, this ends up being the downside to choosing Wi-Fi. Furthermore, even after setting up a Wi-Fi module, we would need to find a way to connect the device to the internet without using Bluetooth.

Bluetooth is another RF technology that is widely used in many devices today. Audio has a huge role in Bluetooth, as many audio devices already use Bluetooth to play a user's music. The benefits of this over WiFi are the ease of implementation, power consumption, and compatibility with our specific project [22], as Bluetooth offers a direct connection,

whereas on Wi-Fi, the devices must communicate through an access point [22]. The ease of implementation comes from the idea that the system can have a button on it to set it to pair mode, and we would not find a way to connect the device to anything other than a phone. Again, the ESP32 would make this easy to implement as it has a Bluetooth module on the chip, however, we can settle for an external module as the MCU can communicate to it.

Table 3.13 Bluetooth vs Wi-Fi Comparison

Specification	Bluetooth	Wi-Fi
Range	-	+
Data Transfer Rate	-	+
Power Consumption	+	-
Ease of Implementation	+	-
Security	-	+
Connection	Direct	Indirect

Although Wi-Fi seems like a better choice in terms of range, data transfer rate, and security, Bluetooth is the option we are going with because of the ease of implementation and power consumption. Bluetooth is also more than capable of meeting our transfer rate goals as it's already used widely in audio applications.

3.9.2 Bluetooth Connection

3.9.2.1 Technology Comparison

Expanding on the topic of Bluetooth, started in the previous section, a few key parts must be understood. The first part is that Bluetooth sends data over the frequency band from 2.4 GHz to 2.4835 GHz [22]. The second is that there are two types of Bluetooth, Bluetooth Low Energy (BLE) and Classic Bluetooth [Source]. These two types must be compared to determine our part. Another part that must be understood is that Bluetooth utilizes frequency hopping [23]. The final part is that Bluetooth data is sent in packets [23].

By using the 2.4 GHz to 2.4835 GHz range, Bluetooth is susceptible to interference as many other wireless electronics work on this frequency band, such as Wi-Fi [23]. This means that precautions must be taken to ensure that the interference is minimized. This is achieved naturally in the communication protocol with frequency hopping [23]. Frequency hopping is the process of switching the communication frequency used pseudo randomly. This switch is driven by the master device in the connection, the user's phone in our system, and happens in between the transmission and reception of packets.

Packets in Bluetooth are groups of data that are sent with each other. The Packet contents and size vary by Bluetooth technology and transmission goal. These packets can be used to connect two devices or send data back and forth to devices that are already connected.

The differences between BLE and Classic Bluetooth lie in the channel scheme, power consumption, latency, and data transfer; Classic Bluetooth also has two different types, Basic Rate (BR) and Enhanced Data Rate (EDR) [24]. Channel Scheme is the least important to our design so discussion on this topic is limited.

In BLE, the idea is to minimize power consumption and latency, while using less communication channels and decreasing data transfer rates. BLE also reserves three channels for discovery allowing for easier connectivity. To minimize output power, BLE utilizes a low output power of 10mW to transmit data and is sent into an idle state when transmission is not required [24]. In terms of latency, BLE only needs about 3ms to be ready to send data packets. Where it falls short is the data transfer rate at about 0.27 Mbps [24], far too small for audio applications.

Classic Bluetooth has the opposite tradeoffs. This form sacrifices latency and power consumption for more communication channels and higher data transfer rates. The latency of such devices is about 200ms total between when we know the data needs to be transferred to the reception of the data and the power required to transmit data is about 100mW, with no idle states [24]. However, Bluetooth BR and EDR support data transfer rates up to 0.7 Mbps and 2.1 Mbps, respectively [24]. This means that EDR can transmit audio signals.

A newer Bluetooth technology, Bluetooth 5, aims to meet the needs of many different designers and merge the benefits of both BLE and Classic Bluetooth. Bluetooth 5 is a technology that incorporates the low power benefit from BLE while allowing designers to cater to the technology to range or speed [24]. In terms of our design, we are more worried about the data transfer rate and Bluetooth 5 can achieve data transfer rates of up to 2 Mbps, keeping up with Bluetooth EDR [24]. This technology is also more energy efficient than Classic Bluetooth as it builds on BLE. The latency of this technology is at worst 40 ms and 20 ms under ideal conditions [25].

Table 3.14 Bluetooth Technology Comparison

Specification	Bluetooth BR	Bluetooth EDR	BLE	Bluetooth 5
Connectivity	-	-	+	+
Power	-	-	+	+
Latency	-	-	++	+
Data Transfer Rate	-	+	-	+

After reviewing the different Bluetooth technologies, it is apparent that Bluetooth 5 is going to be our best option, but Bluetooth EDR can also be used since we are not worried about latency, because our device is designed to play music and not audio for a home

theater, and power requirements as we will be connecting to a wall outlet. Since Bluetooth 5 is better in all categories compared to Bluetooth EDR, Bluetooth 5 will be the focus, however, if a Bluetooth EDR module is better suited for our design, we will not pass on the opportunity to use the technology. Furthermore, the Bluetooth module we choose must have advanced audio distribution profile (A2DP), aptX, or Advanced Audio Coding (AAC) to be able to stream music to the system [26].

3.9.2.2 Component Comparison

The ESP32-S3-WROOM-1U-N16R8 that we selected as our MCU does not include a Bluetooth module on the chip, as a result, an external module must be chosen to be placed on the PCB. There is an abundance of such modules made by various companies that are readily available for order on DigiKey. The modules we have chosen to investigate are the BM83 by Microchip, BM64 by Microchip, and FSC-BT1026D by Feasycom. The latter two were found using ChatGPT to research Bluetooth modules with audio capabilities without AI was extensive with little to no results.

3.9.2.2.1 BM83

The BM83 is a Bluetooth module, qualified to Bluetooth v5.0, specifically designed for audio applications. This module features noise reduction and echo cancellation built in. Additionally, the device uses I2C or I2S to transmit the audio signal from the phone to the MCU and UART to allow the MCU to control the device. The specifications of the module are as follows, up to +11 dBm transmission power, -90 dBm receiver sensitivity for 2 Mbps EDR transfer, and an integrated antenna.

3.9.2.2.2 BM64

The BM64 is another Bluetooth module specifically designed for multi-speaker audio applications. The module is qualified to the Bluetooth v5.0 specification and supports A2DP. The device features noise suppression as well as echo cancellation. To communicate with the MCU the BM64 uses UART and to transmit the audio signal I2C or I2S is used. Additionally, the device can be configured with a Windows GUI or by the MCU. The specifications for the device are up to 15 dBm typical transmit power, up to -90 dBm receiver sensitivity for 2 Mbps EDR data transfer rate, and an integrated antenna. It is apparent that the BM64 and BM83 are very similar parts.

3.9.2.2.3 FSC-BT1026D

The FSC-BT1026 is the final Bluetooth module that we investigated for our design. This device, unlike the previous two discussed, meets the Bluetooth 5.1 specification and is an extremely powerful module as it can support speakers on its own. Instead of using the A2DP protocol for audio handling the FSC-BT1026 uses the aptX technology, which offers higher quality audio streaming. This aptX is handled by a Qualcomm audio DSP. The module is capable of up to +9 dBm transmission power, -100 dB BLE receiver sensitivity, 3 Mbps transfer rates, and an integrated antenna. The device communicates using UART, I2C, SPI, and USB 2.0. There are far more features for this module that are going untouched.

3.9.2.2.4 Comparison

Table 3.15 Bluetooth Module Comparison

Specification	BM83	BM64	FSC-BT1026
Transmission Power (dBm)	+11	+15	+9
Receiver Sensitivity (dBm)	-90	-90	-100
Transfer Rate (Mbps)	2	2	3
Antenna	PCB	PCB	PCB or External
Capabilities	Supports our design	Supports our design	Extremely overkill
Price	\$12.20	\$18.13	\$14.46

3.9.2.2.5 Summary

While the FSC-BT1026 looks like the best option in the comparison, the features not discussed are extensive and we do not need most of them. For that reason, the BM83 is going to be used in our design. This was chosen over the BM64 because the BM83 is the successor to the BM64. Additionally, the BM83 comes with a software package, making it easy to program the device to achieve our goals, without needing to directly program the device with our own script.

3.9.3 USB Connection

To achieve a hardwired connection, to a phone, we must consider a USB connection as most modern phones are equipped with USB-C ports for charging and data transfer, rather than the auxiliary cables of the past. This kind of connection is already common in audio systems like Apple CarPlay, so it is only natural to consider this as a viable option for our project. Even most smartphones of the past, that are not equipped with USB-C, are able to transfer data through the charging port which has a standard USB-A connection on the opposite end of the cable. The main point of consideration for the USB interface is the data transfer rate, as to be able to transmit lossless audio the data transfer rate of lossless audio is 1411 kbps or more [27]. Below we will discuss various options for the USB connection.

The USB-A was the only connection type considered because of its ability to implement with many different phone connections, like apple's lightning port, android phone's USB-B connections, and the modern USB-C ports. A USB-C port would also be capable of supporting our system's needs, however, to provide the maximum number of users with the ability to hardwire to the device, the USB-A port is the best option.

The USB standard is the most important consideration in choosing a specific connection, as the standard used dictates the connectors data transfer rate. However, after further research, every USB standard from 1.1 and on can transfer data at a sufficient rate [29].

Table 3.16 - USB Technology Comparison

Standard	Connectors Supported	Data Transfer
USB 1.1	USB-A	12 Mbps
	USB-B	
USB 2.0	USB-A	480 Mbps
	USB-B	
	USB Micro A	
	USB Micro B	
	USB Mini A	
	USB Mini B	
	USB-C	
	USB-C	
USB 3.0	USB-A	5 Gbps
	USB-B	
	USB Micro B	
	USB-C	
USB 3.2 Gen 1	USB-A	5 Gbps
	USB-B	
	USB Micro B	
	USB-C	
USB 3.2 Gen 2	USB-A	10 Gbps
	USB-B	
	USB Micro B	
	USB-C	
USB 3.2 Gen 2x2	USB-C	20 Gbps
USB 4	USB-C	40 Gbps

[28][29]

USB 1.1 is not going to be considered as the transfer rate is close to the minimum for lossless audio, whereas all other standards have much greater transfer rates when compared to the minimum for lossless audio. Furthermore, USB 3.2 Gen 2x2 and USB 4 are not going to be considered as the connector requires USB-C [29]. This leaves USB 2.0, USB 3.0, USB 3.2 Gen 1, and USB 3.2 Gen 2 to be used.

Any part that matches the standards discussed above can be used for our system, so these parameters were selected in the filters on DigiKey along with horizontal mounting, as the plan is to have the USB port come out one of the sides of the system. The “Exclude Tariffed Products” option was also selected to avoid any additional cost to the group. The selected part for our design was the 87520-0010BLF. This part meets the USB 2.0 standard, has 4 contacts and is rated for 1500 mating cycles.

3.9.4 Senior Design 2 Implementation

As mentioned above we chose the BM83 as our Bluetooth module to be used for our project. Although efforts were made to implement it into our project the module was function within our project as the ESP32 was unable to send the data received from the BM83 to our DAC once all code frameworks were implemented with each other.

The USB we chose ended up being the main way to stream audio to the ESP32. This did not come without additional efforts as the USB port chosen did not include the proper pinout to set a USB-C operated phone to USB OTG mode. An external USB OTG adapter was required to put a phone into USB OTG mode to communicate with the ESP32.

3.10 DAC/ADC vs CODEC

Data conversion is one of the focuses on this project, as the main input to our system will be audio signals that will go through a processing phase done through software, and will then be converted back to analog signals. With this in mind, we considered that having high-resolution converters (both A/D and D/A) is crucial to deliver the user a satisfactory and pleasant sound. One of the constraints that we faced when selecting the MCU was that many of the options that we took into consideration did not come with high-resolution DACs. Even though the ADCs are not as high as we could have, many microcontrollers implement them using 16-bit resolution which is good enough to deliver good audio quality, but we want to aim for better experience. This is why we decided to add external modules that would help achieve this. When researching different options. We encountered two scenarios: we could either add an external DAC module to meet the 16-bit standard of the built-in ADCs, or we could add a CODEC module to bypass both the ADC and DAC that come with the MCU. The focus of this section is to discuss the advantages and disadvantages of choosing between these two options.

3.10.1 16-bit vs 24-bit Resolution

Before jumping into any conclusion, we first want to go over the differences between these two types of resolution. When converting an analog signal to a digital representation, the number of ways in which data can be represented greatly impacts the resulting digital interpretation. When sampling an analog signal, the resolution of the converter determines how much of that signal we can extract to represent it in a digital manner. The more of these “interpretations” we have, the better the original signal will be conserved. In a 16-bit resolution system, we can represent the analog signal in $2^{16} = 65,536$ different ways. In the audio industry, this is usually referred to as “CD quality” and marks the minimum resolution required to interpret the original signal with minimum to no data loss. On the other hand, with a 24-bit resolution we can interpret the same signal in 16,777,216 different values; this clearly shows the advantage of using 24 bits versus using 16. This resolution is known as “Studio quality” because many will choose this resolution to record their music. [SOUNDGUYS]

3.10.1.1 Advantages and Disadvantages of 24-bit Resolution

Advantages:

- Better audio quality
- Better dynamic range (144dB vs 96dB)
- Less signal loss

Disadvantages:

- Files take 50% more size
- Required space and cost increase

3.10.1.2 Advantages and Disadvantages of 16-bit Resolution

Advantages:

- Good enough for most listening environments
- Smaller file sizes

Disadvantages:

- May introduce noise
- Less signal representation

On paper, 24-bit resolution seems to win over 16-bit, but the decision between these two comes down to the target audience intended with this project. While the advantages are clear, it is also worth mentioning that not many people distinguish the difference between these two resolutions. We also must consider the hardware limitations we face; utilizing an MCU means that we have limited storage and memory size. We plan on expanding the memory capacity with an external memory module that will be later discussed. Nevertheless, as stated before, using 24-bit resolution would increase the need for storage by 50%, which we cannot afford due to the constraints mentioned above. At the same time, most people cannot identify the difference between a 16-bit and a 24-bit audio file; only a very well-trained ear can notice it, which takes years in most cases. Finally, 24-bit resolution is usually implemented when recording audio, and, for the purposes of our project, we do not intend to do this. So, even though we will not limit our research to 16-bit converters, our system will not be using more than that.

3.10.2 CODEC

A CODEC is a hardware device, usually in the form of a microprocessor, that is used to compress and decompress data by implementing both analog to digital (ADC) and digital to analog (DAC) converters. In the case of audio, a CODEC is used to convert the analog audio signal into its digital representation and then decompressed back to be played in a sound system. In the case of our project, the CODEC would be used to override both the ADC and DAC converters from our MCU. However, adding an audio CODEC module to

our system would increase the size of the board, as well as the wiring/routing. At the same time, most of these devices implement the same number of ADCs as DACs. This is not a problem, but our only analog input will be the calibrated microphone, meaning that adding extra ADCs would be pointless.

3.10.3 DAC

Adding a single Digital to Analog Converter would simplify our design, as we would only need to worry about connecting it to our MCU, in both hardware and software ways. In terms of space, it varies depending on the number of channels the DAC implements, but in most cases, they are significantly smaller than CODECs, so placing and routing them would not pose a big issue. By adding a DAC, we are limiting our reach to modules that implement up to 16-bit resolution; however, as stated in section 3.10.1, we do not plan on going above this resolution.

3.10.4 Summary & Conclusion

After we decided to not push our final resolution to 24-bit, and considering the capabilities of the built-in ADC in our MCU, the best choice we can make is to add a DAC module and not a CODEC. We realized that implementing the latter would require more routing, space, and programming; due to the time constraints of our project, getting an external DAC is a better choice as it fits our needs and capabilities. In the following section, we will go through a list of candidates for the DAC that will go on our board and choose the one that best accommodates the project's vision.

3.10.5 DAC Module Comparison and Selection

The choice for the DAC depends on many factors. The two most important specifications we will take into consideration are bit resolution and the number of channels. The first one, as was previously stated, will ideally be between 16 and 24 bits. Even though we prefer 16-bit resolution for better file space handling, we will not turn down the possibility of using a 24-bit, as it can still be used as a 16-bit. The downside to this decision is that we would be paying for something that we would not be using. The second aspect of the DAC is the number of channels it implements. This is of high importance, as our system will need to drive sound to three different speakers; so, a multi-channel (preferably more than 4) DAC will be selected. We also do not want to choose the module with the highest number of channels because, in a similar way to using 24-bit resolution, we would be spending money on channels that will be left unused. Other factors, such as SNR or THD, are still important, but due to the advanced technology that is used to design these modules, we do not need to worry about these factors so much. The following table summarizes our candidates for the DAC.

Table 3.1717 DAC Module Part Comparison

Specification	DAC8555I PW	AD1934WB STZ-RL	AD5664AR MZ-REEL7	DAC8574IP W	PCM5122
Resolution (bits)	16	24	16	16	32 (max)
Channels	4	8	4	4	1
SNR (dB)	95	108	N/A	94	112
THD (dB)	-85	-94	-80	-65	-93
Max. operating Frequency (Hz)	50 M	27.6 M	50 M	3.4 M	50 MHz
Interface	SPI, DSP	SPI, I2S	SPI, DSP	I2C	I2C, SPI, I2S
Price (\$)	15.18	11.42	17.72	23.03	7.28

After analyzing the options above, we think the best DAC for our system will be the AD1934WBSTZ-RL. Even though we do not plan to extend our resolution to 24-bit, the module provides an option to scale it down to 16-bit, so the remaining 8 bits would be left unused. The number of channels we need for our system follows a similar idea, since we would only be using 3 of the 8 channels the DAC provides.

3.10.6 Further Considerations

The DAC we have chosen is only able to support a single-ended output, while our mids/highs amplifier (TPA3116) can accept this type of input, the subwoofer amplifier (TPA3251) is not capable of this input orientation in the mode we intend to use as it must be a differential input. This means we must identify a work around to enable our subwoofer amplifier. A few options are available to fit our needs.

The first option is to use a circuit provided by Analog Devices article on this exact topic. The circuit essentially uses two different op-amps, one to provide a buffer/gain stage and the other to convert the signal into the differential signal required for our subwoofer amplifier. The article includes all the necessary calculations required for the design to meet our needs [31].

The second option is to use a circuit provided by TI that also utilizes two op-amps. However, this circuit does not have two stages, one op-amp is for the negative differential signal and the other is for the positive differential signal. This is a better choice than the

one provided by Analog Devices as it can use the same op-amp that comes in a package of one, two, or four op-amps on the same chip [32].

The last option is to by a dedicated single-ended to differential signal converter IC, which seems to be the best option as little design work will need to be completed, and the circuit can be set up in a typical application mode to achieve our goals. The IC in question is the LT6350CMS8#TRPBF. The data sheet for this device is extensive and provides all the necessary formulas required for the design. It also provides a typical application circuit for circuits that only require a gain of one, which follows our needs. This is the device we will be using to achieve the single-ended to differential conversion as it allows for simple integration.

3.10.7 Senior Design 2 Implementation

During our implementation stages in Senior Design II, we faced several issues with the chosen DAC, the AD1934. The most notorious issue faced with this was that the PLL filter was not operating correctly and DAC to not lock onto the incoming clock signals. We considered it was a better option to switch to a different one given the time left to complete the project and the complexity of the problem as with a board redesign, we may have run into the same. In the end, we implemented the PCM5122, a much more friendly DAC which offered similar specifications as the AD1934, but this DAC could be setup with hardware wirings rather than software. With an added constraint, the device only has one channel.

Additionally, the single-end to differential signal converter was changed to the OPA1637DGKR, as the new op-amp required less components and was budget friendly.

3.11 Power Supply Unit (PSU)

3.11.1 PSU Requirements

When considering our PSU requirements, we must abide by the guidance given in the data sheets of our selected components. This information is organized on the table below to give a quick understanding of the power requirements for each component and the system.

Table 3.1818 Power Requirements

Component	Voltage	Current	Power Consumption
MCU	4V	620 mA	2.5W
Mids/Highs Amplifier (x2)	26V	4.7A	122.2W
Subwoofer Amplifier	36V	8.5A	306W
Bluetooth Module	3.2V – 4.2V	59 mA	247.8 mW
LiDAR (x2)	4.3V - 5.2V	50 mA	215mW - 260mW

Microphone	15V - 48V	4 mA	60mW – 192mW
DAC	3.6V	169 mA	608.4mW
Total	N/A	N/A	554.21W

As seen in the table above, our system requires 554.21W to support the design. With this we will need to find a power supply that meets this level and then some to offer ample headroom. The goal of this project is to have the system powered by a wall outlet, so the focus of options meets this goal.

3.11.2 Power Supply Options

There are only a few power supply options available that fit our budget, and they are all from Mean Well USA Inc. The PSUs are the LRS-600-48, LRS-600N2-36, LAD-600C, and LAD-600CU, with price tags of \$59.90, \$64, \$77.90, \$89.30. The LAD units share the same data sheet; the only difference is the CU option has UART capabilities for monitoring the system.

The LRS-600-48 unit can withstand 300VAC surges for 5 seconds, has short circuits, overload, overvoltage, and over temperature protection. The unit is also capable of operating up to 65C and has a built-in fan. The output of this unit is 48V and 12.5A with 92% efficiency. The LRS-600N2-36 is like this but has an output of 36V and 16.6A with 92% efficiency. The N2 model also has a 200% peak power capability, which can offer ample headroom in the case that the instantaneous output power of the amplifiers exceeds expectations.

The LAD models have the same protections as the previous model discussed with the addition of battery charging and protections to pair with the added feature. The unit can withstand the temperature range of -20C to +60C. The units can output 41.5V and 11.45A at 91% efficiency.

Table 3.1919 PSU Comparison Table

	LRS-600-48	LRS-600N2-36	LAD-600C	LAD-600CU
Output Voltage (V)	48	36	41.5	41.5
Output Current (A)	12.5	16.6	11.45	11.45
Efficiency (%)	92	92	91	91
Temp Range (C)	-20 to +65	-20 to +65	-20 to +60	-20 to +60
Distinguishing Features	Fan	Fan and 200% peak power capability	Battery Charger	UART and Battery Charger

Price	\$59.90	\$64	\$77.90	\$89.30
-------	---------	------	---------	---------

3.11.3 Summary

The power supply we chose for our design is the LRS-600N2-36 as the 200% peak power capability is a good feature to pair with our amplifiers and the fan will keep the unit cool in the case operating temperatures exceed expectations. Additionally, the 36V output is about the exact output needed for the subwoofer amplifier. Lastly, the unit is the second cheapest option that was compared.

3.11.4 Senior Design 2 Implementation

We used the power supply we chose from our initial design and found that it provided ample headroom.

3.12 Technology Research and Comparison

The development of the graphical user interface (GUI) and mobile application requires careful selection of frameworks, programming languages, and supporting tools. Multiple options exist for building cross-platform applications, native mobile interfaces, and embedded system dashboards. The following subsections summarize research into the most relevant technologies, along with a comparison of their strengths and limitations for the automated sound system project.

3.12.1 Cross-Platform Frameworks

Cross-platform frameworks enable a single codebase to be deployed across both Android and iOS operating systems, reducing development effort while ensuring consistency in design and functionality.

- Flutter (Google) Flutter uses the Dart programming language and compiles native code for both iOS and Android. It offers a widget-based UI system, high rendering performance, and native-like responsiveness. Flutter also provides strong support for data visualization, making it suitable for rendering EQ plots, sliders, and live system feedback [33].
- React Native (Meta) React Native employs JavaScript/TypeScript and integrates with native APIs to deliver platform-specific features. It benefits from a large open-source ecosystem and strong community support. However, it may suffer from performance limitations in graphically intensive applications compared to Flutter [34].
- Xamarin (Microsoft) Xamarin uses C# and the .NET framework to build cross-platform applications. While it provides deep integration with Microsoft tools, it has a steeper learning curve and fewer community-driven libraries for advanced data visualization.

For the automated sound system, Flutter presents the most balanced option due to its strong support for graphics, responsive performance, and active development ecosystem.

3.12.2 Native Mobile Development

Native development offers direct access to platform-specific APIs and maximum performance.

- Android (Java/Kotlin) Android Studio supports development in Java and Kotlin, providing direct access to device hardware, Bluetooth, and Wi-Fi APIs. Native Android offers high performance but requires a separate codebase from iOS.
- iOS (Swift) Xcode with Swift provides native integration for iPhones and iPads. Swift is optimized for performance and security but, like Android development, requires maintaining a distinct codebase.

Although native development yields performance advantages, the requirement to maintain two independent applications increases development time and complexity, which is impractical within senior design constraints.

3.12.3 GUI Libraries and Design Standards

To ensure consistency and usability, the GUI follows established design guidelines and leverages for proven component libraries.

- Material Design (Google) – Provides a standardized UI/UX framework with responsive design principles, accessibility features, and customizable components [35].
- Cupertino (Apple) - Mimics native iOS design aesthetics for seamless integration on Apple devices.
- Custom Components - For unique visualizations such as EQ frequency plots, custom graphics libraries such as Charts in Flutter or D3.js in React Native can be integrated.

For this project, Material Design is prioritized to ensure platform consistency and minimize training requirements.

3.12.4 Backend Integration Strategies

The GUI and mobile application must communicate with the embedded system in real time. Several communication strategies were evaluated:

- REST APIs - Widely adopted, stateless communication method over HTTP(S). Best for configuration commands and historical data requests.
- WebSockets - Enables persistent bi-directional connections, reducing latency for continuous updates (e.g., real-time EQ feedback).

- MQTT (Message Queuing Telemetry Transport) - A lightweight publish-subscribe protocol optimized for IoT applications with low bandwidth [36].

WebSockets are selected as the primary strategy for low-latency communication, supplemented by REST APIs for configuration tasks.

3.12.5 Security Considerations

Mobile application security is essential for preventing unauthorized access and protecting system data. The following standards and best practices are applied:

- OAuth 2.0 Authentication - For secure user access and role-based permissions [37].
- TLS Encryption - Ensures that all wireless communication is encrypted.
- Secure Storage APIs - Sensitive data such as user profiles and presets are stored using platform-secure keychains or encrypted local storage [38].

3.12.6 Testing Tools

Testing ensures that the GUI and mobile application function correctly across devices and operating systems.

- Appium - Open-source framework for automated testing across iOS and Android.
- Espresso (Google) - Android-specific testing framework with strong integration in Android Studio.
- XCTest (Apple) - Native testing framework for iOS applications.

Given the cross-platform choice of Flutter, Appium is selected as the primary testing tool to ensure comprehensive automated UI validation.

3.12.7 Technology Comparison

A comparative summary of the most relevant technologies is shown in Table 3.2.1.

Table 3.2020 Comparison of GUI & Mobile App Development Technologies

Category	Flutter	React Native	Native (Android/iOS)	Xamarin
Language	Dart	JavaScript/TypeScript	Kotlin / Swift	C# / .NET
Performance	High (native compiled)	Moderate (JS bridge)	Maximum (native)	Moderate
Cross-Platform	Yes	Yes	No	Yes
Libraries & Ecosystem	Strong visualization support	Large open-source support	Platform-specific only	Limited

Development Effort	Single codebase	Single codebase	Dual codebase required	Single codebase
Best Use Case	Data visualization, IoT apps	Social & general-purpose	High-performance apps	Enterprise integration
Project Fit	Excellent	Moderate	Poor (time constraints)	Low (limited support)

3.12.8 Summary

Based on the evaluation, Flutter is selected as the primary framework for developing the GUI and mobile application. Its widget-based rendering engine allows for responsive and visually rich interfaces, which are essential for representing EQ graphs and real-time system monitoring. The combination of Flutter with Material Design guidelines, WebSocket-based communication, and Appium testing ensures that the application will meet project goals for usability, performance, and security while remaining feasible within project constraints.

3.12.9 Senior Design II Implementation

For the final Senior Design II implementation, the application remained built in Flutter and was reduced to two primary screens: the Control Page and the Settings Page. The final app included a Start Calibration button that played a sine sweep, displayed a feedback-based histogram, supported light mode and rainbow mode, and expanded the manual EQ controls to eight frequency bands: 60 Hz, 125 Hz, 500 Hz, 1 kHz, 2 kHz, 4 kHz, 8 kHz, and 16 kHz.

3.13 Test Signal

Our algorithm will depend on the signal captured by our calibrated microphone, which will be produced after a test signal is played through the speakers. Each type has different characteristics, which is going to affect how the logic behind the algorithm is built. The signals that we considered for this project are the following:

- White Noise
- Pink Noise
- Sine Sweep (logarithmic vs linear)

The test parameters for all signals are going to be the same. That is, the duration of the signal will be 10 seconds (reduces memory/storage complexity while still giving us sufficient time to analyze the response), the frequency range is going to be 20 Hz – 20 kHz (human hearing spectrum), and the sampling frequency will be 44.1 kHz. In the following sections we will compare the possible test signals.

3.13.1 White Noise

White Noise is characterized by distributing the same amount of energy at each frequency level. This means that, when white noise is played, it will be able to reproduce all frequencies in the range with the same intensity. [39] It is known for being a simple test signal, because it does not involve sweeping through the spectrum over time or changing the energy at any level. The advantage of this signal is that the analysis turns out to be simpler than others because of its linear characteristics. However, the problem with having all the frequencies play at the same time is that we do not have a direct way to identify how much of the sound belongs to the signal and how much to noise. This makes the spectral analysis a bit more difficult, since we would have to implement other methods to predict how much noise is present and how to minimize it.

3.13.2 Pink Noise

Like White Noise, Pink Noise is characterized by injecting more energy into the lower frequency bands. As opposed to White Noise which distributes equal energy across each frequency, Pink Noise distributes equal amounts of energy over octaves (interval between frequencies), making it look as if the lower frequencies have more energy than the higher ones. [40] This is usually referred to as a logarithmic approach to White Noise, which is a linear approach.

3.13.3 Sine Sweep (Log vs Linear)

The third and last way we could test our system is with a sinusoidal sweep. This sample differs from the previous two because it reproduces all the frequencies in a frequency range one by one over a specific amount of time. This means that, the longer the sample, the slower the sweep will be and vice versa. Sine sweeps are usually preferred in a professional setting because they give a clear, less noisy representation of all the frequency levels. There are two types of sweeps: logarithmic and linear. These two methods are derived from White and Pink noise, since their characteristics are similar. A sweep allows the signal to produce each frequency with more energy, as it does not need to be distributed over all the frequencies at once, giving it an advantage over White or Pink noise. [41]

3.13.4 Senior Design 2 Implementation

For the test used in our final design we decided to use the sine sweep, logarithmic variation, to test not only the frequencies that could be produced by our speakers but also used to provide the EQ algorithm as much needed information as possible.

Chapter 4 – Standards and Design Constraints

4.1 Design Standards

4.1.1 Bluetooth Standard

Bluetooth has become a standard since its introduction in hopes of replacing the RS-232 standard at the time for serial cables. Bluetooth was a low power consumption remedy for wirelessly connecting devices in short range of one another. Since its inception with

Bluetooth 1.0 we have now evolved into Bluetooth 6.0. With each iteration being introduced, they tend to focus on security updates, effort into updating the CODEC and ultra-wideband technology, and faster speeds.

4.1.1.1 Bluetooth 1.0

As mentioned before Bluetooth 1.0, which was released in 1999, and was the answer to the, even outdated at the time, RS-232 serial cable standard. Bluetooth 1.0 allowed peripherals to connect wirelessly to devices, all while being lower power. [42]

4.1.1.2 Bluetooth 2.0-2.1

In 2004 Bluetooth 2.0 was released featuring an enhanced data rate or EDR. It allowed the speed of Bluetooth to climb up to 3 Mbps, which allowed things like wireless audio. This update also brought along the ability to connect to multiple devices at once. Another area of improvement was the longer range of transmission of 30m, which was three times more than its predecessor. With Bluetooth 2.1 a new connection method known as secure simple pairing, or (SSP), was implemented. [42]

4.1.1.3 Bluetooth 3.0

Arriving in 2009 Bluetooth 3.0 established a significant boost in performance speed due to an optional feature known as High-Speed. This allowed transfer rates to reach speeds of 24 Mbps, however the data transfer in these cases took place over Wi-Fi instead of Bluetooth. This was a potential issue because it required the devices to have Wi-Fi capabilities. Because of this, this iteration of Bluetooth was never widely adopted. [42]

4.1.1.4 Bluetooth 4.0 – 4.2

The main attraction for Bluetooth 4.0 was that it implemented Bluetooth low energy or BLE. This accommodated for devices with bursty data, while Bluetooth 4.2 built upon improving security and privacy with smart homes in mind. [42]

4.1.1.5 Bluetooth 5-5.4

Released in 2016, Bluetooth 5.0 allowed BLE links to be twice as fast as the previous iteration, going up to 2 Mbps. In addition, this standard led to the ability to sacrifice speed for increase in range between devices. With Bluetooth 5.1 in 2019, the introduction of direction finding, allowed precise tracking. In 2019 with Bluetooth 5.2 and the LE Audio standard, we saw the integration of a new low complexity communication CODEC, which allows for superior audio quality. With Bluetooth 5.2 the standard also introduced audio broadcasting. Bluetooth 5.3 and 5.4 were released in 2021 and 2023 respectively and ensured stability and connection enhancements. With Bluetooth 5.4 there was the coming of periodic advertising with responses which allowed a Bluetooth beacon to broadcast messages to as many as 7,000 devices at one time. [42]

4.1.1.6 Bluetooth 6.0

The new standard of Bluetooth 6.0 sees the introduction of Bluetooth channel sounding which allows a device to accurately measure the distance between two users. This is done

by round trip time, or RTT, and phase-based ranging, or PBR, for the accuracy of the distance measurement. [42]

4.1.2 USB Standards

USB or Universal Serial Bus is an industry standard developed by USB Implementers Forum (USB-IF) and has been used as a method of sending and receiving data between two linked devices, and in addition can be used to distribute power. Through the life of the USB, there have been many addendums to the standard, as there have been seven iterations of the standard, not including the variety of ports the USB has seen implemented into their technology.

4.1.2.1 USB 1.0

The first of the USB standard came in 1996 and was introduced to replace parallel ports, PS/2 connectors, and SCSI connectors, among many others rivaling connection methods that were prevalent at the time, it came with data transfer rates of up to 1.5 Mbps at low speed, and 12 Mbps at full speed. It is also worth noting that USB 1.0 offered hot-swapping and self-configuration. This standard used type-a and type-b ports. [43]

4.1.2.2 USB 1.1

The next iteration of the USB standard came in 1998 and has also been known as the “Full Speed USB” and was created due to issues regarding the launch of USB 1.0. In USB 1.0, 1.1, and 2.0 these standards use a protocol when spoken to protocol, which means they do not communicate with the host unless the host makes a request. In terms of data transfer speeds, it matched the speeds found in USB 1.0 but could also operate at slower speeds for lower bandwidth devices. This standard used type-a and type-b ports. [44]

4.1.2.3 USB 2.0

Due to the much wider need for data transfer speeds, caused by a tech boom in the late 1990’s and into the early 2000’s, the USB 2.0 standard was created. With an upgrade of data transfer rates of 480 Mbps, but bus limitations capping the speed at 280 Mbps it offered a substantial upgrade to the previous 1.0 and 1.1 standards. To help build the reputation and widespread use of USB 2.0 was given backwards capability with and would still operate at the previous 1.5 or 12 Mbps speed. In addition, the use of USB being used as a power source as became a more common design and could supply up to 500 mA and 5V. Due to these changes, this iteration of the standard was named “High Speed”. [44]

4.1.2.4 USB 3.0

Released in 2008, USB again renovated their technology to support data transfer up to 5 Gbps but attained speeds closer to 3 Gbps. By doubling the four connection lines in the USB 2.0 standard it allowed bi-directional transfer, moving it from the previous spoke when spoken to transfer logic. Since by 2008 power transfer through USB became more of a common practice, they also increased power transfer capabilities from the previous 500mA to 900mA but still produced 5V. In addition, USB 3.0 is still backwards compatible with USB 2.0. [44]

4.1.2.5 USB 3.1

Like USB 3.0, USB 3.1 was released in 2013 and offered many of the same characteristics minus the doubling of the data transfer rate up to 10 Gbps. It was given the SuperSpeed+ moniker. [44]

4.1.2.6 USB 3.2

Arriving in 2017 USB 3.2 was established and replaced the previous 3.0 and 3.1 standards. They added data transfer capabilities up to 20 Gbps and capitalized on what is called the dual lane transfer channel found in USB-C connector ports. This allows for 10 Gbps in each direction over the two wire pairs. [44]

4.1.2.7 USB 4.0

Finally in 2019, the most current USB standard arrived at 4.0. This standard sees the data transfer rate increased again with speeds of up to 40 Gbps, while increasing the power output maximum to roughly 240W. [44]

4.1.3 Flutter Standards

The Flutter framework, which was developed by Google, provides a complex SDK for creating high performance, cross-platform applications using a single codebase. Flutter projects can scale rapidly, so adhering to consistent coding and architectural standards are critical to maintain performance, readability, and collaboration throughout a project's lifecycle.

4.1.3.1 Coding Style and Formatting

Flutter follows an official “Effective Dart Style Guide”, which establishes convention standards for formatting, naming, and structuring code to encourage uniformity and clarity. Developers should use lower CamelCase for variable names, methods, and functions. Whereas for classes and enumerations developers should use Upper CamelCase [44]. File and directory names are written in snake_case to maintain consistency across the codebase. [47].

All code should be formatted automatically using the built in “dart format” tool or the integrated IDE formatter tool committing to a repository [48]. Automated formatting ensures that the code reviews focus on logic and design without any indenting or spacing inconsistencies. Additionally, descriptive function and variable names should clearly communicate their purposes without the need for excessive comments [49]. This standard improves comprehension and long-term maintainability.

4.1.3.2 Folder Structure & Modularity

Developers should maintain a clear and consistent folder structure in the projects. Maintaining these is an important aspect of Flutter development. Venkat GSVS [49] recommends grouping related components such as screens, widgets, models, and services

into dedicated directories. This organization supports any concerns over separation. This allows UI design, state logic, and data management to be developed independently.

Additionally, using a feature-based directory structure, such as features/login & features/settings, allows easier scaling and revision as the project evolves. This also aligns with Google's internal Flutter architecture guidelines that emphasize self-contained reusability and efficient collaboration among a team.

4.1.3.3 Architectural Patterns & Widget Management

Flutter follows a declarative model of UI with widgets and automatically refreshes the UI upon any change in application state with the press of "R" in the terminal. So, to keep it simple, Google suggests models like Model-View-ViewModel (MVVM) or Business Logic Component (BLoC) to separate application presentation from application logic [55].

Widgets should be designed to be lightweight and reusable. Complex widgets can be broken down into smaller widgets, making it easy to read and render them faster on the screen [49]. Developers should avoid using "StatelessWidget" for static UI elements and use "StatefulWidget" to implement stateful interaction. Declaring widgets as const when their configuration is immutable further optimizes rendering and reduces unnecessary rebuild [49].

4.1.3.4 Performance & State Management

Flutter's real-time rendering engine requires strict performance discipline. Blocking the main UI thread with synchronous operations can cause dropped frames. Asynchronous programming using "Future" and "async/await" must be employed for network calls, file I/O, or computation-heavy tasks [49]. Intensive calculations should be moved to isolates, enabling parallel processing [51].

Another key standard is to avoid global or "hidden" application state. Venkat GSVS [49] cautions against using singletons or global variables for data management. Instead, structured state-management frameworks such as Provider, Riverpod, or BLoC should be adopted to ensure predictable state flow and easier debugging.

4.1.3.5 Flutter Design Standards Summary

Flutter design standards guide developers toward sustainable and maintainable codebases that meet both functional and performance requirements. By following these standards, the team can ensure the following:

- Consistent naming, formatting, and directory conventions [46], [47], [48].
- Modular feature organization and clear separation of concerns [49].
- Widget-centric architecture using stateless and constant widgets [49].
- Responsive UI through asynchronous operations and an optimized build process [49], [44].

Adhering to these standards ensures a clean, efficient, and scalable Flutter application.

4.1.4 PCB Standard

The Printed Circuit Board (PCB) used to interconnect the ESP32-WROOM module, the BM83 Bluetooth audio module, and the supporting circuitry must follow established PCB industry standards to ensure electrical reliability, thermal performance, signal integrity, and manufacturability. Our PCB is designed using 1 oz/ft² copper traces ($\approx 35\text{ }\mu\text{m}$), which is consistent with industry norms for mixed-signal and wireless communication systems.

According to IPC-2221 (Generic Standard on Printed Board Design), copper thickness and trace width must be selected based on the expected current, allowable temperature rise, and environmental conditions [55]. In our design, the ESP32 power rail and BM83 digital/command traces are routed using 1 oz copper, which provides sufficient current-carrying capability while maintaining a reasonable board area.

4.1.4.1 Copper Thickness and Trace Sizing

Per the IPC-2152 Standard, the current-carrying capacity of 1 oz copper depends on trace width, ambient temperature, and acceptable temperature rise [56]. The ESP32-S3 and BM83 draw relatively low to moderate currents:

Table 4.1 Copper Thickness and Trace Sizing

Subsystem	Copper Weight	Typical Trace Width	Rational
ESP32 3.3V	1 oz	10-12 mil	Supports $\approx 250\text{ mA}$ peak draw with thermal margin
BM83 Power rail	1 oz	10 mil	Ensures stable supply for RF operations
GPIO / UART / I ² S lines	1 oz	6-8 mil	Low-current digital lines

The use of 1 oz copper ensures adequate thermal performance while minimizing board cost and complexity. Multi-via stitching is used on the power nets to reduce impedance and distribute current.

4.1.4.2 Manufacturing and Reliability Standards

The PCB is designed to meet IPC-6012 Class 2, the industry standard for general electronic products requiring consistent performance but not mission-critical reliability [52]. Class 2 suitability includes:

- Plated through-hole quality

- Solder mask adhesion
- Minimum annular ring dimensions
- Acceptable copper continuity and plating consistency

This class is appropriate for consumer audio and embedded communication systems such as ours.

4.1.4.3 RF and Bluetooth Layout Requirements

Both the ESP32-S3 and BM83 Bluetooth module require compliance with their vendor-provided hardware layout guidelines. Espressif specifies:

- A continuous ground plane under the ESP32 module
- A “keep-out” zone around the antenna
- Short, direct routing for high-speed signals (UART, I²S)
- Antenna placed at the board edge to reduce attenuation [53]

Similarly, Microchip’s BM83 hardware guide specifies:

- No copper pour under the antenna region
- Controlled-impedance routing for RF paths
- Isolation between digital switching traces and RF/power domains
- Mandatory ground stitching around the module perimeter [55]

These requirements reduce RF reflections, minimize interference, and ensure stable Bluetooth performance.

4.1.4.4 Signal Integrity Considerations

Following IPC-2221 recommendations, sensitive analog and digital circuits are separated to reduce coupling. Ground-return paths are kept short, and decoupling capacitors are placed as close as possible to ESP32 and BM83 power pins to ensure stable operation.

High-speed communication (e.g., UART to BM83 and I²S later in SD2) is routed according to controlled-impedance and length-matching practices where applicable. This prevents ringing, signal degradation, and data corruption.

4.1.4.5 Summary

By following IPC-2221, IPC-2152, and IPC-6012 PCB design rules, along with vendor-specific hardware guidelines from Espressif and Microchip, the PCB provides:

- Adequate current capacity
- Reliable thermal characteristics
- Proper RF behavior for Bluetooth connectivity
- Stable power distribution

- Manufacturable board geometry compliant with industry standards

These design choices ensure that the ESP32 and BM83 modules operate reliably within the final product's embedded system environment.

4.2 Design Constraints

4.2.1 Power Constraints

4.2.1.1 Power Budget and Supply Limits

For our power supply, we need to produce enough instantaneous power, roughly 600W to supply the MCU, 2 LiDAR sensors, the DAC, and each of the speaker amplifiers. In addition, our power supply unit must be able to deliver this power while adhering to the 3.6V, 4V, 26V, and 36V voltage rails, all while maintaining thermal stability. By implementing a 20% – 25% safety margin, we would be providing enough headroom for the power supply units capacity. Since we are planning on using a wall outlet as the source of power, we would need a robust AC to DC conversion stage while still adhering to UL and IEC safety standards for audio equipment.

4.2.1.2 Voltage Regulation and Power Distribution

By including high voltage rails such as the 26V – 36V rails, while having components that require much smaller rails such as the MCU, and DAC which require 3.3V – 5V rails we must limit the amount of interference or ripple between these rails. By designing a multi-rail power distribution network, we would be able to operate at different voltage levels and noise tolerances but still account for system efficiency. We would need to ensure we are not creating ground loops due to the mixed-signal system, by setting star-ground points and separating power ground and signal ground. In addition, we need to make sure high current traces are kept short and separate from analog and digital traces.

4.2.1.3 Efficiency vs. Quality

By utilizing class D amplifiers which have built-in thermal regulation we may introduce switching noise into the DAC lines. The efficiency of class D amplifiers comes at the cost of signal reconstruction. In this case the constraint becomes balancing power efficiency with total harmonic distortion, and electromagnetic interference. This type of interference can cause hisses and harmonic distortion in quiet audio passages. By implementing a low pass LC filter, we can remove high frequency pulse width modulation components while still allowing the full audio band to pass through

4.2.1.4 Transient Current

Since we are dealing with large amplifiers, we could experience several amps of inrush current at startup. This is caused by amplifiers having dynamic loads from when every low frequency pulse or bass note is exerted from the speaker, making the amount of instantaneous current draw in the amplifier to increase. These swings can reach two to three times the amplifiers' average current consumption. This would lead to clipping, distortion, logic errors on the MCU and ADC inaccuracies. The constraint would be that the system

and power supply need to be able to handle the transient rushes without tripping, implementing a staged power-up sequence could fix this issue. Furthermore, we need to design the PCB to absorb and smooth out these transient load variations. We could also incorporate decoupling strategies such as implementing reservoir capacitors, low-ESR and high frequency bypass capacitors, localized decoupling, and star topology power distribution.

4.2.2 Audio Constraints

When designing an automated audio system, understanding and mitigating audio constraints is critical to ensuring consistent sound performance across varied acoustic environments. Audio design constraints define the physical and digital limitations of sound reproduction and processing. These constraints include acoustics in different audio settings, technical capabilities, audio enclosures, and audio latency constraints. These play critical roles in defining audio performance in any audio system

4.2.2.1 Room Acoustics Constraint

The acoustics in the room significantly affect the way in which sound waves interact with enclosed spaces. Hard materials such as concrete, glass, and tile tend to cause unwanted echoes and standing waves, while softer materials such as carpet and curtains help absorb sound and minimize reflections [57]. These factors affect the tonal balance of the audio being produced.

Constraint: The system must include an adaptive digital signal processing (DSP) calibration algorithm capable of adjusting frequency response to compensate for acoustic variations between reflective and absorptive environments.

4.2.2.2 Speaker Placement Constraint

Speaker placement is related to the imaging in stereo, low-frequency performance, and clarity of spatial location in audio sources. The placement of speakers in corners or close to walls can lead to boundary gain, which is an amplification in low-frequency performance, and incorrect alignment can cause interference resulting in phase cancellation [58].

Constraint: The system must maintain balanced sound reproduction regardless of speaker location. The DSP algorithm should include equalization compensation for near-wall or asymmetrical placement, allowing consistent playback performance in multiple configurations.

4.2.2.3 Crossover Frequency Constraint

Crossover frequency defines how sound energy is divided among the speaker's drivers (woofer, midrange, and tweeter). Poor crossover design can create spectral overlaps that cause distortion or frequency gaps that reduce clarity [59].

Constraint: The DSP algorithm should perform crossover filtering tasks in the digital realm to keep the resulting frequency response curve level. The ability to change crossover filtering points via software should be considered to suit different speakers.

4.2.2.4 Speaker Enclosure and Size Constraint

Speaker cone diameter and enclosure volume influence the system's frequency response and efficiency. Larger woofers produce deeper bass, however, require larger enclosures. Whereas smaller speakers facilitate portability with reduced low-frequency extensions [55]. Enclosure designs, namely sealed, ported, and passive radiators, further impact transient responses and harmonic distortions.

Constraint: The enclosure needs to capitalize on internal air space and damping to reconcile larger dimensions with portability and sonic fidelity. Vibrations and resonance can be damped with internal bracing and absorption materials.

4.2.2.5 Audio Hardware Resource Constraint

The microcontroller and DSP hardware have limited computational capacity for real-time signal processing. Audio equalization, filtering, Bluetooth streaming, and microphone analysis all share the same processing resources [61].

Constraint: DSP algorithms must remain computationally efficient and execute within available clock cycles and memory. The total processing latency per audio frame must remain below the maximum allowable delay for real-time operation.

4.2.2.6 Audio Latency Constraint

Latency, the delay between input (microphone or command) and output (speaker), is another critical constraint. Excessive latency can cause echo, feedback, or misalignment between the GUI and the audible response.

Constraint: The total end-to-end latency, including ADC/DAC conversion, DSP processing, and wireless transmission, should not exceed 20 ms to preserve real-time responsiveness [62].

4.2.2.7 Psychoacoustic and Frequency Response Constraint

Human hearing exhibits non-linear sensitivity to frequency and loudness, described by Fletcher-Munson equal-loudness contours [63]. At low listening levels, bass and treble frequencies are perceived as quieter.

Constraint: The system must apply psychoacoustic correction curves to ensure perceived tonal balance across volume levels, maintaining a consistent user experience whether the sound is quiet or loud.

4.2.2.8 System Integration Constraint

The interface between the Flutter-based GUI and the embedded DSP system must ensure reliable synchronization. Control commands, equalizer presets, and feedback values are transmitted wirelessly, and packet delays or jitters can disrupt real-time adjustments [69].

Constraint: The Bluetooth Low Energy (BLE) or Wi-Fi communication stack must maintain an average command latency under 100 ms, ensuring seamless user interaction and immediate system response to GUI input.

4.2.3 Software Compatibility & Integration Constraint

When it comes to an embedded audio solution consisting of a Flutter-based mobile application, a Digital Signal Processor, and a Microcontroller Unit, ensuring software compatibility and data integrity in communications is essential to ensure consistency in the functioning of the hardware and software interfaces of the project. The project should facilitate the timely exchange of command signals, audio updates, and status information without any glitches in communications.

The Flutter app communicates with the MCU via the UART, I²C, and SPI interfaces. These interfaces function in different aspects of the process: UART for asynchronous command communication, I²C for low-speed control of peripheral devices, while SPI handles high-speed data exchange between the MCU and the DSP [68]. The communication process between the sensors, MCU, and the Flutter app involves multiple aspects to be considered. Synchronization among these communication buses ensures deterministic execution, preventing latency accumulation or command overlap during data transactions.

To preserve user experience and maintain control accuracy, all command transmissions from the Flutter GUI to the MCU must occur deterministically, with a maximum allowable latency of 100 ms between user input and system response. Additionally, since firmware updates and configuration data are transmitted over the same serial channels, memory allocation on the MCU must be carefully managed. Firmware image size must not exceed the available on-chip flash memory capacity to avoid system instability or partial writing failures [66].

Constraint: The system software must maintain deterministic communication across UART, I²C, and SPI interfaces, ensuring that API commands execute within 100 ms and firmware updates remain within MCU memory constraints.

4.2.3 Financial Constraint

As a part of a project that is not funded by a company, we, the group members, are responsible for funding the entirety of the project. However, as much as we budget for the components, we must remind ourselves of the cost and performance trade-off. By electing to pick the highest quality components every time, we run the risk of being way over budget and not able to complete the entirety of the project. On the other hand, if we elect to choose the cheapest components available, we run the risk of being unable to produce a high-

performance product. The trade-off is one that cannot be taken lightly. Our response to this constraint can be seen as our breakdown of why we choose each component, including comparisons of all specs required to meet our specifications while still being within our budget.

4.2.4 Time Constraint

With taking Senior Design, we are tasked with creating a project and doing research in a two-semester time. Although this may seem like a lot of time, there is a complexity level required for the project that requires each member of their team to be proactive in terms of commitment time. In addition, when determining the scope of the project and deciding what technologies we would like to implement, we must be mindful of whether that task can be completed within our allotted time. Meeting frequently and going to meetings with an agenda helps speed the level of progress being achieved in each meeting and brings us closer to our goal. By practicing positive time management our group can meet the deadlines set in place by the class and can meet the demands and specifications for the project.

Chapter 5 – Application of ChatGPT and Other Platforms

5.1 Overview

ChatGPT and other AI platforms are new tools that engineers can use to benefit their learning, research, and design. These tools have started to find their way into the lives of many people, not only engineers, as they can be used as a friend, resource, search engine, etc. to help the user achieve any goal they have. However, these resources are only a few years old and have some significant drawbacks to their use, which will be discussed in detail later in the chapter.

With AI being so easy to use, many people are beginning to rely on it to complete many of their delegated tasks. This can be both a blessing and a curse, as AI is not completely reliable and must be fact checked. As a result, the group has come up with a set of rules and expectations for the use of AI, specifically ChatGPT, as it is the most popular AI model.

The content of this chapter will dive into a brief history of AI, the pros and cons of AI, the subsequent rules and expectations we have set for ourselves when it comes to using AI, and examples of how the resource has benefited (or harmed) each member of the group during our investigations.

5.2 Brief History of AI

The concept of artificial intelligence (AI) dates to the mid-20th century, when researchers began exploring whether machines could replicate aspects of human reasoning and decision-making. In 1950, Alan Turing published *“Computing Machinery and*

Intelligence”, introducing the famous Turing Test; a method to evaluate whether a machine could exhibit intelligent behavior indistinguishable from that of a human.

The early AI researchers, like John McCarthy, Marvin Minsky, Allen Newell, and Herbert Simon, worked on the first programs that could solve problems using logic during the 1950s and 1960s. The term “Artificial Intelligence” was first coined by McCarthy during the Dartmouth Conference in 1956, and it is marked as the birth of AI.

The following decades saw a period of stagnation, often referred to as “AI winters.” The early forms of AI were too computationally expensive for the technology available at the time and offered little learning ability, making it difficult to apply in real-world situations. The 1980s and 1990s witnessed a revival of interest in artificial intelligence driven by improvements in computer capabilities and storage, leading to the emergence of expert systems. This refers to software that contained human knowledge expressed through a rule-based system.

The beginning of modern AI began in the 2010s, driven by the emergence of machine learning (ML) and deep neural networks. The concept enabled computers to train on big datasets instead of depending on algorithms programmed by human developers. The availability of graphics processing units (GPUs), cloud computing, and big data enabled rapid training of large models for image recognition, speech synthesis, and natural language processing.

The release of ChatGPT and other LLMs in 2022 introduced another milestone in the domain of AI for the masses. ChatGPT, for example, combines natural language processing, reasoning, and generation, enabling users to engage conversationally with AI for education, research, and design. Ultimately, over the past three years, AI has transitioned from a niche research discipline to a mainstream technological tool embedded in many fields, including engineering, software, healthcare, and business.

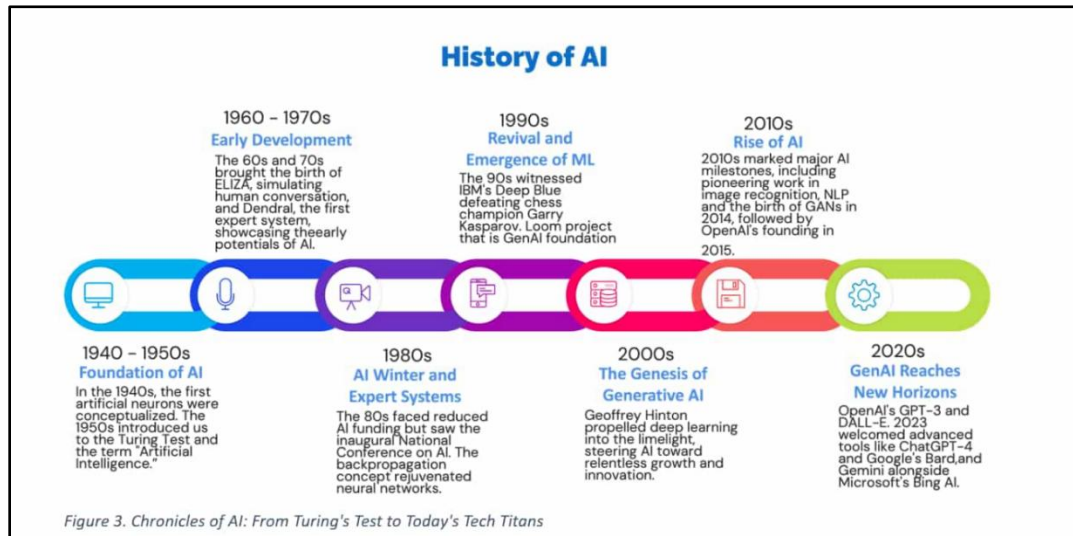


Figure 5.1 [1] (pending approval)

5.3 Advantages & Disadvantages of AI

AI tools such as ChatGPT, Copilot, and Gemini have become valuable resources for engineering students and working professionals. These tools provide immediate assistance with brainstorming, coding, documentation, and research. However, with the benefits, there are notable limitations and issues that these tools may generate.

Advantages:

- **Efficiency and Speed** – Computers can process information much quicker than human capabilities. This helps the engineer complete work related to documentation, coding, and analysis.
- **Knowledge Accessibility** – The immediate accessibility of various technical knowledge domains enabled by ChatGPT and other LLMs systems act as an on-demand learning assistant; especially those connected to the Internet.
- **Idea Generation** – The role of artificial intelligence in aiding design thought processes, problem-solving solutions, or writing templates may speedup the initial stages of a project.
- **Error Detection and Debugging** – In programming-related work, artificial intelligence may analyze data and code to detect syntactic or logical mistakes and offer solutions almost immediately.
- **Improved Communication** – The capabilities of AI increase the effectiveness of communication in reports, presentations, and business communications.

Disadvantages:

- **Accuracy and Reliability** – The outputs from the models may at times be erroneous or antiquated. All outputs should be checked against confirmed sources.

- Over-Dependence – Over-reliance on the capabilities of AI may result in reduced levels of critical thinking and problem-solving capabilities.
- Data Bias and Limitation – As the training data of an AI model includes data collected from the internet, biased data may affect the outcome.
- Ethical and Academic Integrity Concerns – Use of AI without disclosure in academic or professional settings may pose authorship/plagiarism and/or integrity considerations.
- Lack of Contextual Judgment – These systems and technologies tend to lack real-world understanding. The system may misinterpret an ambiguous set of instructions or fail to consider certain limitations even when specifically mentioned in the prompt.

If it is used properly, AI can be an important addition, rather than a replacement for engineering knowledge. The objective here should be the inclusion of AI in such a manner that it increases both efficiency and intelligence and at the same time retains human control. To ensure that AI functions as an addition and rather than a subtracting factor in engineering research and design, the team needs to adhere to certain rules of use.

5.4 Rules and Expectations of Group AI Usage

To ensure the integration of such artificial intelligence resources as ChatGPT and other forms of generative technology is used as an educational supplement and not as a replacement within the realm of student academic and intellectual activity, a set of principles and rules within these groups has been created. These principles were designed to ensure academic integrity as well as technical knowledge and personal accountability throughout our Senior Design project.

5.4.1 Purpose of Guidelines

The goal of these guidelines is to ensure that AI tools are used in such a way that they increase productivity and understanding and do not act as substitutes for individual effort. The team appreciates the benefits and privilege of using AI tools in the domains of writing, coding, as well as research.

5.4.2 Accepted Uses of AI

The following tasks may be performed by AI tools:

- Research Assistance – Gathering background information, standards, or component data to support design decisions.
- Coding and Debugging Support – Offering examples of code syntax or algorithms to refer to, followed by testing by the team.
- Brainstorming/Ideation – This entails the generation of design possibilities or ideas for visualization that the group can work on.
- Technical Summarization – Shortening complex technical information to facilitate comprehension and a better understanding.

In each of these instances, the role of AI is to supplement human capacity rather than substitute for engineering judgment, experimentation, and decision-making.

5.4.3 Unacceptable Uses of AI

To maintain academic integrity and ensure that all deliverables represent true teamwork, the following uses of artificial intelligence are prohibited:

- Uncredited or Blind Submission – Plagiarizing AI writing without reviewing or citing it in reports or presentations.
- Fabrication of Data or Results – The use of AI to generate experimental data, test results, or system measurements that were not actually validated.
- Bypassing Design Responsibility – Delegating the design of the AI system to the algorithm on technical design decisions such as circuit design.
- Academic Dishonesty – Passing off an AI's effort as original work by human students or as an answer to assignments that will be graded.

5.4.4 Verification and Accountability

All AI generated content included within the project documentation needs to be checked and validated by at least one member of the team.

In instances where the AI system assists in design, the role of the system should be properly cited either in the Appendix.

5.4.5 Team Ethic and Learning Value

The group stresses that the implementation of AI should be used as an educational facilitation technique and not as a shortcut. The eventual use of the AI should be as an approach by each member of the team to apply it to gain better insight into new concepts, alternative solutions, and/or to better our understanding of theoretical principles. Through these rules, we can facilitate transparency, fairness, and educational value within its operations while properly utilizing AI as an emerging technique within the engineering design process.

5.5 Xander Levine's AI Use

5.5.1 Flutter Build/Debug Assistance

Please refer to Appendix E for this prompt/conversation [1a] [1b]

Task: Resolve Android Gradle lock issue and establish rapid UI iteration (hot reload) for emulator.

AI contribution: Provided steps to terminate stale Gradle daemons, clear build caches, and enable flutter run to hot-reload; recommended VS Code shortcuts.

Verification. Successful rebuild on emulator; confirmed hot reload without full relaunch.

Benefit. Shortened iteration loop for GUI changes; improved developer productivity.

Limitation observed. Some platform-specific commands required adaptation to local environment.

Policy alignment. Acceptable per 5.4.2 (debugging aid) with hands-on verification (5.4.4)

Gradle daemons (brief): Background JVM processes Gradle uses to speed builds. When a daemon crashes, is orphaned (IDE closed mid-build, sleep/hibernate), or multiple builds contend, it can leave a lock file (android\gradle\noVersion\buildLogic.lock) that blocks new builds until daemons are stopped, and the lock is removed.

5.6 Noah Bixby's AI Use

Please refer to Appendix E for the full prompt/conversation for each case: [2a], [2b], and [2c].

5.6.1 Case [2a] – MCU Buck Regulator Selection & Sizing

Task: Identify and size a high-efficiency buck regulator to convert ~34.2–39.6 V input to the MCU logic rail (≈ 3.6 V @ ≥ 620 mA), including component selection (L, CIN/COU, diode/MOSFET where applicable) and quick stability checks. (Context: 6.1.1 MCU Regulator design.)

AI contribution: Provided a shortlist of suitable TI buck controllers/regulators, a first-pass inductor ripple target (20–40% of IOUT), capacitor ESR considerations, and a quick checklist for WEBENCH cross-verification and bench bring-up (probe points, load-step sanity, thermal check).

Verification: Cross-checked values in TI WEBENCH and confirmed the LMR51625-based solution meets the 3.6 V/0.62 A requirement with acceptable ripple; design integrated into Section 6.1.1.

Benefit: Accelerated part screening and first-order sizing; reduced iteration time before detailed simulation/layout.

Limitation observed: AI suggestions required datasheet validation (limits, compensation notes) and tool corroboration; parasitic/layout still determined by bench work.

Policy alignment: Acceptable per 5.4.2 (research/debug assist) with hands-on verification per 5.4.4.

5.6.2 Case [2b] – Class-D Mid/High Amp Supply Details (TPA3116D2 IVDD & Edge Conditions)

Task: Clarify the TPA3116D2's low-voltage rail (IVDD) expectations under 50% duty and worst-case PVDD and determine whether an auxiliary LDO/rail is needed versus deriving locally. (Context: amplifier and power sections.)

AI contribution: Summarized typical IVDD current envelope vs. operating mode, highlighted when a quiet local LDO can reduce control-loop noise coupling, and suggested decoupling topology (local bulk + HF ceramics near IVDD pin) with star return to PVDD ground.

Verification: Cross-read with TI datasheet guidance; benchtop plan added for scope checks on IVDD ripple during burst content and clip-on transients before finalizing BOM.

Benefit: Reduced risk of intermittent amp behavior and control-rail noise; clearer decision path on whether to keep IVDD isolated.

Limitation observed: AI lacked device-specific numeric limits; final values come from the TI datasheet/app note and bench data.

Policy alignment: Acceptable per 5.4.2 with verification per 5.4.4.

5.6.3 Case [2c] – 36 V Rail Strategy from 34.2–39.6 V Source (Buck vs. Fixed-Output PSU)

Task: Evaluate options to supply a clean ~36 V system rail from a variable source (34.2–39.6 V), comparing: (1) direct high-side buck to 36 V, (2) pre-reg + point-of-load, or (3) a ready-made 36 V PSU.

AI contribution: Outlined trade-offs among efficiency/EMI/thermal complexity and suggested using a COTS 36 V supply to de-risk schedule, with downstream local regulation for sensitive loads; provided a short checklist for line/load regulation and surge behavior.

Verification: Team BOM now reflects a 36 V COTS unit (LRS-600N2-36) as the primary rail, with local regulation where needed.

Benefit: Lowered development risk and integration time on the high-power rail; predictable compliance and thermal characteristics.

Limitation observed: Slight reduction in peak conversion efficiency vs. a fully custom buck; mechanical mounting and airflow now factor into PSU choice.

Policy alignment: Acceptable per 5.4.2 with documentation/bench validation per 5.4.4.

5.7 Jordan Riley's AI Use

When researching Bluetooth modules, coming across modules that would meet the requirements for our system was extremely difficult as there was no filtering option to find the protocol required for audio streaming. This resulted in frustration and delays in our research until ChatGPT was consulted. With the prompt found in Appendix E, [4a], ChatGPT was able to find multiple Bluetooth modules that fit our needs. One of which was the BM64, the part eventually chosen for our design. ChatGPT was then prompted with [4b] and gave more examples that were available through our main parts provider, DigiKey. The FSC-BT1026 was one of the modules that was given through this prompt, and it turned out to be the most capable device with a long list of features. This was an extremely powerful module but turned out to be too powerful for our application.

Most of my AI use has been to get a starting point for research and as a search engine like how one would use google. This has turned out to be very beneficial in Senior Design as less time is spent on finding sources of information or parts on my own. Having ChatGPT give me options to start with, answers to questions can be found faster through sources cited by ChatGPT. However, it is imperative that the sources are checked thoroughly for credibility.

5.8 Matias Segura's AI Use

5.8.1 [3a] Parallel DAC Implementation

When the STM32H735ZG was selected as our MCU, we had to consider adding an external DAC to achieve 16-bit resolution, since the STM32H735ZG only comes with 2 x 12-bit DACs. At one point we also considered the possibility of trying to combine these two channels and “merge” them together to create a 24-bit DAC. We weren't sure if this was possible, because the datasheet did not contain very straightforward information. So, I decided to ask ChatGPT to see if this possibility was feasible; I provided the information for the DAC from the STM32H735ZG's datasheet and asked if it was possible to achieve what we wanted. The exact prompt can be found in Appendix E, [3a] along with ChatGPT's response. Its response helped me to realize that implementing parallel usage of the DACs in the MCU was going to add a lot of software complexity. This led to the decision that adding an external DAC to achieve the desired bit resolution would be a better and faster alternative.

5.8.2 [3b] DAC Recommendation

After narrowing down the search of the external DAC module to 4 different products, I followed up the conversation from **5.8.1** and asked the tool to summarize each DAC and give me the best one based on characteristics. The exact prompt can be found in Appendix B, [3b] along with ChatGPT's response. ChatGPT's response was very concrete and precise, as it described in detail each DAC and suggested why it might be a good option and why it might not. The use of AI in this case helped to summarize some details that would have taken more time to compare between them all. It also helped to see why one

of the DAC that was selected for comparison was not a strong option; this shows the tool's capability of giving a different insight than the one we could have given.

Chapter 6 – Hardware Design

6.1.1 Regulators

To create these regulators, we made use of the Texas Instruments WEBench Power Designer tool, where we inputted the necessary input and output specifications, regarding the power supply for the input and the output, which was denoted for either the MCU, subwoofer amplifier chip, subwoofer amplifier, and the mid/high speaker amplifier.

6.1.2 Digital Circuitry Regulator

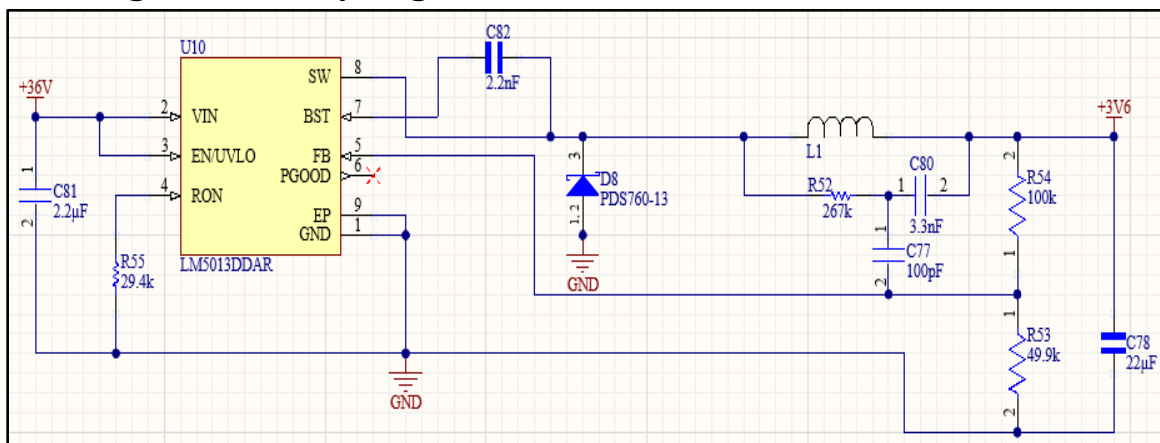


Figure 6.1 +3.6V Regulator Schematic

To provide the necessary power needed for the logic of our chosen MCU, Bluetooth module, and DAC, a Buck converter was designed. Assuming a 35.64V – 35.36V, and a necessary 3.6V with 1A output, the figure above shows the necessary regulator configuration. For this regulator the TI LM5013DDAR integrated circuit was used. By rapidly switching MOSFETs and using an LC filter it can provide the smooth 3.6V output needed.

6.1.3 Senior Design 2 Implementation – Analog and Digital 3.3V Regulator

For the +3.6V regulator we ended up dividing this circuit into two separate boards, along with scaling the voltage down to +3.3V. One +3.3V was dedicated for digital logic circuits and another dedicated to analog logic circuits. The difference between the two split boards was that the analog variation contained an LDO on the output that stabilized the output ripple, since analog logic components are highly sensitive.

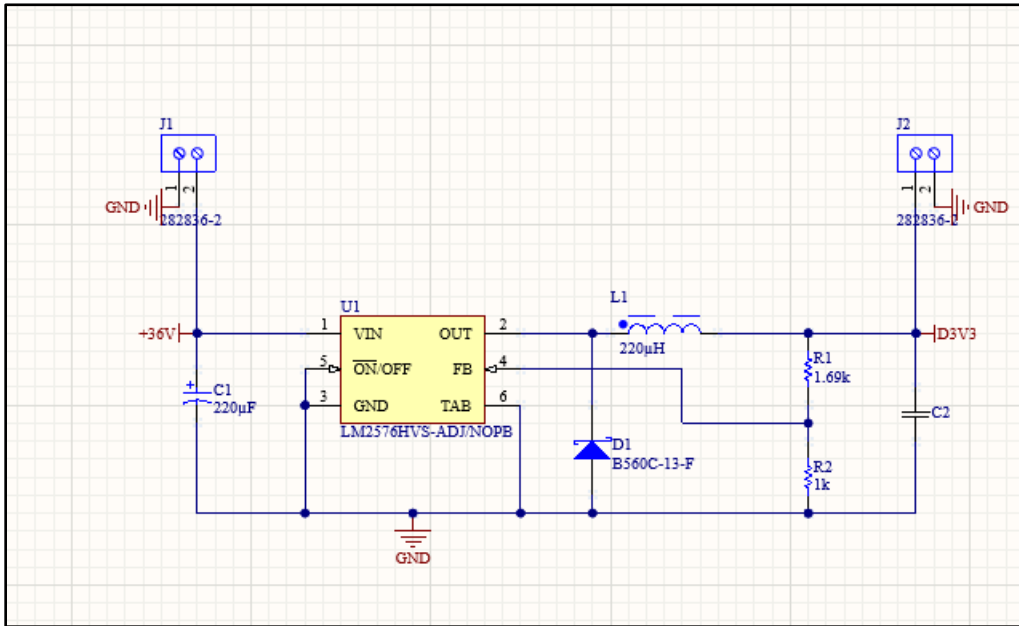


Figure 6.2 +3.3V Digital Regulator Schematic (Updated)

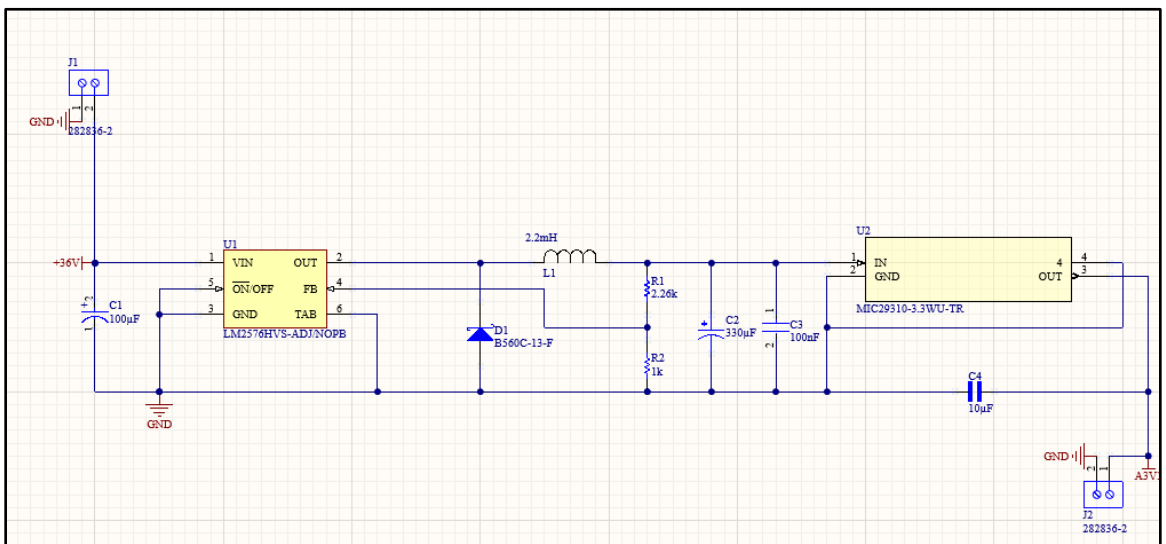


Figure 6.3 +3.3V Analog Regulator Schematic (Updated)

6.1.4 +5V and -5V Regulators

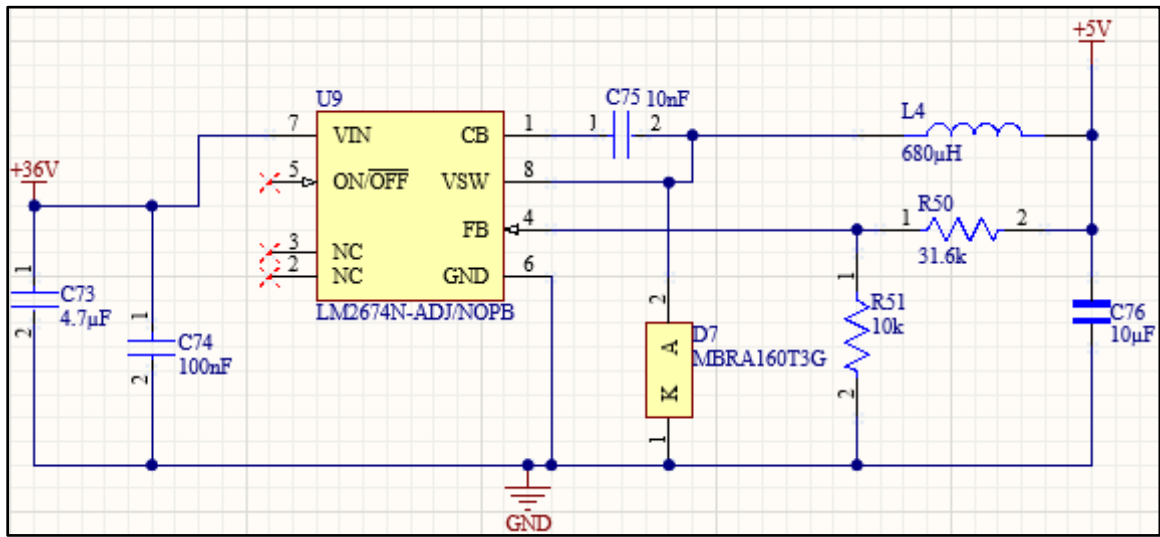


Figure 6.4 +5V Regulator Schematic

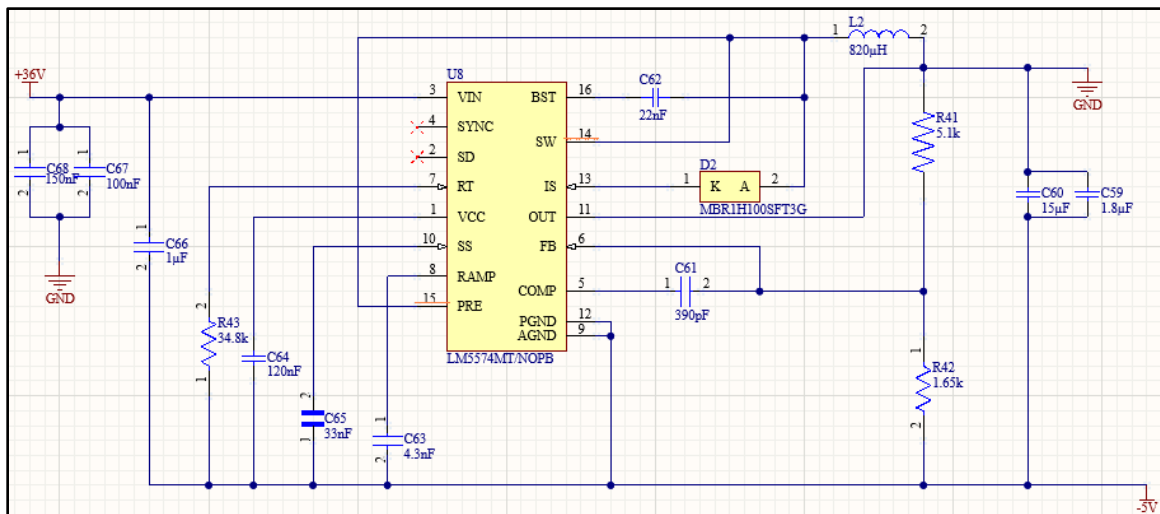


Figure 6.5 -5V Regulator Schematic

The $\pm 5V$ regulators are needed to drive the various op-amps on the board. For the +5V regulator, the TI LM2674N-ADJ/NOPB was used and for the -5V regulator, the LM5574MT/NOPB was used. These two regulators take the 36V input from the power supply and can output 100 mA each.

6.1.5 Senior Design 2 Implementation - $\pm 5V$ Regulator

For our +5V regulator we also changed the IC to match the +3.3V regulators, along with +12V regulators, which again was the LM2576. Like the analog +3.3V regulator since the

+5V is dealing an analog component we included an LDO on the end of it. Regarding the -5V regulator the board we designed unfortunately did not work, and neither did our redesigned -5V charge pump. As a result, we had to create a makeshift circuit that does not have a PCB or schematic layout.

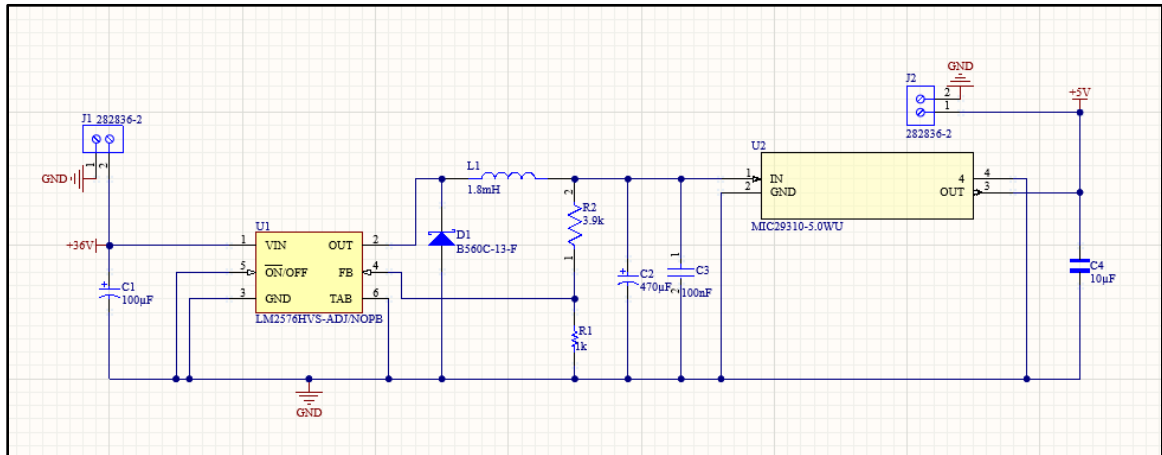


Figure 6.6 +5V Regulator Schematic (Updated)

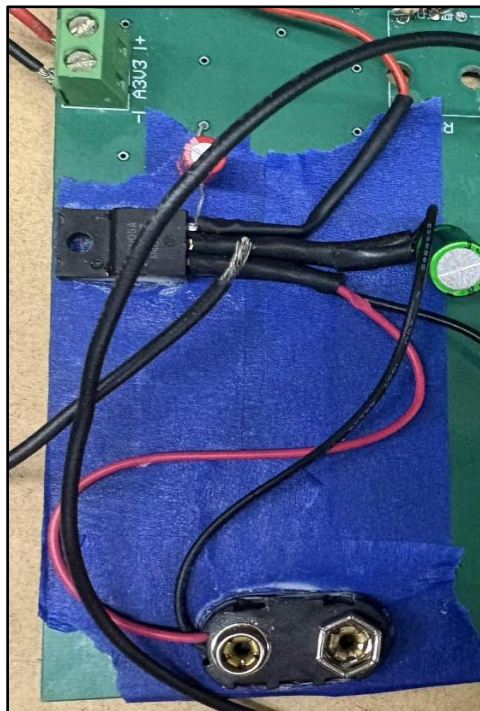


Figure 6.7 -5V Regulator (Updated)

6.1.6 +12 Volt Regulator

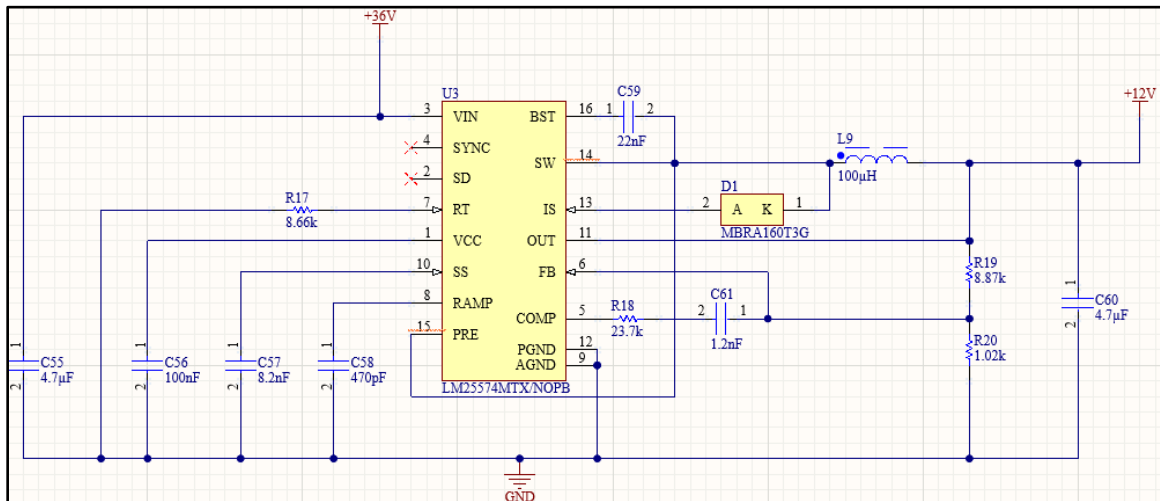


Figure 6.8 +12V Schematic

Like the previous regulators created, this regulator was designed to take the 36V input from our power supply to create a 12V and 100mA output. This regulator would be connected to the subwoofer amplifier to run the on-chip peripherals, such as the transistor drivers and digital circuit regulators. Like the MCU regulator, this is also a DC/DC Buck regulator. In this instance, we are using a different IC but sticking with Texas Instruments, using the LM25574MTX/NOPB model.

6.1.7 Senior Design 2 Implementation - +12V Regulator

We ended up changing the layout of the +12 volt regulator to incorporate an IC that was known to work, that being the LM2576, and fit all the specifications needed by the project. Below is the schematic for this regulator.

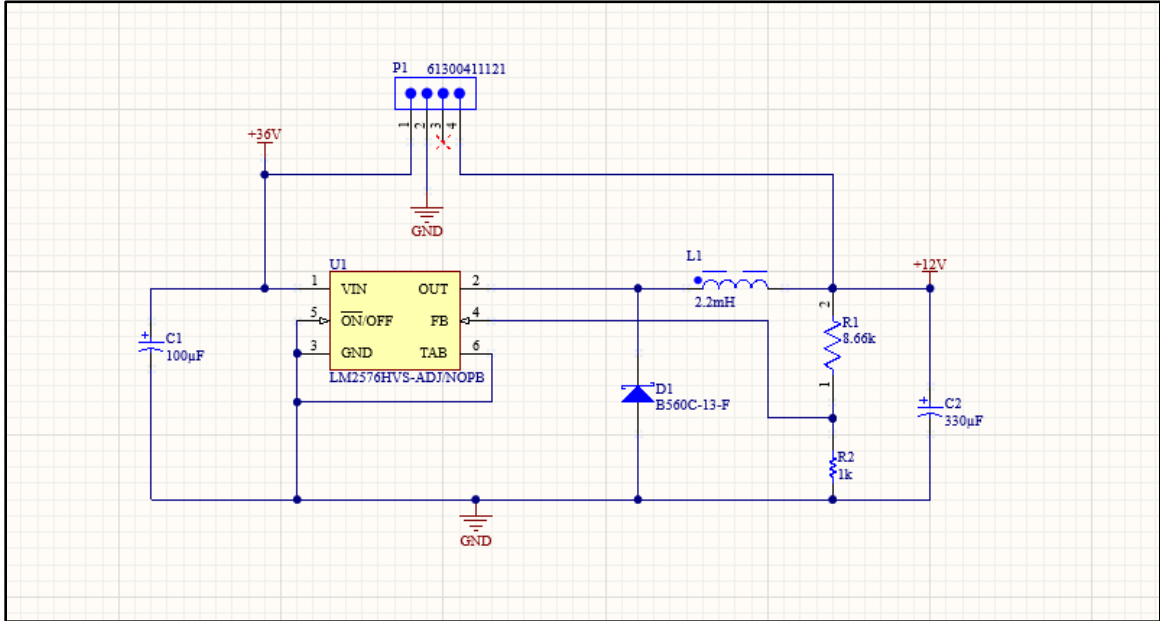


Figure 6.9 +12 Volt Schematic (Updated)

6.1.8 Mid/High Amplifier Regulator

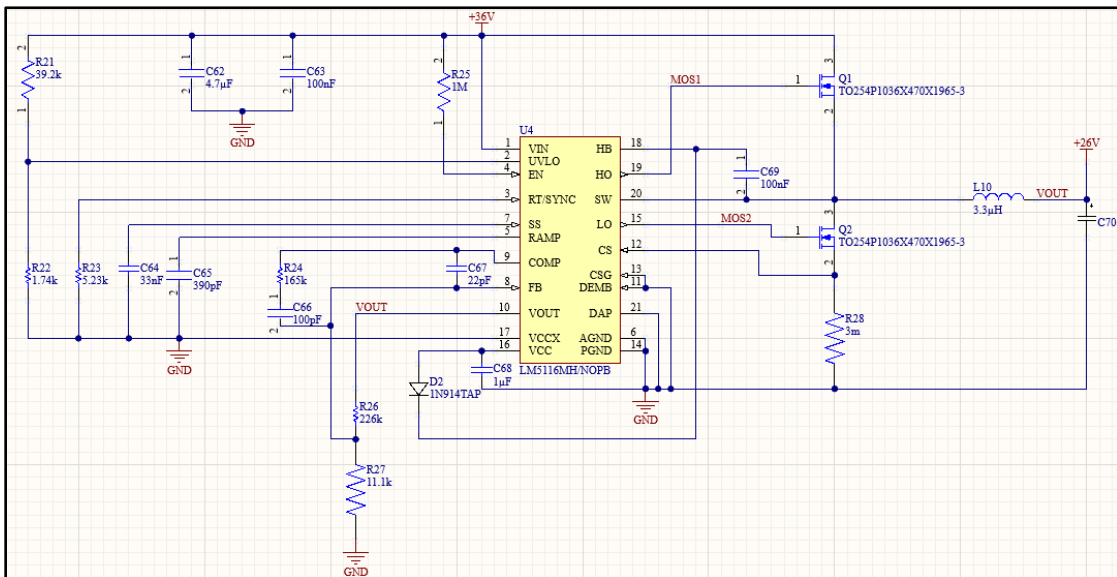


Figure 6.10 Mid/High Amplifier Regulator Schematic

The next designed regulator was one that was to be connected to the mids and highs amplifier itself. Keeping with the same input voltage seen in the past, two regulators this one needed a 26V and a 9.4A output. This design is as well a DC/DC Buck regulator. The

IC used in this configuration was the Texas Instruments, or TI, LM5116MH/NOPB controller. Another key detail worth noting is that this IC is nonsynchronous which means it must make use of external MOSFETs, and a diode to handle the high-power levels, as seen in the schematic with the Q1 MOSFET going from 36V to the output and Q2 going from output to ground.

6.1.9 Senior Design 2 Implementation – Mid/High Regulator

With the +24V regulator the design did work, however, the first iteration did not work due to a design rule issue within Altium, so all our through-hole components were shorted to ground. This issue was fixed, but near to the deadline our project our FETs did blow up, so we had to create a makeshift regulator, like the -5V, that used an adjustable LM2576 that was rated for 3 amps.

6.2 Amplifier Schematics

6.2.1 Subwoofer Amplifier

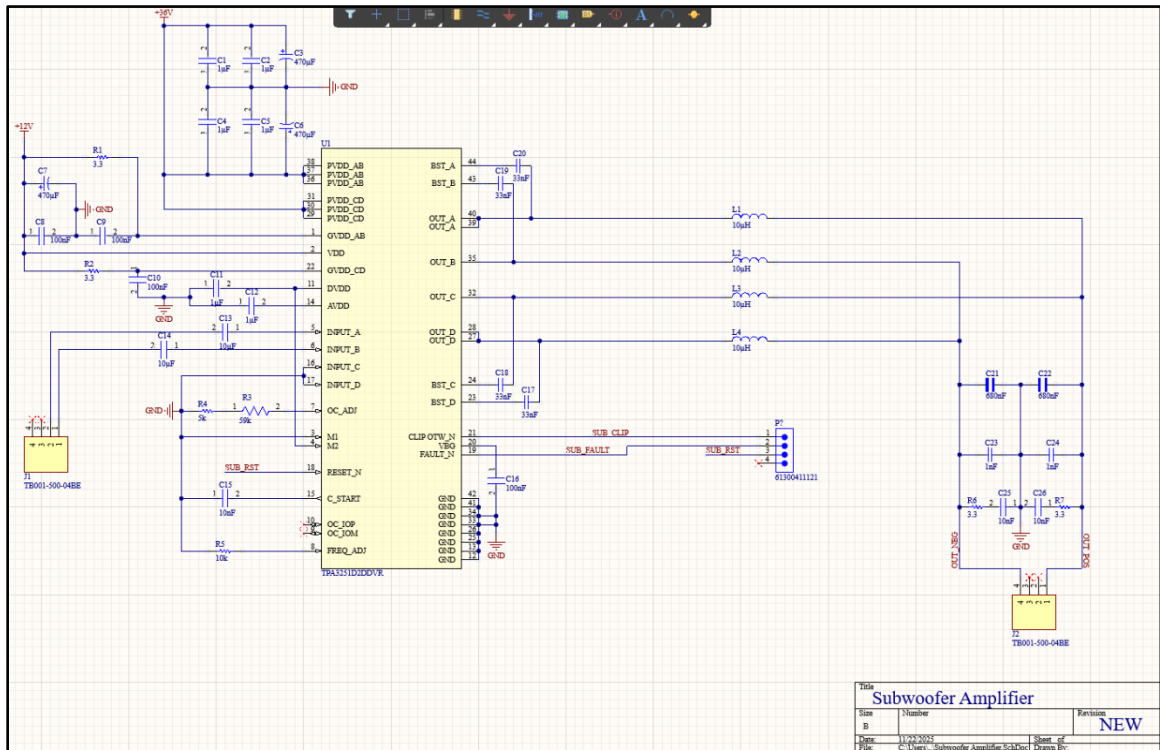


Figure 6.2 Subwoofer Amplifier Schematic

The M1, set low, and M2, set high, pins in this configuration have it designed as a Typical Differential (2N) Parallel Bridge-Tied Load (PBTL). Since this amplifier's frequency range is operating at a different one than mid/high amplifier, we designed this chip to run in master mode, meaning it runs on its own internal clock and defines its own PWM frequency. In a PBTL configuration the 4 sets of output are tied in pairs of two, in this case

the A/B and the C/D output channels are tied. OUT_A is put in parallel with OUT_C and OUT_B is put in parallel with OUT_D. Because of this we can create a differential pair that delivers more current and in turn more power to the subwoofer than a typical Bridge-Tied Load (BTL). A benefit of setting the chip in PBTL mode is that it decreases the issues faced with electromagnetic interference, provides sufficient noise rejection, and increases output power capability.

The DAC's output is connected through the J1 input terminal where INPUT_A is the positive reference and INPUT_B is the negative reference as the amplifier is in the 2N mode. Both are analog inputs and are AC-coupled with the capacitors to remove any DC bias from the signal to ensure the input signal never reaches the input pins' maximum and minimum voltage.

The /FAULT and /CLIP_OTW connect to the GPIO input and are responsible for having the MCU monitor the amplifier for any errors such as temperature issues, too much current, etc. /RESET pin is connected to the MCU output and oversees resetting the amplifier in the event of fault recovery or startup.

The design for this amplifier is based on the configuration seen in Figure 27 in the data sheet which is a typical application of this design scheme for the TPA3251D2DDVR.

6.2.2 Mid/High Amplifier

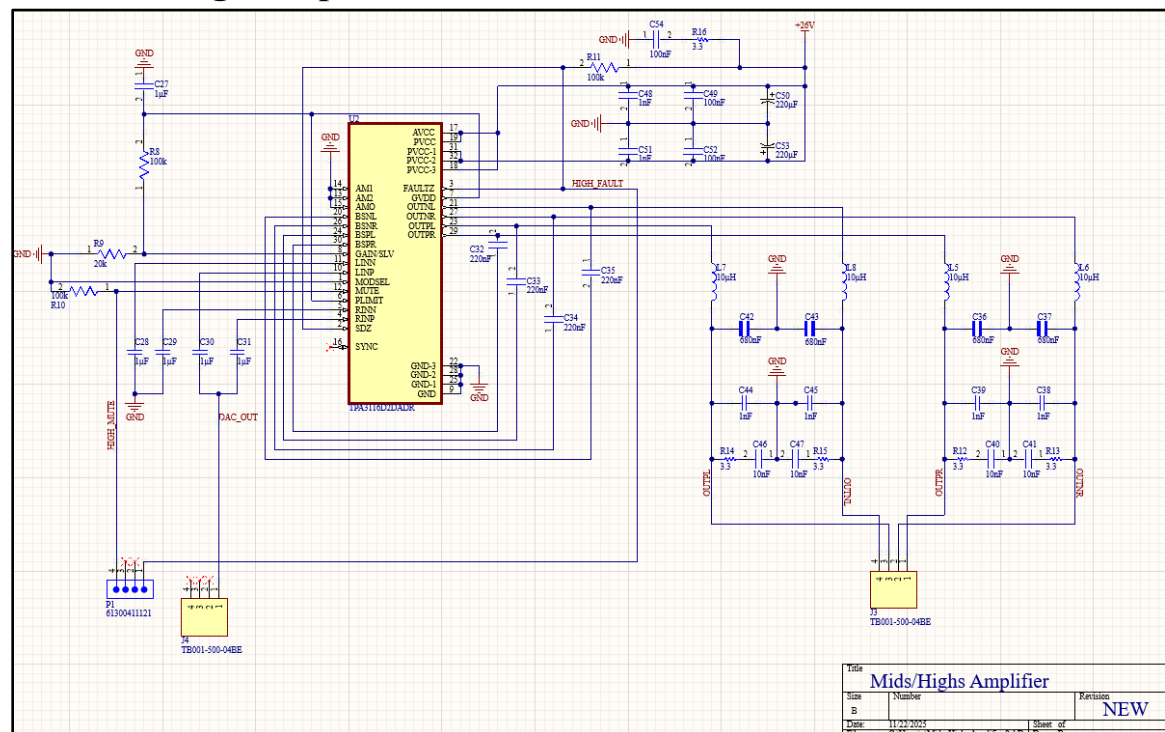


Figure 6.3 Mid/High Speaker Amplifier Schematic

This schematic captures our design for the mid/high speaker amplifier. Like the subwoofer design process, careful analysis of the TPA3116d2 data sheet was paramount. Unlike the subwoofer design, this setup puts the amplifier in single-ended BTL mode, which can be seen in the configuration of OUTP and OUTN. In addition, we can see that through the R7 and R8 resistors we are placing the amplifier in master mode. C27, C28, C29, and C30 connect the outputs and bootstrap pin, providing passage for the drive voltage provided by the FETs.

The power supply is inserted by the PVCC and like the subwoofer amplifier is decoupled for each half-bridge.

Another similarity between the high/mid amplifier and the subwoofer amplifier is that before reaching the speaker output stage power is passed through an LC network which smooths the switching output into a useful audio signal.

The FAULTZ, SDZ, and MUTE pins are controlled by or report to MCU to allow the MCU to see the condition and control the amplifier. The FAULTZ reports faults caused by overheating, short-circuiting, and other various conditions to the MCU where the MCU would then intervene. An additional design spec worth noting is the SDZ and FAULTZ pins being shorted, as this allows the amplifier to restart itself once a fault has been cleared.

6.2.3 Senior Design 2 Implementation - Amplifiers

For both amplifiers that we designed, both amplifiers worked for our final design and no adjustments were needed to be made. However, troubles did arise when fabricating our boards as ESD killed the ICs on both the mids/highs amplifier and the subwoofer amplifier.

6.3 Digital Schematics

Due to our design having both analog and digital signals the digital devices have their own section. This section covers all components that will be implemented on the digital board.

6.3.1 Main MCU Schematic

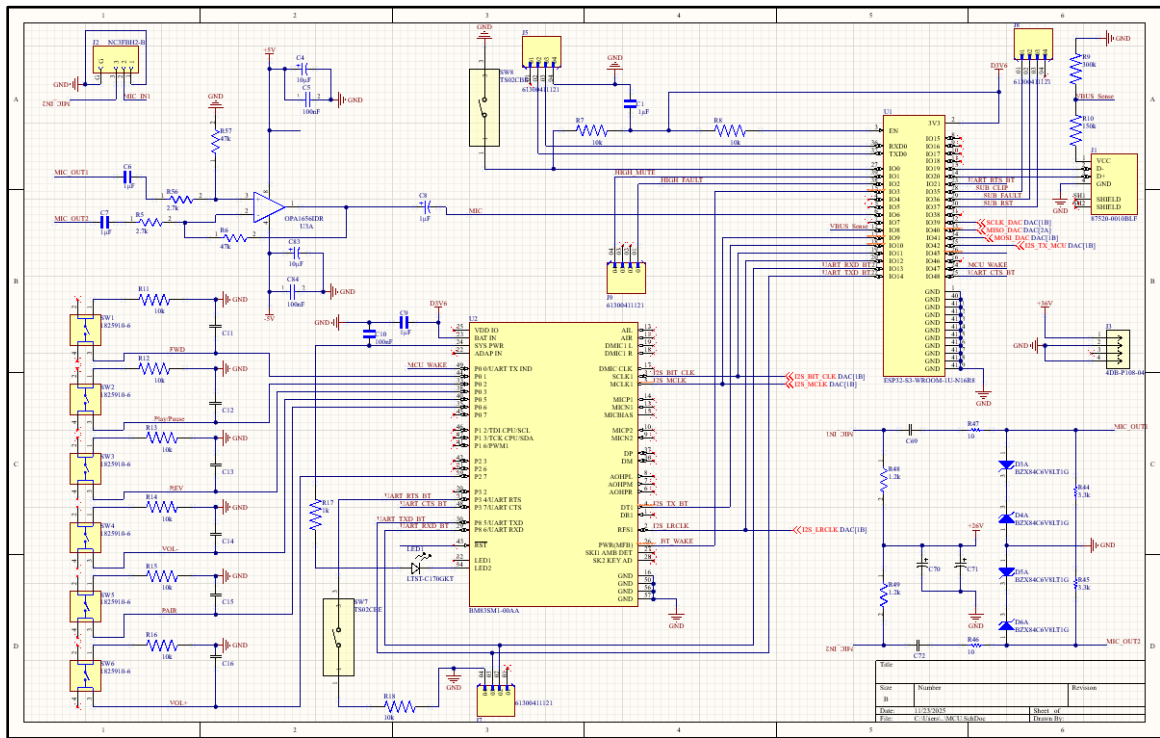


Figure 6.4 Digital Board Schematic 1

The first schematic for our digital board illustrates almost all connections that will be made for this board. The schematic includes the ESP32-S3-WROOM-1U-N16R8, BM83, and XLR microphone input.

Starting with the ESP32, most connections start or end here. This is because the ESP32 acts as the brain of our system. The ESP32 is responsible for sending and receiving commands to and from the peripherals, as well as processing the data that is received to make the necessary adjustments to the audio signal. Additionally, the ESP32 is connected to four input ports, and a switch. One of the ports act in conjunction with the switch as the switch is used to put the device into flash mode, when the switch is off the pin IO0 is pulled high through a pull up resistor indicating normal operation. When the switch pulls the pin low, placing the ESP in flash mode through UART. This is where input terminal J5 comes into play as this connection is used to connect a USB-to-UART flashing tool to the board to flash the ESP32 with our program. Two of the input ports are used to send and receive signals to and from the amplifier board. The final input port is the USB connector; this is where the phone will connect to the system to send commands or music.

The ESP32 also has an RC timer on the EN pin to set the startup time to about 10ms as recommended by the datasheet. No decoupling or bulk capacitors were required for the power pin because those are integrated on the device.

The BM83 is implemented with six total buttons, a switch, an input port, and connections to the ESP32 and DAC (see Figure 6.9).

The buttons for the device are implemented to allow the user to send play/pause, fast forward, rewind, volume up and down, and pair commands to the user's device. These commands can be set up in Microchip's provided config tool.

The switch and input port are used the same way the ESP32 uses them together. When the switch is off, the pin is pulled HIGH by internal pullup resistors, telling the device to enter normal operation mode. When it is on, the BM83 is in pair mode, allowing a USB-to-UART flash tool to be placed on the J7 header pins to load the configuration built using Microchip's config and isUpdate tools. The J7 header pins are connected to the UART pins on the BM83, which are also used to communicate with the ESP32. As a result, when flashing the BM83, the ESP32 must be loaded with a program that places the connected pins in a high-Z state to avoid disrupting the signal.

The BM83, in addition to the UART communication, communicates with the ESP32 through the I2S protocol. This has four total connections, the bit clock, master clock, left/right clock (also called word clock), and the transmit data line. The receive data line is not needed for this as the BM83 will not be used to transmit any audio.

The BM83 also has an LED that can be used to communicate to the user what state the device is in.

The XLR connection (J2, top left) for the microphone is the only analog signal that will be used on this board outside of the DAC's output. The purpose of this is to take the microphone's differential signal and process it in the ESP32. To utilize the XLR connection, a phantom power supply must be used. The circuit for this is seen in the bottom right corner of the schematic. The phantom power supply utilizes the 26V power from the mids/highs amplifier's regulator and passes it through two resistors of $1.2\text{k}\Omega$ as recommended by the Wikipedia page on phantom power, this is also subsequently where the circuit we used is from [71]. The 26V power is supplied to pins 2 and 3 for the XLR cable and the EMM-6 microphone uses it to pass the audio signal onto the bypass capacitors to remove the DC biasing. The phantom power circuit then transitions to a clamping circuit to ensure the signal does not create too much voltage swing between the pins.

The differential outputs of the phantom power circuit are then passed to a different amplifier (U3A) to mitigate the noise picked up through the microphone's wires and turn the signal into a single-ended signal so the ESP32 can process the received frequency response. The gain of the amplifier was set to amplify the microphone's signal enough to read on the ESP32's ADC and the rail voltages were set $\pm 5\text{V}$ to clip the signal at a point that avoids damage to the ESP32.

6.3.2 DAC Schematic

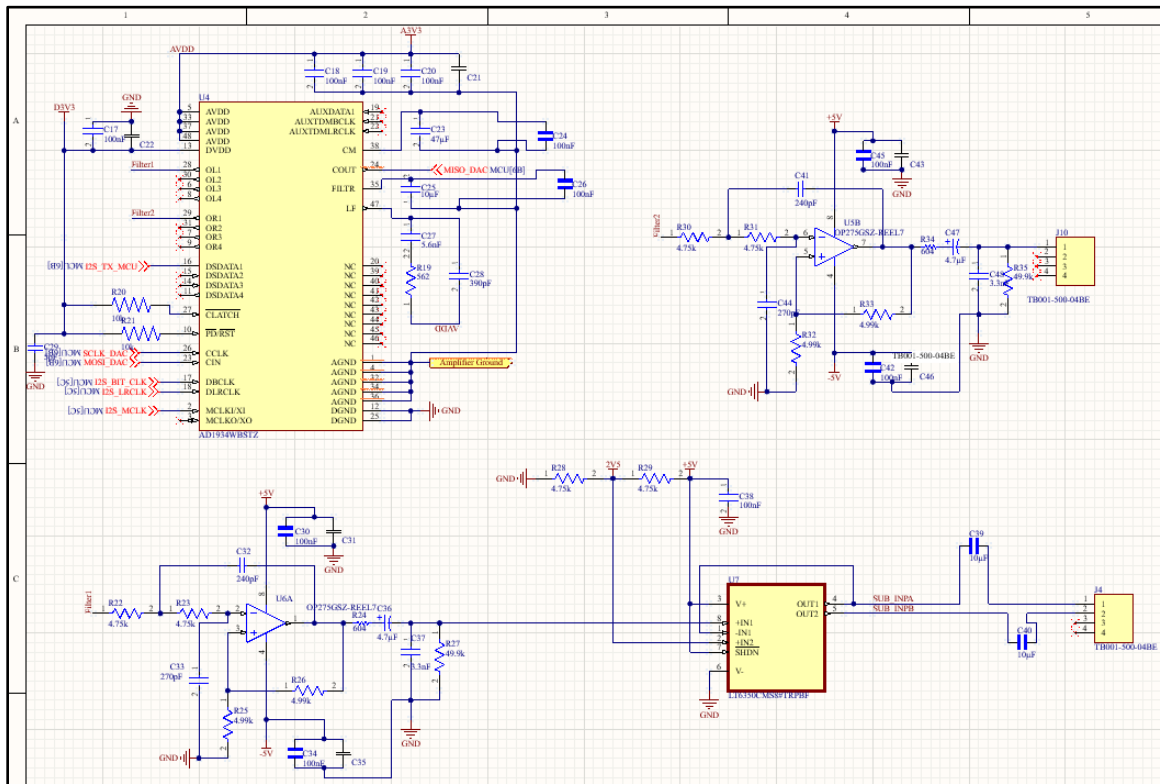


Figure 6.5 DAC Schematic

The purpose of the DAC is to take the audio signal, processed by the ESP32 and sent over I2S, and convert it to an analog signal that is usable for the amplifiers. The DAC will be acting as a slave for the I2S communication.

By referencing the data sheet for the AD1934WBSTZ the AVDD and DVDD, needing to isolate these two pins was crucial for ensuring interference between these two pins was kept to a minimum. We can see for the AVDD it contains 100 nF and 10 uF capacitors to decouple these properly. In addition, for the FILTR the data sheet says the board requires a 10 uF capacitor, which explains its placement.

As a result of the DAC being operated in slave mode, we are not using the “hardware mode”, meaning that the SPI configuration must be integrated. By connecting the CLATCH pin to the ESP, the ESP will pull the wire low allowing for the SPI communication to occur. Furthermore, by connecting SDATA, MISO, and SCLK to the ESP we further show the SPI configuration is being used. With SCLK connected we are essentially creating the SPI clock input.

When the data is sent over I2S, it has two different signals, the left and right signals. One side will carry our subwoofer signal and the other will carry the mids and highs signal. This is the most efficient way to process our data as only one transmission line is required.

After the DAC converts the I2S data to an analog signal the outputs are fed into filters to clean up the switching noise be ready for amplification. The first filter (top right) is for our mids/highs signal and can be passed directly to the amplifier. However, the low frequency signal for the subwoofer needs further processing by a single-ended input to differential output converter, as the TPA3251 can only except differential inputs in the mode that we have selected. This conversion was done with the LT6350CMS8#TRPBF.

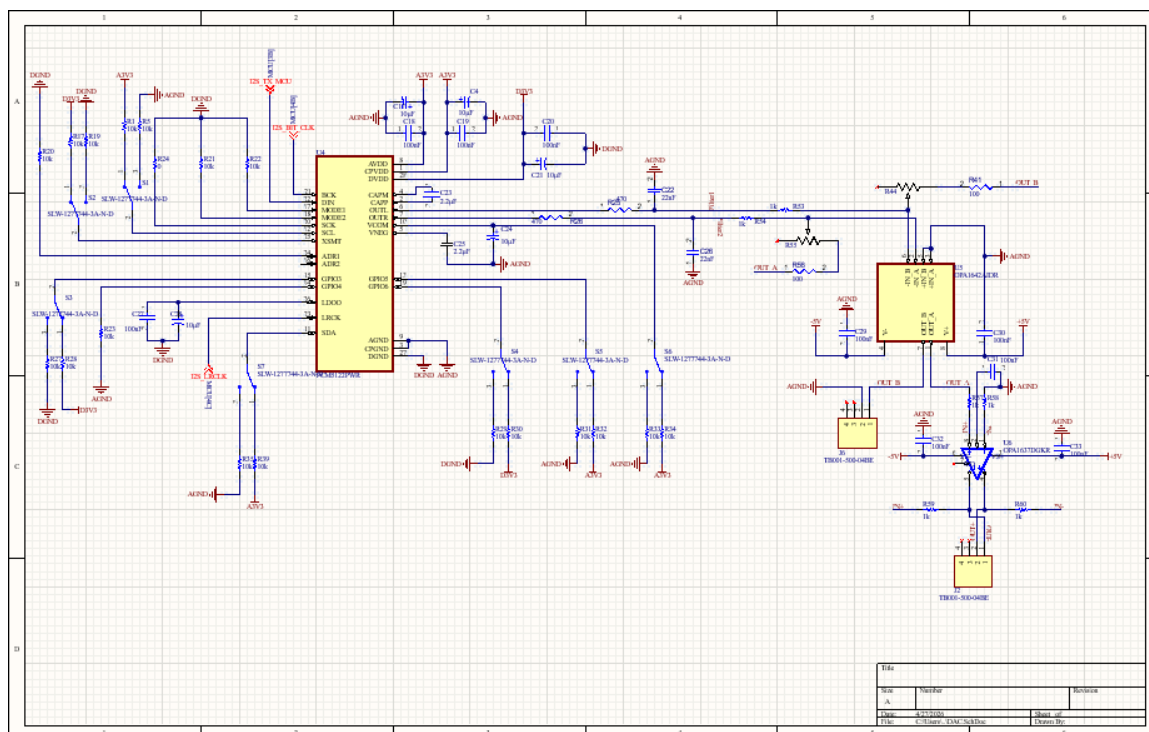
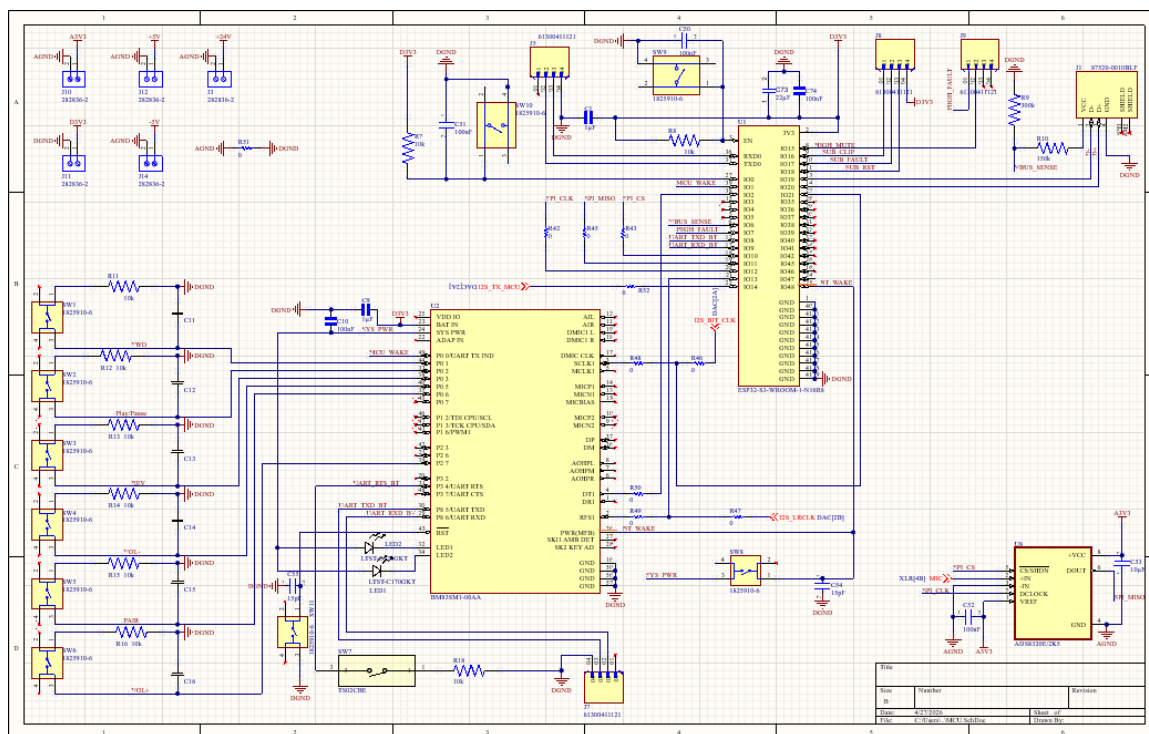
After the filtering and conversion, the signal is passed to speaker wire terminals to pass the audio signal from the digital board to the amplifier board.

6.3.3 Senior Design 2 Implementation

The digital board produced the most headaches during fabrication and testing as the first version of our design had the rules set wrong, so all hole components were shorted to ground during manufacturing.

After setting the rules properly, the second version included the AD1934 and this part did not work for us. Additionally, when implementing the full phantom power circuitry for the microphone, introduced noise prior to the ADC.

For version 3 of the digital board we moved to the phantom power circuitry tested in Senior Design 1 as we knew it worked, the ADC was switched to the PCM5112, and an ADC was added to remove the unwanted noise introduced by the ESP32 ADC. Additional changes were made to this board after the board arrived as an SPI pin needed to be swapped for the ADC to operate at the required sampling frequency. The final digital board schematics are shown in the figures below.



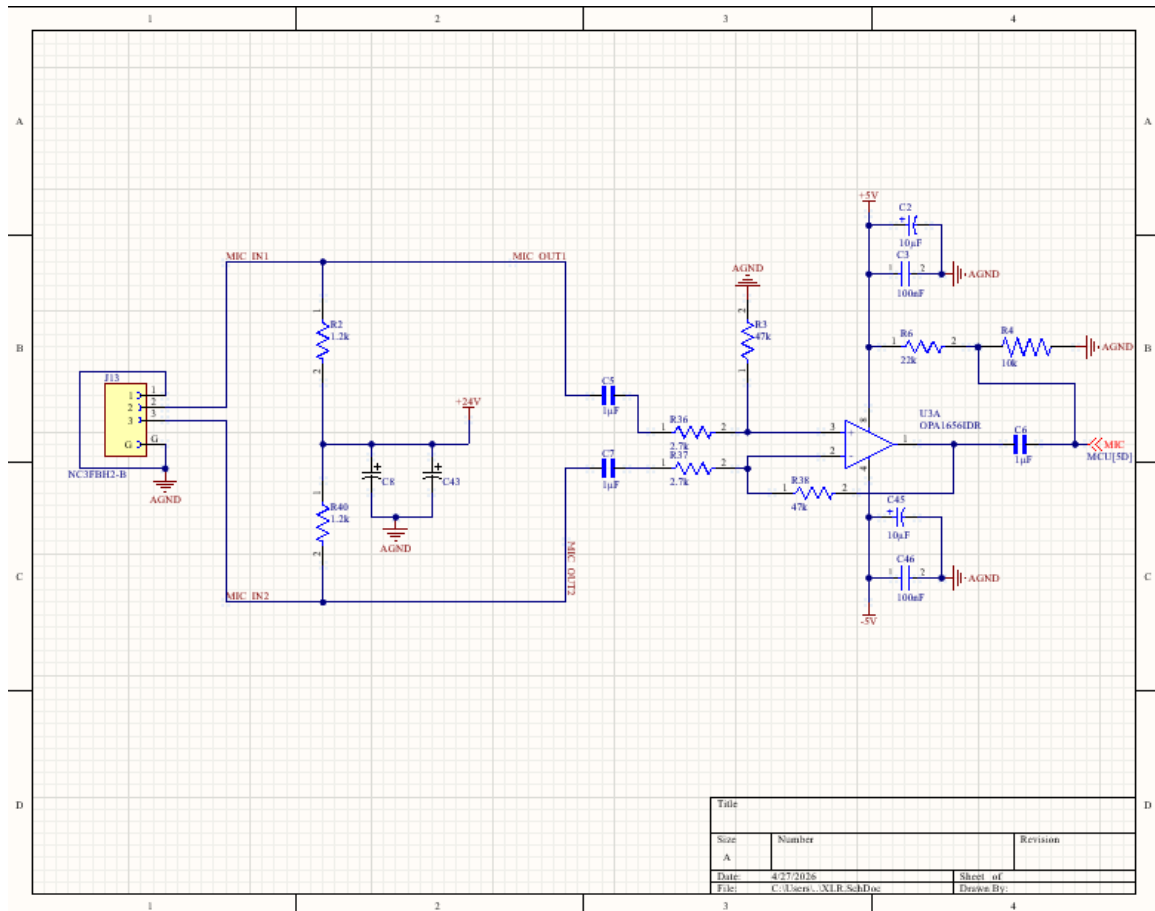


Figure 6.12: Final XLR Schematic

Chapter 7 – Software Design

7.1 Introduction to GUI and Mobile Application

The user interface is the graphic user interface (GUI) and the mobile app through which users control our automated sound system. They allow users to check, control, and adjust system operations in real time — such as volume adjustment, mode changes, and inspection of how the room's acoustics are automatically equalized. Whereas the embedded hardware undertakes all the sensing, signal processing, and amplification, the GUI and app allow users easily to visualize what is occurring and make changes without the need to handle the hardware directly.

Having a good interface is a large part of why the system is usable across varying environments; be that a home theater, a lecture hall, a classroom, or an auditorium. Without one, the user would be left with preset settings and hardware adjustments, which is impractical. The mobile app and the GUI fill that gap by pairing the technical performance of the system with accessibility and usability.

7.1.1 Role of the GUI in the Project System

The GUI is the control panel of the system, displaying important data and controls in concise, uncluttered fashion. Its main responsibilities are:

- Room-Sensing EQ Visualization – Shows frequency response before and after EQ, allowing users to view how the system is treating the sound in association with the room.
- Volume and Channel Control – Allows sliders and switches that can be used for level adjustment on independent channels such as mains, subwoofers, and any other speakers.
- Status and Error Notifications – Informs users if the sensors detect any feedback, distortion, or other irregularities.

The user interface design is compatible with common usability practices like ISO/IEC 25010 [7] and usability heuristics by Nielsen [8]. Essentially, this translates into sensor data and sound data being converted into images that can easily be interpreted rather than technical values, decreasing the likelihood of user error and making the interface natural and simple even for non-technical individuals.

7.1.2 Significance of the Mobile Application

The mobile app provides the same functionality on smartphones, giving users wireless control and monitoring via Bluetooth. This aligns the project with contemporary IoT and home-smart principles, in which convenience and out-of-room access are standard expectations.

The advantages of using the mobile app are as follows:

- Remote Monitoring and Control – Conduct EQ, modes, and volume changes without access to the equipment.
- Portability – It makes the system usable anywhere in the coverage area.
- Dynamic Feedback – Graphs and visual indicators indicate changes immediately as the system recognizes and adjusts to room sound.
- Custom Profiles – Users can save personalized audio presets on a per-location basis.

The app's layout and operations are the same as the GUI, meaning users don't need to learn new controls when moving between devices. Being consistent, the system avoids causing user confusion and looks uniform across platforms.

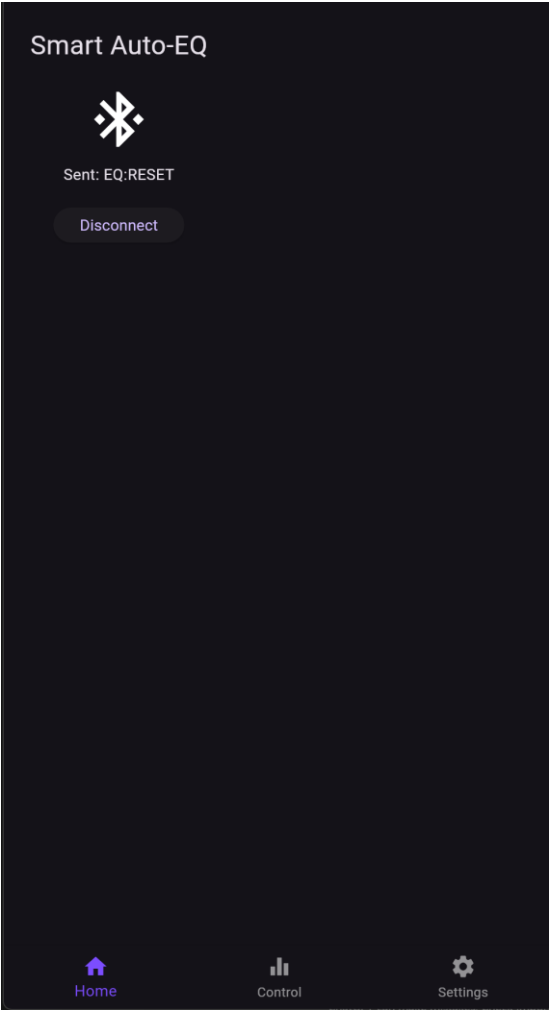


Figure 7.1 Mobile App Home Page View

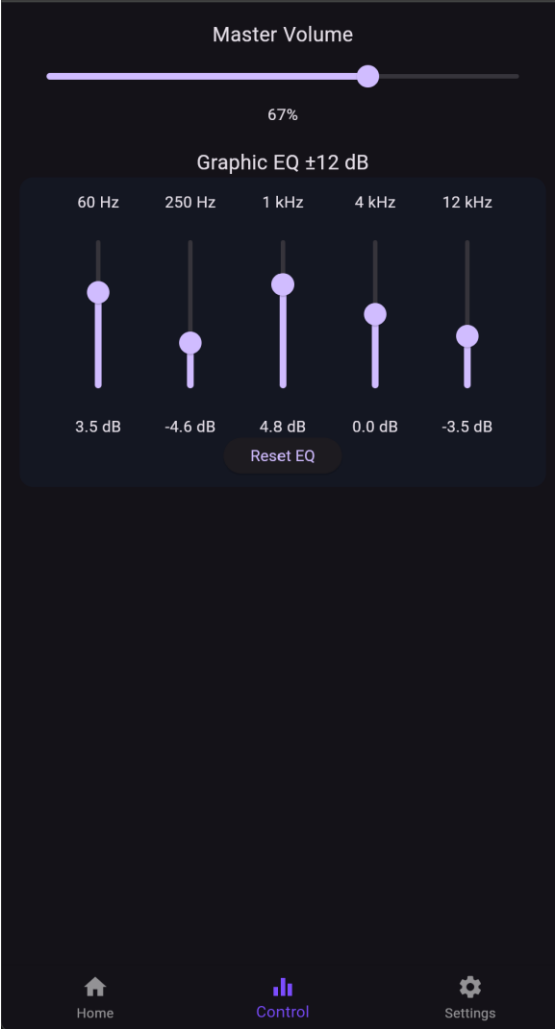


Figure 7.2 Mobile App Control Page View

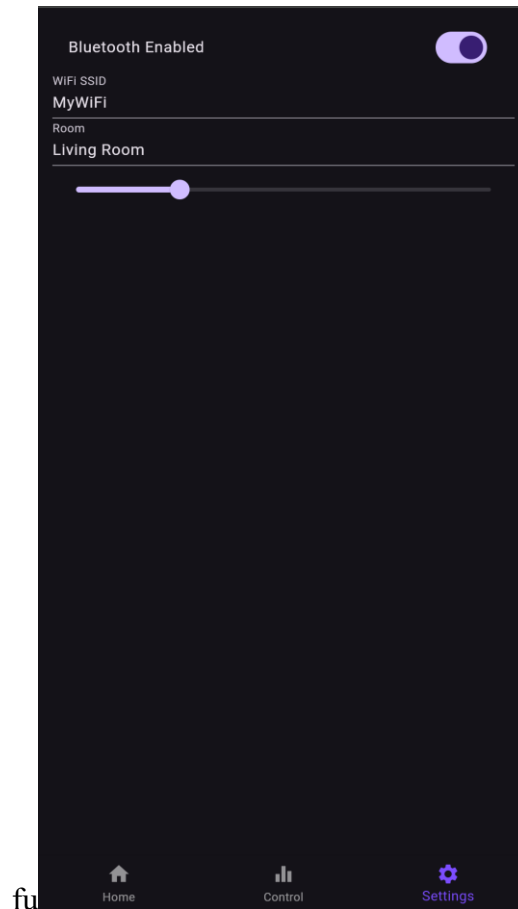


Figure 7.3 Mobile App Settings Page View

7.1.3 Integration with Embedded System

The mobile application and GUI communicate directly with the embedded hardware through Bluetooth 5.0, implemented using the BM83. This module enables fast, reliable, and secure data exchange between the ESP32-S3-WROOM-1U-N16R8 microcontroller and the user interface while maintaining very low latency, ensuring that system changes are reflected on the GUI almost instantaneously.

Data transmission follows standard, scalable communication protocols such as RESTful APIs for configuration and WebSocket-like event channels for real-time streaming, allowing efficient updates without excessive power or bandwidth consumption.

During development and testing, these same interfaces will be utilized by the team to monitor sensor data, system logs, and DSP feedback in real time through the GUI. This capability should significantly accelerate system validation, debugging, and integration, enabling us to observe the embedded system's behavior dynamically and make immediate adjustments throughout the build process.

7.2 App Architecture and Flow

The software architecture of the sound system is built around a Bluetooth 5.0 dual-mode framework implemented through the Microchip BM83 module, which interfaces with the ESP32-S3-WROOM-1U-N16R8 microcontroller using both I²S and UART communication channels.

This configuration enables simultaneous high-quality digital audio streaming and bidirectional control signaling, providing a secure, low-latency bridge between the embedded DSP hardware and the mobile application.

7.2.1 System Overview

The stack is organized into three functional layers:

- Presentation Layer (Mobile App GUI) – Displays frequency-response graphs, control sliders, and EQ settings for real-time user interaction.
- Communication Layer (Bluetooth 5.0) – Manages wireless connectivity and protocol handling. The I²S bus transmits digital stereo audio data between the ESP32 and the BM83, while the UART interface carries control commands such as connection setup, playback control, and parameter adjustment.
- Embedded System Layer (ESP32-S3-WROOM-1U-N16R8) – Performs acoustic sensing, digital-signal-processing (DSP), and equalization adjustments, returning processed data to the GUI

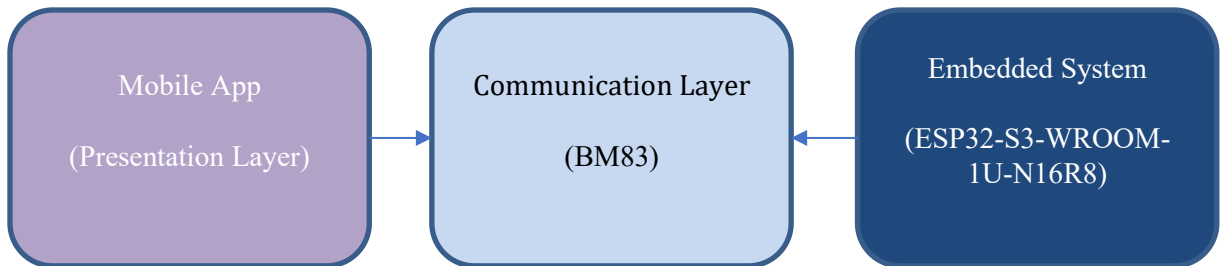


Figure 7.4 App Functional Layer Block Diagram

7.2.2 Use Case Flow

The Bluetooth interaction follows five primary stages:

1. Initialization and Pairing – The BM83 advertises device information; the mobile app connects and authenticates using standard Bluetooth pairing.
2. Audio Calibration – The ESP32 measures room response and generates corrected EQ coefficients.
3. Audio Streaming – Processed audio is transmitted to the BM83 through I²S for playback, while UART simultaneously handles control data.

4. User Adjustment – The app sends commands (e.g., volume, EQ band changes, mode selection) via UART through the BM83 to the ESP32.
5. Feedback and Visualization – The ESP32 transmits updated DSP and system data back to the GUI for real-time visualization. This procedure enables synchronized, low-latency communication and ensures that interface updates correspond accurately to hardware changes.

Simplified use case diagram is shown in Figure 7.2.2.

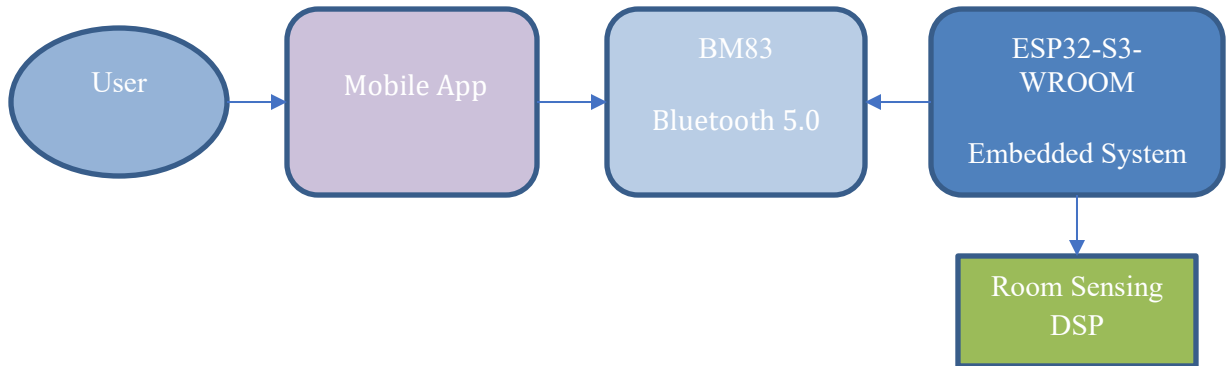


Figure 7.5 System Interaction Flow

7.2.3 Activity Flow

The Bluetooth-based system involves the following operational states:

- Advertising Mode – The BM83 goes into discoverable mode, advertising the name of the device and the Bluetooth profiles it supports, which are A2DP, AVRCP, and SPP. The mobile application discovers devices in range and pairs with them.
- Connected (Paired) – After the authentication process, the BM83 initializes and maintains the secure BR/EDR connection with the smartphone wirelessly. The ESP32-S3-WROOM-1U-N16R8 microcontroller communicates with the BM83 via UART to exchange audio parameter settings, control commands, and connection status checks.
- Audio Streaming and Data Transfer – The I²S bus constantly streams the processed audio samples from the ESP32-S3-WROOM-1U-N16R8 to the BM83 to be wirelessly transmitted. On the other hand, UART transfers real-time control and feedback messages, such as equalizer values, play commands, or diagnostic information.
- Feedback Display – The ESP32 transmits DSP output information, system information, and feedback via the UART channel in the BM83. The smartphone application refreshes the graphics displays in an asynchronous manner to represent real-time audio performance data and sensor information.
- Error Handling/Reconnection – When there is Bluetooth disconnection or signal integrity issues, the BM83 automatically goes back to pairing mode and tries to

reconnect. The ESP32 watches for the connection status pins on the BM83 and resumes streaming after connection is re-established.

The above activity chain makes sure there is constant synchronization between the mobile application, hardware components, and the Bluetooth module so that performance is not interrupted even if there are temporary wireless instabilities.



Figure 7.6 Activity Flow Block Diagram

7.2.4 Data Handling and Synchronization

Data synchronization occurs across two independent communication paths:

I²S Bus: Streams processed digital audio samples from the ESP32-S3-WROOM to the BM83. Clock signals (MCLK, BCLK, LRCLK) ensure phase alignment and prevent jitter.

UART Interface: Handles bidirectional command packets and feedback messages between the MCU and the Bluetooth module.

This separation of audio and control data prevents congestion, improves throughput, and ensures consistent timing between user actions and audio output. The ESP32 employs interrupt-driven UART routines to guarantee deterministic message handling during active streaming sessions.

7.2.5 Security and Reliability

The BM83 module provides AES-128 link-layer encryption and supports Secure Simple Pairing (SSP) for authenticated device connections. Its internal firmware implements error correction, watchdog recovery, and automatic re-linking in the event of signal interruption.

Combined with the ESP32-S3-WROOM-1U-N16R8's cryptographic and DMA (direct memory access) capabilities, the communication framework achieves stable, encrypted, and low-latency operation suitable for real-time acoustic systems.

7.2.6 Summary

The app architecture for the automated sound system is designed around a Bluetooth 5.0 dual-channel configuration using the Microchip BM83 and the ESP32-S3-WROOM-1U-N16R8 microcontroller.

Audio signals are transmitted digitally via I²S, while UART manages control and system communication, ensuring seamless coordination between data and sound streams.

The Bluetooth architecture supports event-driven operation with minimal delay, maintaining smooth synchronization between user input, DSP computation, and wireless playback. With integrated DSP capabilities, low-power design, and hardware-based encryption, the BM83 provides a robust foundation for secure, high-fidelity, real-time communication. By maintaining a layered structure, presentation, communication, and embedded processing, the design achieves modularity, scalability, and energy efficiency, ensuring reliable performance and user responsiveness under realistic operating conditions.

7.2.7 Senior Design II Implementation and Changes

During Senior Design II, the Flutter mobile application remained the primary user interface used during the final demonstration. The application was refined to provide a more direct control flow between the user and the embedded system. A dedicated calibration button was added to begin the Auto-EQ process. When activated, the system played a sine sweep through the speakers while the ESP32-S3 captured microphone feedback from the room. After the sweep was completed, the application displayed a histogram based on the measured frequency response, giving the user a visual representation of the acoustic feedback used by the Auto-EQ process.

The application interface was also simplified and improved for the final prototype. The overall layout was reduced to two main screens: the Control Page and the Settings Page. This made the application easier to navigate while still preserving the main system functions, including calibration, manual EQ control, and connection settings. Additional visual customization was implemented through light mode and rainbow mode, allowing the user to change the appearance of the interface. The manual equalizer was also expanded to eight frequency bands: 60 Hz, 125 Hz, 500 Hz, 1 kHz, 2 kHz, 4 kHz, 8 kHz, and 16 kHz. This increased range gave the user more detailed control across the low, mid, and high frequency regions while maintaining a simple slider-based interface.

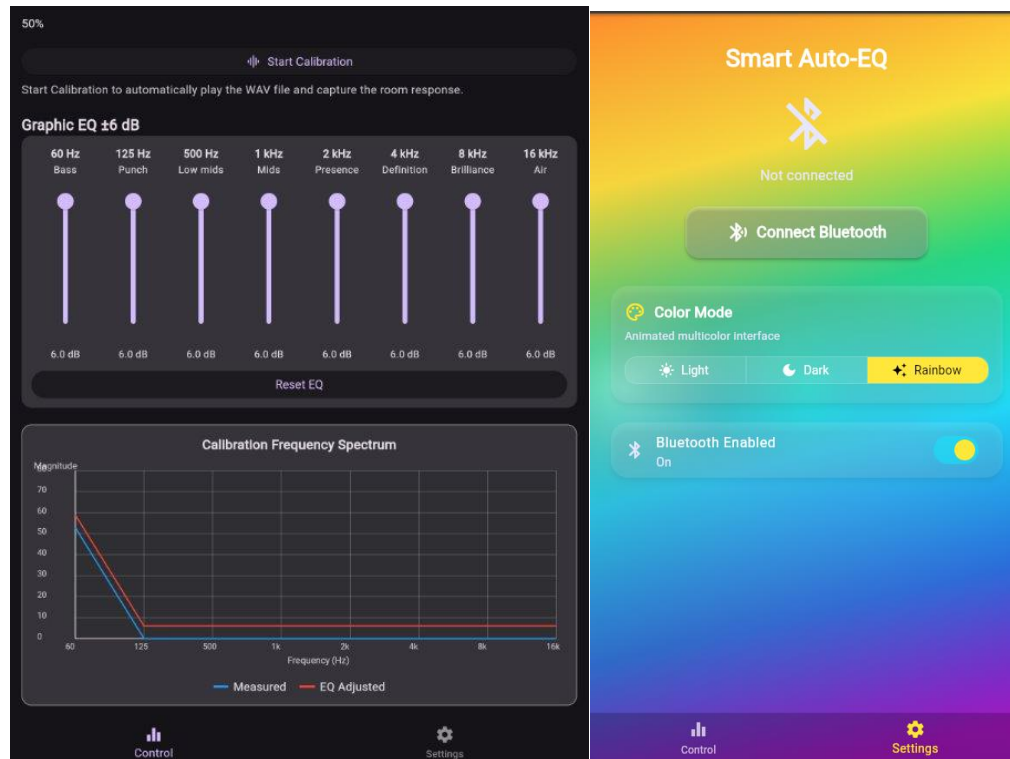


Figure 7.2.7 Senior Design II Application

7.3 Auto-EQ Algorithm

The Auto-EQ algorithm will be the backbone of the software flow, which will oversee processing the signals picked up by the microphone. This algorithm will be designed in accordance with our hardware constraints, as specific components might limit our processing power. Therefore, we will not be able to design the most powerful Auto-EQ algorithm, but it will be the most optimized for our hardware capabilities. As stated in the MCU selection phase, the ESP32-S3-WROOM-1U integrates the same microprocessor as an S3 MCU, which inherits the DSP library called ESP-DSP. This library will help us with simplifying most of the DSP work, as we will need to perform many functions that are related to this, such as FFT and windowing. Moving to the hardware components, the microphone will be needed in the early stages of the software flow, where we will use it to capture the response of the room when exposed to a frequency sweep. Once we gather the necessary information (in the form of 16-bit unsigned numbers), the MCU will start processing the data to produce an equalization setting that will be automatically set (we will also provide a manual override to this, in case the user decides to adjust the EQ after or without running the algorithm) to the sound being played on the speakers.

7.3.1 1st Stage: Signal Generation and Data Gathering

The first phase of the algorithm will consist of playing a test signal, picking up the room's response using the calibrated microphone, and creating the data that represents the response, which will be stored in memory.

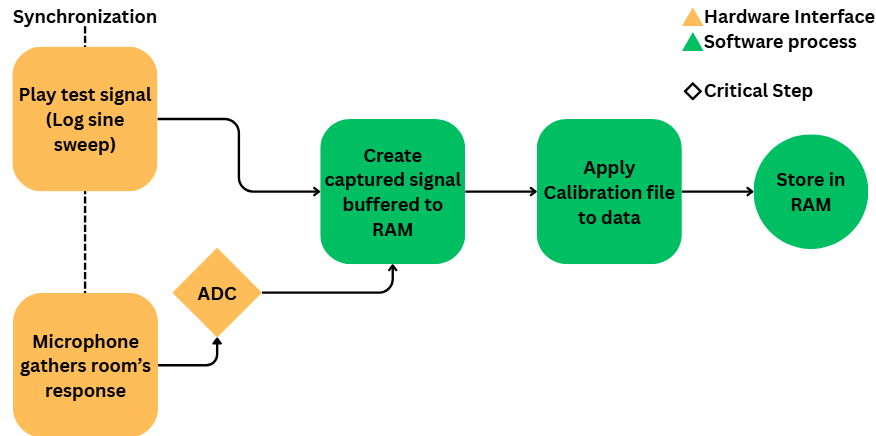


Figure 7.7 Test Signal Generation and Data gathering

The test signal in hand will be a logarithmic sine sweep, which will be played from a .WAV file stored in the MCU's flash memory. Before running the test, we must synchronize the clocks for the speakers and the microphone to avoid disparity in the measurements. In other words, the microphone will start sampling at the same time the sine sweep starts playing. It will sample the signal with a sampling frequency (f_s) of 48 kHz; to avoid aliasing, the sampling frequency must be a little over twice the maximum frequency, which in this case is 20 kHz. 48kHz was selected because it is a very common sampling rate and is easier to work with than 44.1 kHz (which is one of the most common rates). The EMM-6 transmits the data in raw format, meaning we will need to route it to one of the ADC units in the ESP32 where it will interpret the data digitally, so that we can store it in our MCU for future processing. Once we get the data, we can move to the next stage. Since we are using a calibrated microphone, we will apply calibration at the end of this stage. This calibration file (a .txt file) will also be stored in flash memory beforehand and is provided by the manufacturer; this file is unique for every microphone, so we can be certain that the correction will be specific to our microphone only. The calibration data is to compensate for the natural response of the microphone, which might struggle or exaggerate in some frequency ranges.

7.3.2 2nd Stage: Signal Processing

Once the necessary data is gathered, we will begin processing the signals (for simplicity we will refer to the signal x/X and the processed signal as y/Y hereinafter) to extract the information that will later help to design and adjust the equalizer (each band specifically). First, we will need to compute the Fast Fourier Transform (FFT) for both $x(t)$ (known) and the $y(t)$ (unknown until after **Stage 1**) to find the representation of each in the frequency

domain, $X(f)$ and $Y(f)$, respectively. But why do we want the FFT in the first place? Since we want to perform a spectrum analysis (analysis of frequencies) of our system, we need to take the FFT of both signals to identify frequencies that may affect performance [66], which will be digitally processed to provide the best sound. It's worth noting that the FFT is an algorithm based on Discrete Fourier Transform, the actual mathematical process that provides such results. We can pre-compute the FFT for $x(t)$ to save processing time and speed up the process, since we would already be in possession of the file representing this signal, and the WROOM-1U comes with plenty of storage. Once we find both FFTs, we need to find the impulse response of our system. In time domain, the Impulse response of a system is given by:

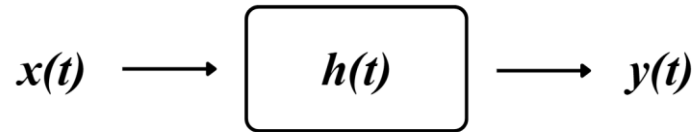


Figure 7.8 Impulse Response Representation

$$y(t) = x(t) * h(t)$$

where $h(t)$ is the impulse response of the system.

Convolution is a math-heavy approach, and we want to simplify the processing so that the ESP32-S3 runs smoothly; therefore, we convert both signals to frequency domain. After that, by applying the Convolution property for Fourier Transforms, we can write the equation above as

$$Y(f) = X(f) \cdot H(f)$$

and solving for $H(f)$

$$H(f) = \frac{Y(f)}{X(f)}$$

This is the general representation of the transfer function (or Impulse Response in this case) for any digital system. However, one important factor we do not want to avoid talking about, and minimize for our analysis, is noise. This is present in any system, in both digital and analog, and how each one takes care of it varies. Since this is not intended to be a high-end audio system, and our resources are limited, we will need to suppress noise in the software domain. We will be using a Wiener Filtering approach. Sometimes also referred to as “Wiener Deconvolution,” this is a type of filter utilizes a statistical approach to reduce the mean square error (known as LMS) between the estimated and desired signal. [65] In other words, it predicts what the output signal would be like if noise and other factors were

not present at the time of capturing the signal. The general formula for designing a Wiener Filter is given by:

$$H(f) = (X^T \cdot X + \delta)^{-1} \cdot X^T \cdot Y \rightarrow H(f) = \frac{X^T Y}{X^T X + \delta} [65]$$

where X^T is the conjugate of X (the input signal) and δ is the regularization factor. This term is usually a very small number used to “avoid biased solutions” and can vary to provide the best result possible. [66] This means that during our design of the software system we will need to tune this factor to achieve the best approach for our Wiener Filter. At this point, we are ready to go back to time domain. Since we have already computed the Transfer Function of our Impulse Response (H) with the expression above, we will need to perform the Inverse FFT (IFFT) to find H in time domain ($h(t)$).

At this point we must realize that our analysis so far has been focused on a discrete-time signal, as our test noise will only be played for a certain amount of time. But our system will be driving continuous data/music to the speakers, as it is expected that music will be played continuously until the user decides to stop. This means that we would be using a discrete spectral analysis and apply it to a continuous time signal. The Fourier Transform also introduces some error on the frequencies next to the ones being processed, usually called “side lobes.” This can be minimized using a Windowing Technique, which reduces the effect of these lobes by only multiplying the desired space (inside the window) and multiplies the rest by 0 (outside the window). [72] There are different types of windowing, but our ESP-DSP library supports a few of them; for our design, we will be using the Hann Window as it is one of the most common windowing methods and is also supported by our ESP-DSP library. Hanning windowing is also helpful to effectively handle late reverberation and echoing, two common factors in many listening spaces. It is worth noting that this method is not entirely necessary to accomplish our goals, as it only enhances and “polishes” the impulse response function for further processing. Knowing that our hardware capabilities are limited not only by our MCU, but also how much software power we can allocate to achieve this. We will need to evaluate if this positive impacts our outcome if we want to add it to our final implementation. Finally, we want to continue our analysis in frequency domain, so we must take the FFT of our window impulse response.

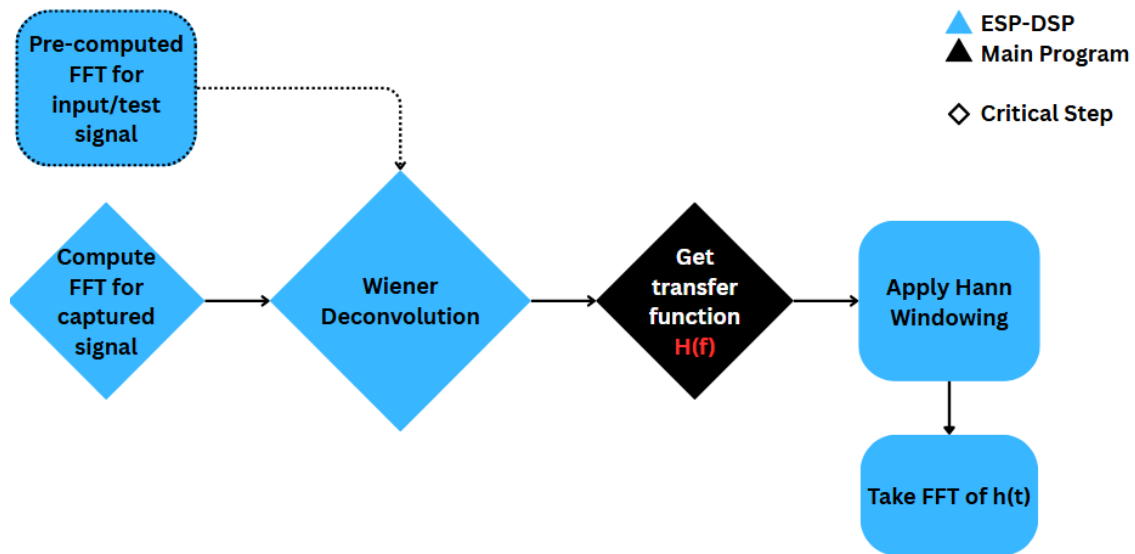


Figure 7.9 Digital Signal Processing Step

The last step in this stage would be to compute the correction curve for equalization. This step will vary depending on the choice of the user. If they decide to apply a preset (increase bass, decrease high frequencies) the curve will be specific to that, but if they decided to run automatic equalization, then the curve would be based on achieving a flat correction compared to the processed impulse response. The desired response (or equalization effect) will be called the target curve.

7.3.3 3rd Stage: Filter Design

After processing the necessary signals and gathering the necessary information, we are now ready to design our filter to implement Equalization for our system. The correction curve can be represented as

$$C(f) = \frac{\text{Target Curve}}{H(f)}$$

Due to its simplicity and effectiveness for real-time implementation as equalization requires, an IIR Filter for each frequency band will be designed. The number of bands that we will implement will depend on how the system responds in terms of latency and computational power as we begin to design the filter. Our phones, for example, usually implement 9 or 10 bands; if we consider our limitations, we think a minimum of 6 bands is very achievable, but we will perform some tests to determine if we can implement more bands, and if this has a great impact on software and hardware resources, as well as the end result of the system. The ESP-DSP library contains many functions that make this design-friendly. The majority of these functions generate bi-quad filters of all sorts: high-pass, low-pass, band-pass, etc. These filters have the following similar representation:

$$y(n) = b_0x(n) + b_1x(n - 1) + b_2x(n - 2) + a_0y(n - 1) + a_1y(n - 2)$$

where a_0 , a_1 , b_0 , b_1 , and b_2 are the filter's coefficients. These coefficients are generated in accordance with the analysis of the correction curve, as they represent the magnitude by which each filter term must be multiplied to achieve the desired outcome. In other words, the correction curve produces magnitude coefficients that will later be used to realize the necessary bi-quad filters. These filters will then be applied before sending the stream/data to the DAC for playback. Another approach we considered is building only the right number of necessary filters to achieve the correction curve. For example, if after the signal processing phase is done the algorithm realizes that only three frequency bands need correction, then three filters are sufficient for achieving this. This would save processing time and complexity, which would make our algorithm more efficient. Finally, if at any point more filters are needed, we can create as many as we need without going over the maximum number of bands.

7.3.4 4th Stage: Playback & Correction Application

There are two main sources of data in our system: Bluetooth streaming and USB connection. The first one, as the name suggests, utilizes a Bluetooth module (BM83) that transmits data via I2S to the MCU which then applies the necessary correction to the data in real time. The latter implements a USB-A connection that is hooked directly to the ESP32 via the necessary data line pins and applies the same correction. After the data is processed by the MCU, it is transmitted again via I2S to an external DAC which finally converts the digital signal to an analog signal that is to be delivered to the mid/high and subwoofer amplifiers.

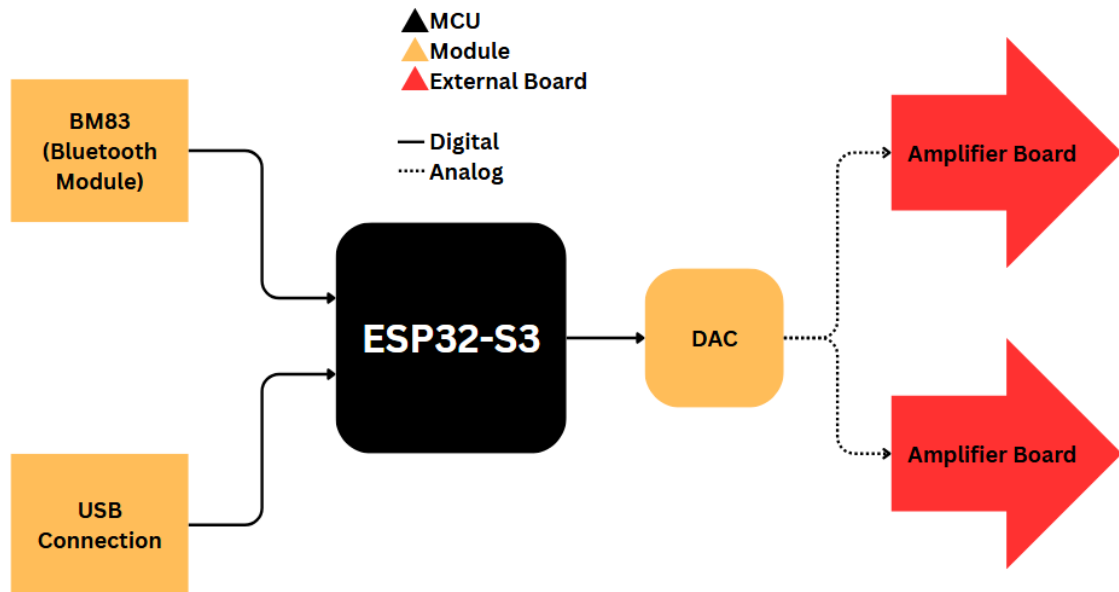


Figure 7.10 Datapath Overall Flow

Chapter 8 – System Fabrication/Prototype Construction

Once our schematic layout was completed, we needed to export our schematics into a printable PCB layout. Upon starting our PCB layout, we decided that creating two PCBs, one for analog signals, and another for digital signals would be most beneficial to avoid issues with establishing ground planes. This conclusion was made by understanding that when having analog and digital components on the same board you are essentially turning one PCB into two smaller ones by dedicating each side of the PCB to a specific signal component.

8.1 Analog Signal PCB

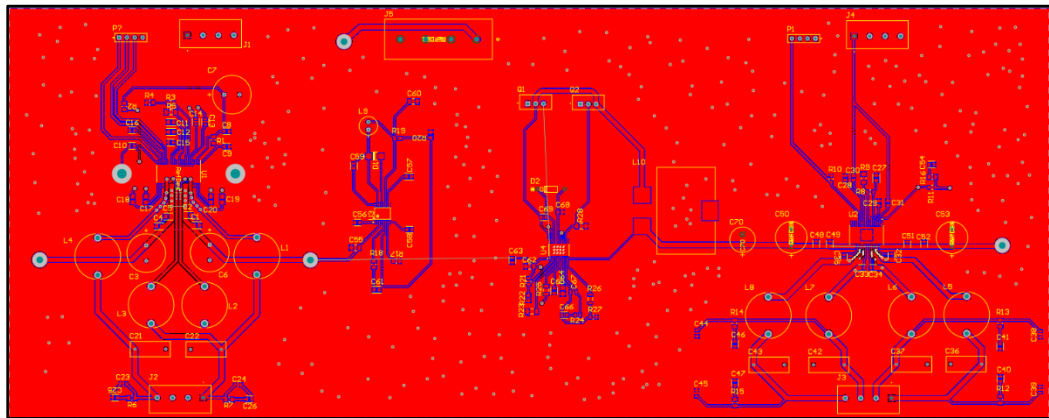


Figure 8.1 Analog PCB 2D View

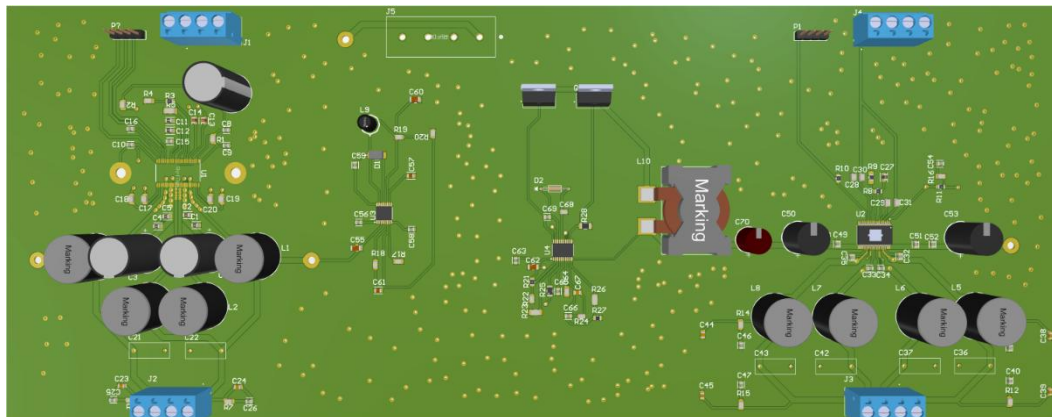


Figure 8.2 Analog PCB 3D View

Above is the printed circuit board (PCB) to implement the subwoofer amplifier, 12V regulator, 26V regulator, and the mid/highs amplifier. The 12V regulator is utilized to

supply the logical power for the TPA3251 and the 26V regulator is used to supply the TPA3116 all its power.

The board utilizes 2oz copper as recommended by the TPA3251 datasheet to allow the traces to carry the required current, while keeping the trace widths at a minimum. However, this consideration does have its tradeoff as 2oz copper boards are more expensive than the traditional 1oz copper boards.

Additionally, since the 36V pads and output pads for the subwoofer amplifier require large currents, wide traces were utilized to achieve around a 7A current carrying capacity. The same can be for the 26V pads and output pads of the TPA3116.

To ensure our components have maximum thermal stability, a heatsink was included for the TPA3251 this heatsink is mounted through two mounting holes created to fit M3 screws. The mounting holes were also created in a way that allows the heatsink to couple to the ground plane as recommended in the PCB layout examples in the datasheet. The heat sink chosen was the ATS-TI10P-519-C1-R3 as recommended by TI forums regarding the TPA3251. The TPA3116 has a bottom thermal pad that couples the ground eliminating the need for an external heatsink, however for this version of the TPA3116, TI recommends that the designer scatter vias throughout the board to minimize increase the effective heatsinking abilities of the ground plane.

This board has seven total input and output terminals. Two of the terminals are speaker wire terminals to allow the output of the DAC on the digital board to transmit the unamplified audio signal to the analog board through speaker wires; one is to carry the lower frequency band to the TPA3251 and the other is to carry the higher frequency band to the TPA3116. Another two are used to output the amplified audio signal to the system's speakers; just like the audio inputs one is for low frequency transmission to the subwoofer and the other for higher frequency transmission to the mids/highs. Two more terminals are to transmit the status and command signals to the MCU, one for the TPA3251 and another for the TPA3116. The final terminal is used for the 36V DC power input from the power supply.

For this first rendition of the audio board, holes were created on the board to allow for external wires to connect the 36V supply to the subwoofer amplifier to avoid using very long and wide traces to carry the power through the board and potentially causing noise in our audio traces. The same was done for the output of the 26V regulator. This consideration may be removed in later updates of the layout as a more efficient layout may remove the need for this sort of power distribution.

8.1.1 Senior Design 2 Implementation – Analog PCB

Although the schematics the analog PCB remained the same, we opted to separate the 24V regulator to mitigate switching noise on the board. Additionally, the amplifiers were split

into two separate boards to promote modularity and make it easier to troubleshoot the amplifiers.

The 24V regulator was operated for a majority of the semester, however, during the final week of Senior Design 2, a MOSFET blew so the Digital 3.3V regulator PCB was utilized to create a 24V, 3A regulator.

The layouts for these boards are shown in the figures below.

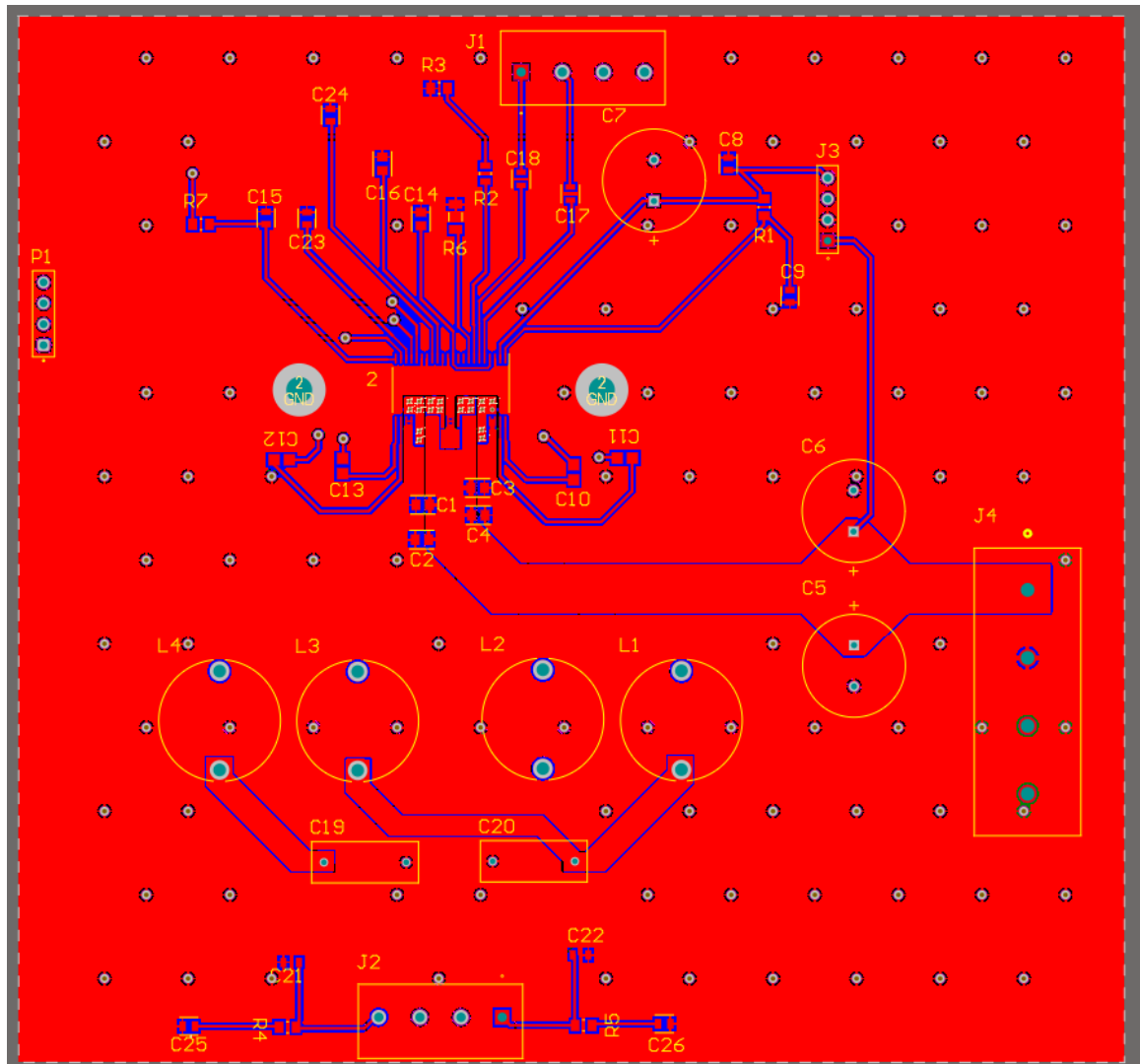


Figure 8.3 Analog PCB 3D View

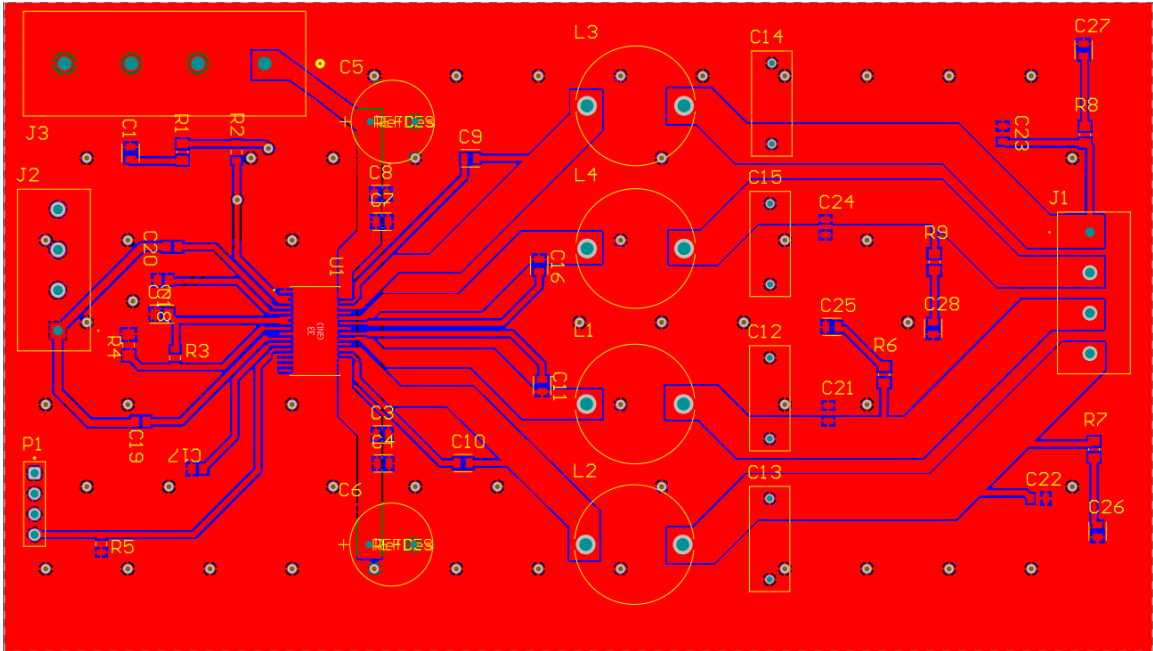


Figure 8.4 Analog PCB 3D View

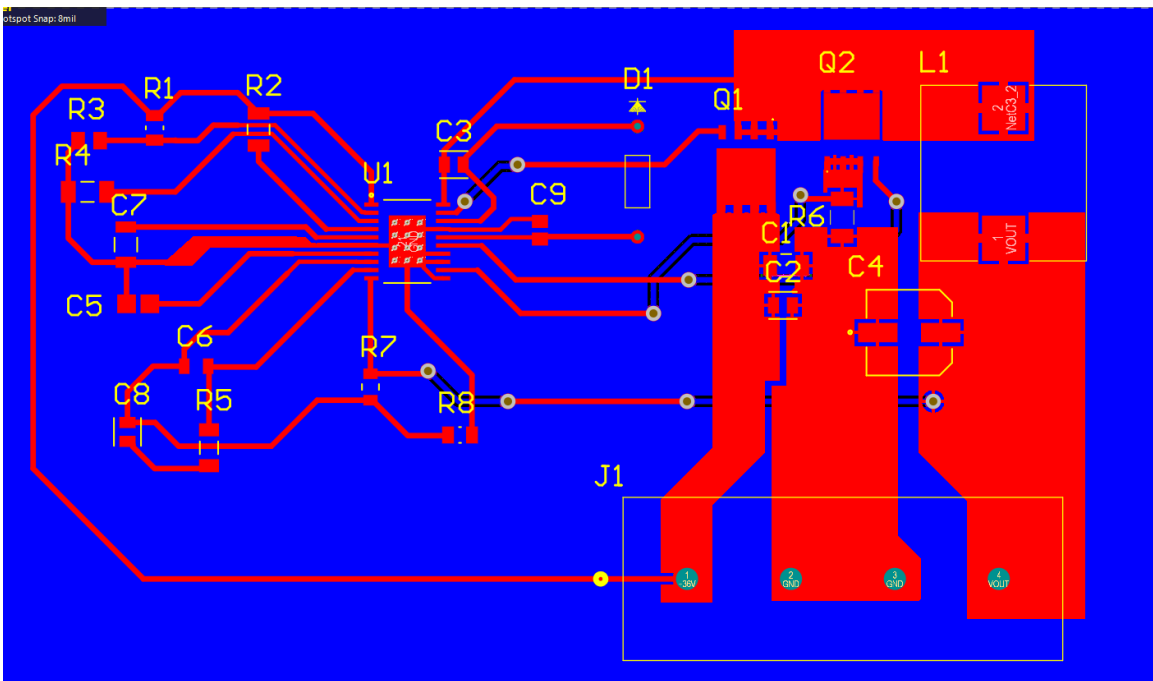


Figure 8.5 Analog PCB 3D View

8.2 Digital Signal PCB

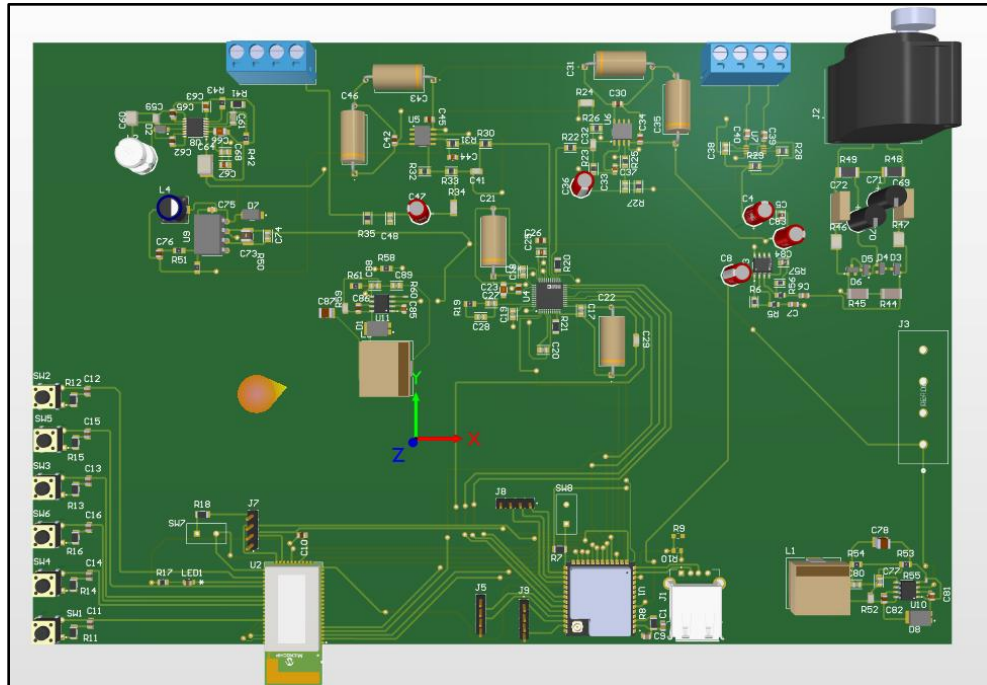


Figure 8.5 3D Representation of Digital PCB

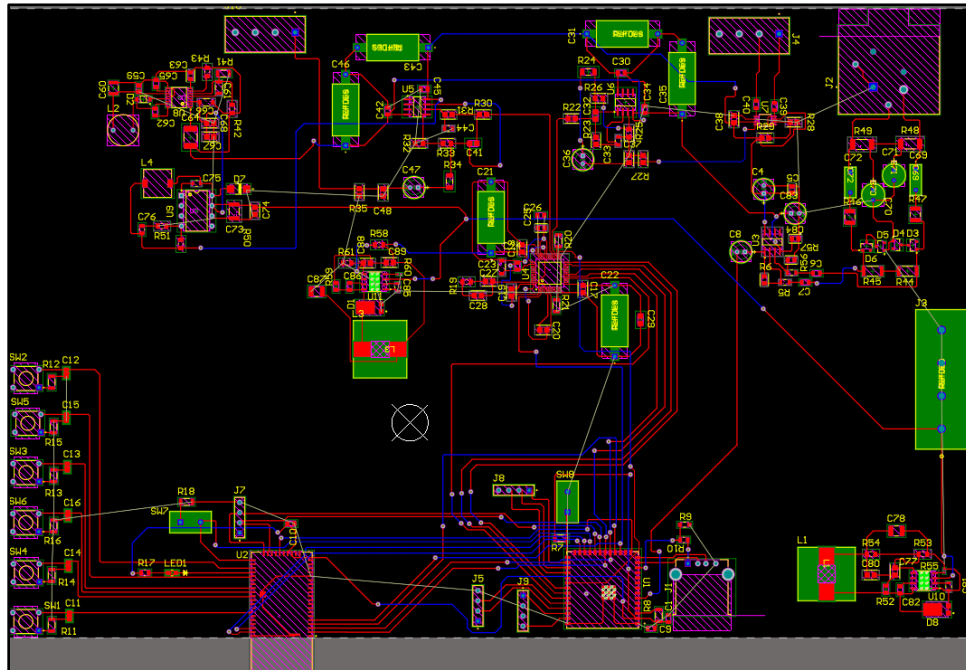


Figure 8.6 2D Representation of Digital PCB

The above PCB schematic contains the MCU, the DAC, the Bluetooth module, the -5 volt regulator, +5 volt regulator, 3.6 volt regulator (for both the DAC and MCU), the XLR cable connector, the USB connector, and switches for the volume control, play/pause, pairing, and forwarding. The -5- and +5-volt regulators are used to power the op- amps needed for the various filters on the board, while the 3.6-volt regulator is powering the logic of the MCU. The XLR cable connector is what will be used to plug in our microphone and have it send its audio feedback to the MCU. In addition, the component J3 is the connector that will be used to drive the input voltage seen by our power supply.

One of the first things that was monitored during the fabrication of this PCB was the separation of the DAC from the MCU and the Bluetooth module. With the DAC dealing with analog signals, we need to keep this away from components such as our MCU and Bluetooth module as they are digital signal dependent so by having them close together it can cause interference between the systems. As the MCU inputs a digital signal to the DAC, the DAC outputs an analog signal. By having an analog signal on this board, we will also need to separate the ground planes on this PCB, as mentioned before by having the DAC on the opposite side of the board than the MCU it will allow us to create a more clean and divisive ground plane between the two signals. By separating the two ground planes we will need to establish a star ground.

Due to the appearance of two different signals, both AC and DC, we will need to implement a PCB fabrication strategy called star grounding. This technique has all ground connections connect into a central port, to reduce electromagnetic interference, while still maintaining

a common ground. This ground is typically placed in the middle of the PCB, with each load's ground connected through a straight line. An added benefit of adding a star ground to this PCB is that it can also prevent ground loops between the two types of signals [74].

Like how the DAC needed to be separated from the MCU and Bluetooth module, all the regulators needed to be separated too due to the switching that occurs from the regulators. These switching regulators can cause crosstalk, which is the unwanted transfer of energy between traces due to capacitance and inductance.

Another aspect of the schematic that is worth noting is that the MCU and Bluetooth module are on the edge of the board and facing the same direction, this is to assist us with signal integrity and RF performance. By placing the boards in the same direction and on the edge of the board we have the antenna keep out zones aligned, matching the correct polarization of the antennas, along with a clear space to allow for optimal connection between the two devices.

Between all components there are a total of 9 input/output terminals on the board, these are notated in the following way.

Table 8.1 Input and Output Connections

Component Tag	Component Name	Input, Output, Input/Output
J1	87520-0010BLF	Input/Output
J2	NC3FBH2-B	Input
J3	4DB-P108-04	Input
J4	TB001-500-04BE	Output
J5	61300411121	Input
J6	TB001-500-04BE	Output
J7	61300411121	Input
J8	61300411121	Input/Output
J9	61300411121	Input/Output

Pins J5 and J7, although the same type of component as J8 and J9 are input because these are the ones that we will be using to connect our MCU to program the board.

As opposed to our amplifier board we will be using 1 oz copper because we don't need the extra current carrying capabilities that is needed on the amplifier board. The BM83 and ESP also use 1oz copper as a standard as 1oz copper is a lot more efficient relating to speed than the 2oz counterpart [75].

8.2.1 Senior Design 2 Implementation – Digital PCB

Like what we did with the analog board, we separated all regulators including the digital +3V3, analog +3V3, +12V, and +24V. In addition, changes were made to the layout to reduce noise on our phantom power circuit and rerouting the new components that were needed through various iterations. Some of these choices included using different pins on the ESP along with bringing ICs closer to meet I2S trace rules. Another aspect worth mentioning is that the rule issue that was had with Altium also affected one of the iterations of our main board, which shorted all through-hole components to ground, however, this issue was fixed.

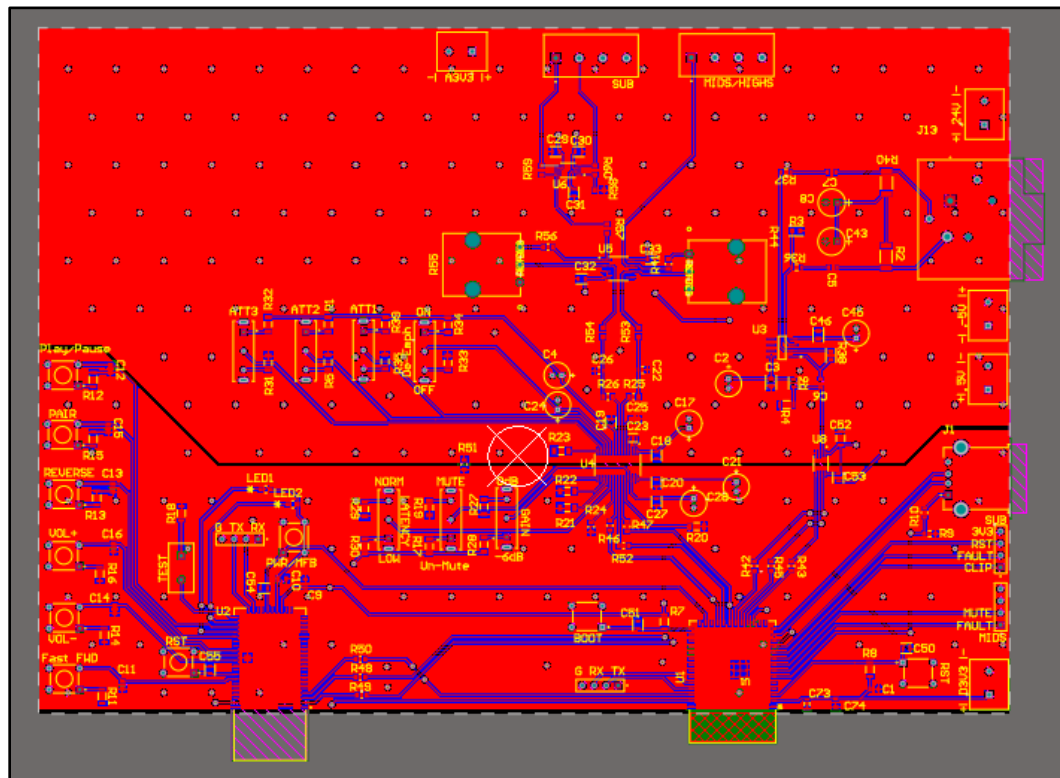


Figure 8.7 2D Updated Representation of Digital PCB

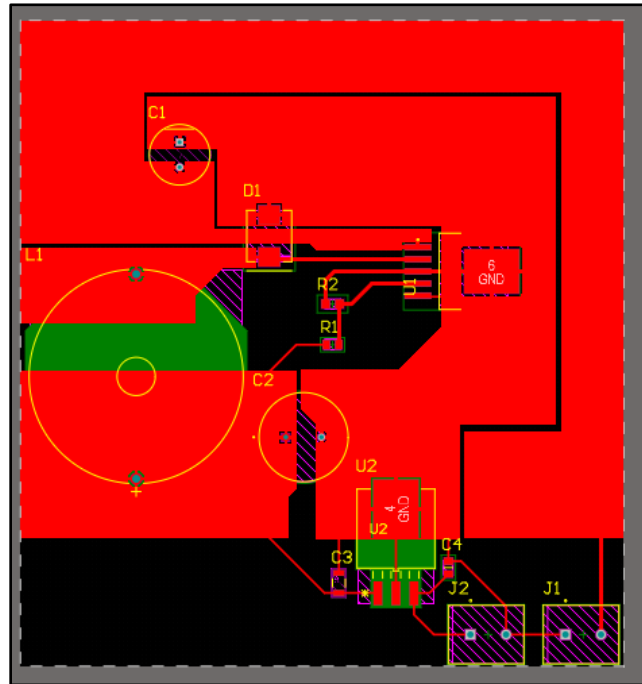


Figure 8.8 2D Updated Representation of +A3V3 PCB

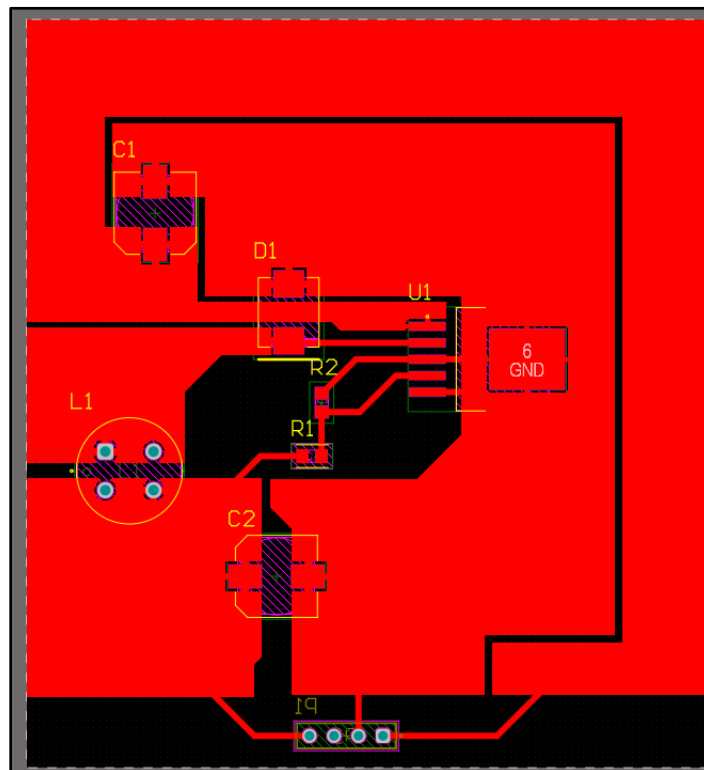


Figure 8.9 2D Updated Representation of +12V PCB

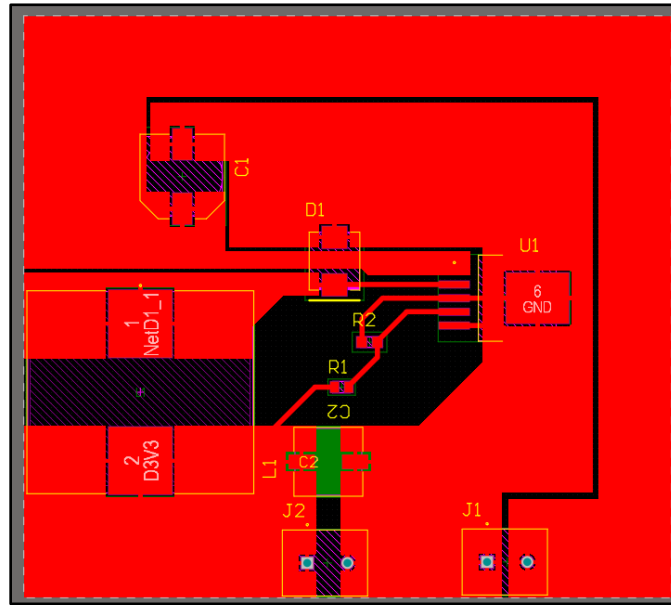


Figure 8.10 2D Updated Representation of +D3V3 PCB

8.3 Speaker Cabinet Fabrication

8.3.1 Mids/Highs Enclosure

The mids/highs enclosure was not created because mid/high speakers are not as susceptible to enclosure design as a subwoofer is. The mids/highs speakers can perform under many different conditions, so a sealed enclosure with enough volume to fit the speakers will suffice.

8.3.2 Subwoofer Enclosure

With the help of online software packages like subbox.pro and diyaudioandvideo.com, the design of the subwoofer enclosure was made easy. The diyaudioandvideo website enabled us to input the TS parameters and the software spot out the recommended box specifications, such as the internal volume, 1.12 cubic feet, and tuning frequency, 43.48 Hz. This data was then input into the subbox.pro website, along with the speaker, and the software gave us a design to implement with our subwoofer. The dimensions of the speaker cabinet are 23.62"x11.12"x13.78". The port area is 23.25 cubic inches, and the total port length is 23.7". The port will be on implemented on the front of the subwoofer enclosure and wrap around to the back. A 3D model of the enclosure design given by subbox.pro is shown in the figure below.



Figure 8.11 Subwoofer Enclosure Design

8.3.3 Senior Design 2 Implementation

The enclosure design for the subwoofer seen in Figure 8.5 was not used for the final project as the material requirements favored a smaller, cheaper sealed enclosure. Additionally, the mids/highs enclosure was fabricated with leftover material as the two drivers chosen did not have any enclosure requirements, as the midrange driver controlled its own backwave and tweeters operate at a high enough frequency where backwave mitigation is not necessary the figure below show the final enclosures.



Figure 8.12 Full Enclosure Design

Chapter 9 – System Testing and Evaluation

For our system testing, we ensured we were able to operate our main components such as the microphone, ESP32 Bluetooth, BM83, and amplifiers. The microphone and ESP32's Bluetooth were tested independently of the other components, while the amplifiers BM83 was tested with the ESP32 and one amplifier at a time.

The microphone was tested independently of the other components because we only needed to see the ESP32 read the signal and convert it using the ADC. While the ESP32's Bluetooth was tested independently because the module only needs to send and receive commands which will then be processed to adjust.

The other components were tested with one another to ensure that the audio signal was able to pass through the ESP32, get sent to a DAC, and be amplified by the amplifiers. This gives us confirmation that the system flow can be successfully implemented.

9.1 Microphone Testing

9.1.1 Test Equipment and Components

To confirm that our microphone works and can be implemented successfully, the following equipment and components were used.

Equipment:

- Rohde & Schwarz RTM3004 Oscilloscope
- 2-Rohde & Schwarz NGE100 Power Supply
- Rohde & Schwarz HMC8012 Digital Multimeter
- Device with 1kHz test tone
- Breadboard

Components:

Table 9.1 List of Components

Component	Quantity
Dayton Audio EMM-6 Microphone	1
XLR Cable	1
XLR Female Connector	1
ESP32-S3-WROOM-1 Breakout Board	1
1.2k Ω Resistor	4
1.5k Ω Resistor	2
47k Ω Resistor	2
1 μ F Capacitor	3
TL084	1

9.1.2 Electrical Setup

For the setup of this test the XLR cable was first fitted on the edge of the breadboard. Pin 1 was connected to ground and pins 2 and 3 were brought out to another breadboard node to fit the phantom power circuitry.

The phantom power circuitry consisted of two 1.2k Ω resistors, two 1 μ F Capacitors, and a 26V supply. The 26V supply was hooked up to the two resistors, one resistor was sent to pin 2 of the XLR connector and the other was sent to pin 3. The capacitors were also hooked up to pins 2 and 3 of the XLR connector with the other side of the capacitor set to their own respective nodes. The basic phantom circuit topology is shown below.

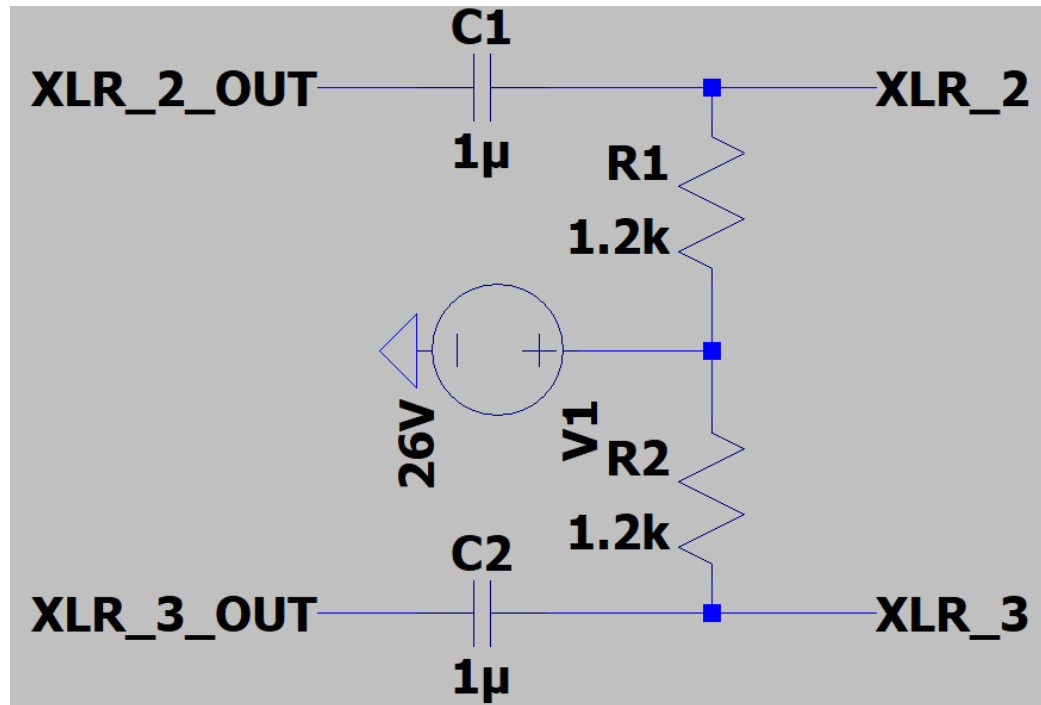


Figure 9.1 Basic Phantom Power Circuitry

Next, the XLR_2_OUT and XLR_3_OUT nodes were set to the input of our difference amplifier. The difference amplifier was set up exactly how it is in the schematic documentation in figure 6.8. To create the $2.7k\Omega$ resistors, a $1.2k\Omega$ resistor was placed in series with a $1.5k\Omega$ resistor. The rail voltages for the TL084 op amp were set to $\pm 5V$ just as they were in the schematic. The output of the difference amplifier was decoupled from DC with another $1\mu F$ capacitor. A $1.5V$ DC bias was added to the output of this capacitor through a pull-up resistor of $10k\Omega$ to set the node voltage to a point where the ESP32 can read all values across the sine wave, not just the positive side. The output was then fed to pin 6 of the ESP32 breakout board.

Channel one of the oscilloscope was set to read the output of the difference amplifier that is sent to the ESP32 and channel two was set to read the XLR_2_OUT node seen in figure 9.1. The full test setup is shown in the figure below.

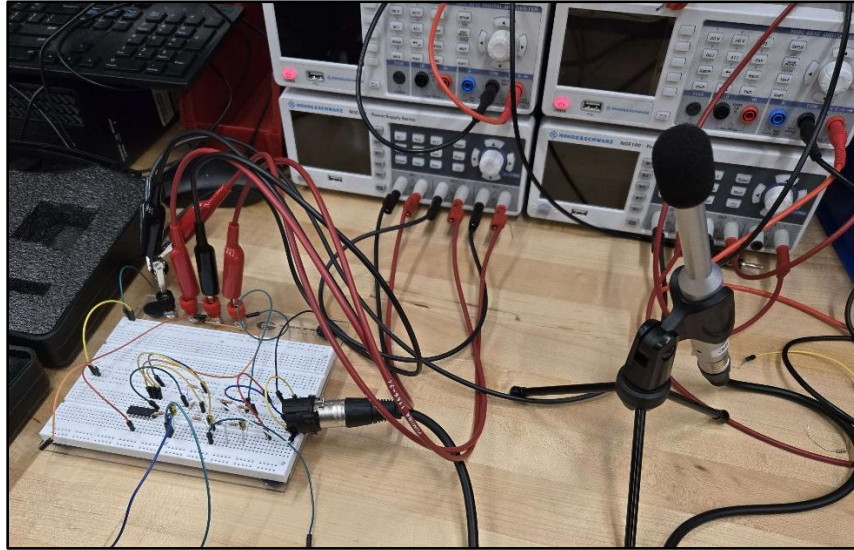


Figure 9.2 Full Microphone Test Setup

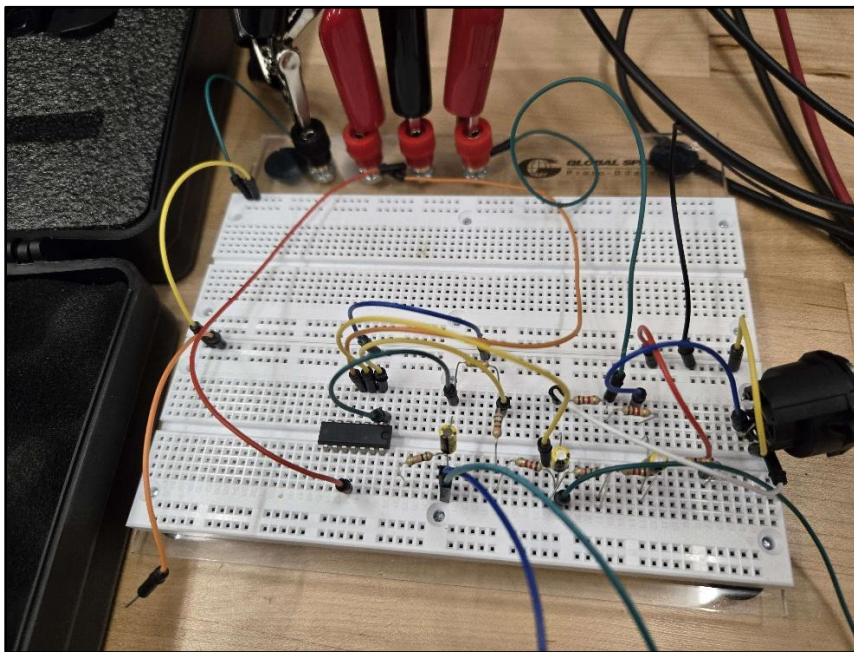


Figure 9.3 Microphone Circuitry Close Up

9.1.3 ESP32 Setup

To realize our implementation, we needed to create a program capable of reading data from the microphone while using the ADC to convert it to digital domain and print the data to a serial monitor (UART COM channel). The ESP32 comes with a library full of built-in functions and helpers that eased this process. The ADC was set up in continuous mode,

which basically reads data until the internal buffer is filled up, or the program stops the function. After data was processed by the ADC, enough space was allocated in memory to store this data using DMA (Direct Memory Allocation). In our main while loop, we implemented an ISR that tells the program that the conversion from the ADC has finished and is waiting in its internal buffer. Then, we start reading data and store it in the allocated memory space; once the 'adc_continuous_read' function times out (the functions from the Espresiff libraries throw a special type of error defined as 'esp_err_t' that makes this easy to handle), we start printing the data in the serial monitor. Each conversion frame contains multiple conversion results, each 4 bytes long (ESP32-S3 specific). Within these 4 bytes (32 bits), the actual sampled number is a 12-bit value. Once we extract this value, which just represents the raw data, we need to convert it using the equation provided in the ESP32-S3's datasheet:

$$V_{data} = \frac{V_{ref}}{4500} * (raw\ data)$$

where $V_{ref} = 1100\text{ mV}$ by design. Once the conversion was computed, the data was printed to the serial monitor for verification.

9.1.4 Tests

The tests for the microphone were split into three segments, ensuring the ESP32 was converting an analog signal to digital values through its ADC, then the microphone and supporting circuitry was tested, and finally the two were put together to ensure the entire process was working in conjunction with each other.

To test the ESP32's ADC function a known DC value was input to the breakout board through a $10\text{k}\Omega$ resistor. The ADC reference was set to 3V with 2^{12} bits, 4096 total values, and a 44.1kHz sampling frequency. The program was run multiple times with different DC values to ensure the ADC readings were accurate.

Next, the microphone was tested at the XLR output and the amplifier output, both tests were shown to be successful as the microphone was picking up our voices and taps on the microphone transducer. To further confirm the microphone was working correctly, a 1kHz sine wave was played off of YouTube. The oscilloscope was monitored to ensure that the output of the microphone was a 1kHz sine wave with minimal hiccups, as hiccups are to be expected from ambient sounds traveling to the microphone. This monitoring showed the microphone was indeed outputting a 1kHz sine wave and the difference amplifier was also outputting the same signal with a greater amplitude.

During the full test, a few changes had to be made, from the original design to make the microphone work with no hiccups. The changes made include adding the 1.5V bias to the op-amps output and fine tuning the difference amplifier with the resistor divider on the non-inverting input to make the gain of both the input terminals the same to mitigate the noise picked up in the transmission.

After the changes were made, the microphone was successfully outputting a signal from to the difference amplifier and the amplifier was able to output a clean audio representation of the sound picked up by the microphone at a level the ESP32 could process. The ESP32 was also able to read the data and convert it to a digital signal.

9.1.5 Conclusion

After multiple successful tests of the microphone a demonstration video was created for the system demo video required at the conclusion of the semester. The overall test yielded successful results and granted insights into further design considerations for when the microphone is implemented with a PCB.

9.2 Embedded System Prototype

9.2.1 Flutter Application to ESP32 Integration

The ESP32-WROOM module was powered via USB and programmed with the firmware in main.cpp. No external sensors, pre-amplifiers, LiDAR modules, or audio hardware were included in this stage. The purpose of this fabrication step was strictly to verify:

- Bluetooth advertising
- Bluetooth connection stability
- Successfully write operations from Flutter to ESP32

The Flutter app (files: main.dart, control_page.dart, eq_sliders.dart, etc.) were compiled and run on Chrome browser. The prototype app performs the following:

- Discovers and connects to the ESP32 BLE service
- Displays EQ sliders linked to frequency bands
- Sends slider values immediately upon movement
- Verifies communication using terminal output on the ESP32

This validates the command-path, which is the heart of the system's user interaction.

9.2.2 ESP Firmware Functionality

The ESP32 firmware is designed to establish a stable Bluetooth Low Energy (BLE) communication channel and receive EQ slider commands from the Flutter mobile application. This firmware validates the communication subsystem that all later stages depend on.

9.2.2.1 Goal of the Firmware

The primary objective of the code is to configure the ESP32-S3 as a BLE GATT server that exposes a custom service with two characteristics:

1. RX Characteristic (0xFF01) – the mobile app writes slider values here

2. TX Characteristic (0xFF02) – reserved for future notifications from the ESP32

This allows the mobile application to send EQ data, which the ESP32 prints to the terminal for verification. By confirming smooth BLE communication, the team establishes the foundation needed for real-time equalization control in future development stages.

9.2.2.2 BLE Initialization and Configuration

When the ESP32 boots, “app_main()” initializes all Bluetooth components:

- NVS Flash (required for BT stack)
- BLE-only controller mode (classic BT memory is released)
- Bluedroid host stack
- GAP (advertising) callback
- GATTS (GATT server) callback
- GATT attribute table registration

The device advertises under the name "esp32s3", which is the name the Flutter app discovers during scanning.

9.2.2.3 Custom GATT Service definition

A custom 16-bit service 0x00FF is created with six handles

Table 9.2 GATT Service Definitions

Index	Attribute	Purpose
IDX_SVC	Primary Service	Declares the BLE service
IDX_CHAR_RX	RX Declaration	Defines properties (Write / WriteNR)
IDX_CHAR_VAL_RX	RX Value	Where the Flutter app writes EQ data
IDX_CHAR_TX	TX Declaration	Define Notify Property
IDX_CHAR_VAL_TX	TX Value	Placeholder for outgoing ESP32 notifications
IDX_CHAR_TX_CFG	CCCD	Enables/disables notifications

9.2.2.4 Advertising and Connection Handling

The GAP event handler manages:

- Setting advertising data
- Starting/stopping advertising
- Logging connections
- Updating connection parameters

When the Flutter app connects, the ESP32 logs: Client connected, conn_id=0. When it disconnects, the ESP32 logs: Client disconnected, restarting advertising. This ensured a smooth demo.

9.2.2.5 Receiving and Printing Slider Commands

The functionality demonstrated occurs in the GATTS write event. When the user moves a slider in the app:

- Flutter sends the new slider value over BLE
- The ESP32 receives it through the RX characteristic
- The firmware logs it to the terminal

During the demo, slider movements print the commands sent to the app terminal, such as *Send command: EQ2: 7*, this confirms the communication is functioning as intended.

9.2.3 Fabrication Constraints

1. Emulator Bluetooth Limitations

- Some Android emulators do not support BLE fully, requiring use of a physical device or Chrome browser for reliable testing

2. Hot-Reload Sync Issues

- When rebuilding Bluetooth-dependent code, the connection occasionally requires app restart or ESP32 reset.

3. Timing Constraints

- BLE write commands must be completed within the target system constraint (<100 ms), aligning with the design standards in Chapter 4.

9.2.4 Flutter App / ESP32 BLE Summary and Prototype Conclusion

The firmware and Flutter app successfully demonstrates the core communication infrastructure required for the final system. ESP32 reliably advertises BLE, connected to the Flutter application, and receives EQ slider data in real time. Proving that the foundational Bluetooth pathway, from user input → BLE transport → microcontroller reception, is fully operational.

This accomplishment clears the path for SD2 where: the received commands will drive an actual DSP equalizer, hardware and amplifier stages will be integrated, and TX notifications will carry system status back to the mobile app.

The working firmware from the prototype ensures that communication will not be a bottleneck as the system moves into full audio processing and hardware integration.

9.3 Audio Testing

9.3.1 Test Equipment

Equipment:

- Rohde & Schwarz RTM3004 Oscilloscope
- Rohde & Schwarz NGE100 Power Supply
- Device with Bluetooth Capabilities and 1kHz test tone
- Breadboard

Components:

Table 9.3 List of Components

Component	Quantity
BM83 Breakout Board	1
ESP32-S3-WROOM-1 Breakout Board	1
UD1334 DAC Breakout Board	1
SOTAMIA TPA3251 Power Amplifier	1
USB-to-UART Flash Tool	1

9.3.2 Electrical Setup

For the setup of electrical components, the BM83 was connected and powered through the USB-to-UART tool plugged into a laptop via the 3.3V line to the battery in pin. The battery out, ground, RFS1 (Word Select Clock), SCLK1 (Bit Clock), and DT1 (Data Transmit) pins were also used. The battery out pin was used to power the DAC, and the ground pin was used to reference the BM83 ground to the other components' ground. RFS1, SCLK1, and DT1 were used for data communication to the ESP32 on pins 7, 12, and 10, respectively.

The ESP32 was connected to the DAC as well through the 7, 12, and 6 pins to the WSEL (Word Select Clock), BCLK (Bit Clock), and DIN (Data Receive) pins, respectively. The ESP32 was also connected to a computer through USB-C to monitor the status of the program.

Outside of the connections already mentioned the DAC's AGND, SF0, and SF1 pins were connected to ground; the SF0 and SF1 pins were connected to ground to ensure the pins remained low so the mode selection, I2S Default, does not change. Additionally, the LOUT and ROUT pins were connected to the right and left channels of the amplifier inputs.

The amplifier was set to receive power from the power supply set to 15V and with the current limit set at its maximum value. The full electrical setup can be seen in the figure below.

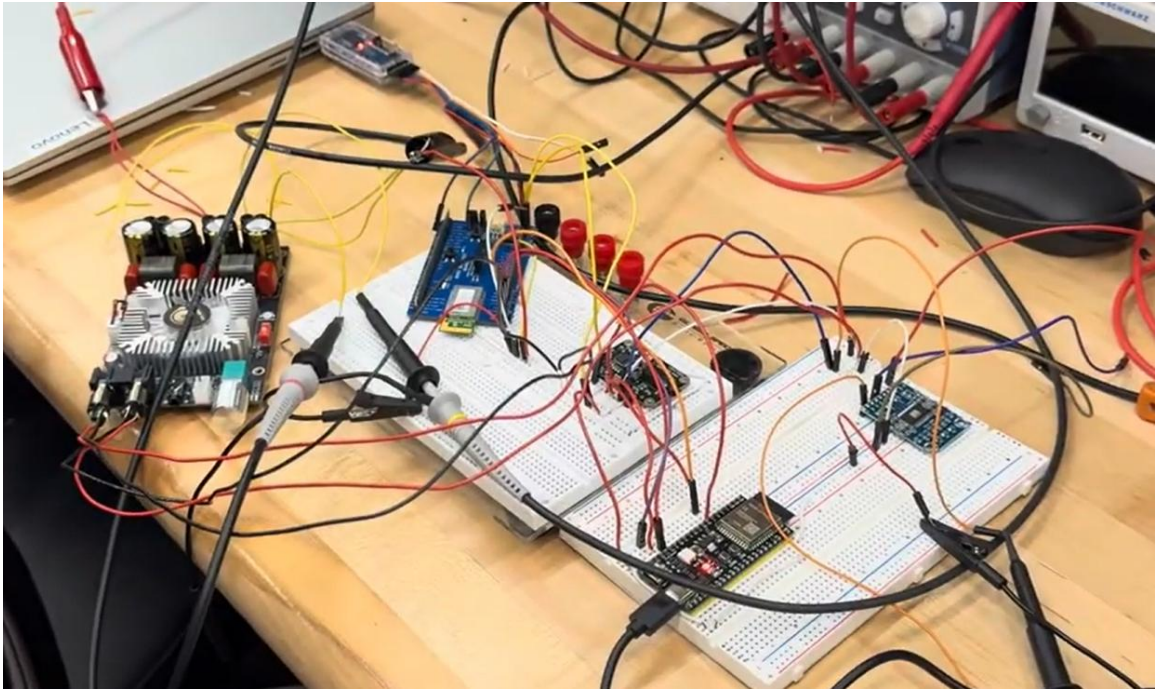


Figure 9.4 Audio Test Setup

9.3.3 Software Setup

9.3.3.1 ESP32 Programming

For this demo, the ESP32 acted in the role of the master. In I2S protocol, a “master” is needed to provide reference clock signals such as BCLK (Bit Clock) or MCLK (Master Clock). The BM83 and the external DAC behaved as “slaves”; the first one was used as a sender, while the latter was used as a receiver. In between them two was the ESP32, which used only one I2S unit (ESP32-S3 has two) in full-duplex mode. The structure for full duplex consists of configuring the main clock source, the slot configuration, and the GPIO configuration. Since the DAC in the breakout board defaults to 24-bit configuration, we had to set up the clock source in a different manner, because 24 is not a power of 2. In this configuration, we had to activate a clock divider such that it would implement a clock source that was divisible by 3; for this demo, we used the `I2S_MCLK_MULTIPLE_384` which then gets multiplied by the sampling rate. The BM83 uses Philips standard for its

I2S unit, so on the ESP32's side we also had to set it up like this. Lastly, we set the pins as mentioned before in the 'gpio_config' structure and enabled both the receiving (rx) and transmitting (tx) channels, as we are in full-duplex mode.

In our main program, we initialize the full-duplex communication by calling the function 'i2s_bt_to_mcu_to_dac_init()'. Then, we allocate space in memory that will be used to store the data transmitted by the BM83 and initialize all the variables needed to implement the logic within our loops. The 'i2s_channel_read' function appears at the beginning of the outer while loop, which reads the data from the BM83 and stores it in the buffer that we allocated space for. After that, we enter the inner while loop in which we try to transmit the data to the DAC: in case the transmission fails, we print a message stating the error number the function throws, else we continue writing to the DAC. Once it finishes, we exit the inner loop and do another read iteration, and the process starts again.

Some lines are commented out, as they served as a debugging tool and also to print information that would ensure our implementation was working, but for the demo they were commented out to avoid introducing latency into the system.

9.3.3.2 BM83 Programming

The BM83 was able to be programmed through software programs provided by Microchip. The programs used were the isUpdate tool and the Config tool. The Config tool was the main tool used to create the image file used to implement all functions for the test. The only things that were changed from the default settings were the I2S role to Slave mode, and the word length to 24 bits.

The image was then saved as a .hex file and the BM83's UART_TXD and UART_RXD pins were connected to the USB-to-UART flash tool. The Test jumper was set to be able to flash the new firmware onto the device. Once all that was completed the BM83 was power cycled and the MFB button was pressed. The isUpdate tool was then used to connect to the device and load the new firmware.

After the update was complete the Test jumper and UART pins were removed from the device and the BM83 was power cycled to enter normal operation. The device was then ready for use.

9.3.4 Issues Faced

As with any demo demonstration there were a couple of obstacles that were encountered in our testing. The first obstacle that we encountered was the fact that even in the slave mode configuration the ESP continued to try and override the BM83's clock signal. This essentially created garbage output when we were trying to measure the output of the DAC. Multiple I2S orientations were attempted to have the BM83 remain the master, however, the issue was only corrected after changing the ESP to be the master and having its clock sent to the BM83 and the DAC. The other issue we encountered in our demoing was that our amplifier's wasn't amplifying the signal, but instead almost replicated the exact output.

The output voltage of the amplifier was slightly lower than the DAC's output and can be explained by the amount of resistance within the amplifier. This issue will be examined before, and during Senior Design II.

9.3.5 Test Results

The output signal matched the frequency, and waveform of the input signal, however, at not the correct voltage magnitude. We were able to replicate a 1kHz sine wave, using the Bluetooth, MCU, and DAC. The area of this demo that needs the most attention moving forward will be integrating both the subwoofer, and mids/highs amplifier before the completion of this project.

9.3.6 Conclusion

Overall, being able to match the waveform and frequency of an input signal over Bluetooth marks a resounding step in the completion of this project. Without this piece of the project, we are limiting the connection capabilities for the user, making it crucial for the overall success of the project. As mentioned in earlier sections there is still work to be done, but substantial progress has been made in realizing this project.

9.4 Senior Design 2 System Verification

For our project we focused on three key deliverables that needed to be met. These included: meeting our THD, output power, and frequency response. More information on these can be found in the engineering specifications table.

Relating to frequency response we test this by playing a logarithmic sine sweep that was set for 20 Hz – 20 kHz. We observed to see that the speakers we capable of producing this output by hearing if the speakers clipped along with probing the output of the two amplifiers that were designed for each enclosure.

Regarding the THD we played a 1kHz sine wave and measured the output of the amplifiers and took the FFT, we then found the DBV level of the harmonics, and converted that to an associated voltage level, using the formulas below

$$V_{ref} = V \cdot 10^{-\frac{L}{20}}$$

$$THD = \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 + \dots}}{V_1}$$

To measure the output power seen by our speakers we measured the voltage in RMS of each individual amplifier channel and then calculated the output power based on this RMS value and the resistance of the speakers, which were roughly 4 ohms.

$$P = \frac{V^2}{R}$$

Chapter 10 – Administrative Content

10.1 Budget and Financing

Table 10.1 Bill of Materials

Component	Name	Price	Quantity
Mechanical			
Tweeters	BC255C08-04	\$23.95	2
Midrange Drivers	Goldwood GM-85/4	\$15.98	2
Subwoofer	Goldwood GW-10PC-4	\$79.98	1
Electrical			
MCU/DSP	ESP32-S3-WROOM-1U-N16R8	\$6.00 – 6.56	1
Mids/Highs Amplifier	TPA3116D2DADR	\$3.10	1
Subwoofer Amplifier	TPA3251D2DDVR	\$7.81	1
Bluetooth Module	BM83SM1-00AA	\$13.18	1
DAC	AD1934WBSTZ-RL	\$11.42	1
Microphone	Dayton Audio EMM-6	\$59.98	1
Power Supply	LRS-600N2-36	\$64	1
3.3V Regulator	LM5013DDAR	\$3.88	2
+5V Regulator	LM2674N-ADJ/NOPB	\$5.05	1
-5V Regulator	LM5574MT/NOPB	\$4.45	1
12V Regulator	LM25574MTX/NOPB	\$3.25	1

26V Regulator	LM5116MH/NOPB	\$6.67	1
PCB			
USB-A Connector	87520-0010BLF	\$0.91	1
XLR Connector	IO-XLR3-F-EH	\$2.63	1

10.2 Project Milestones

10.2.1 Senior Design I

Table 10.2 Senior Design 1 Milestones

Weeks	Completed Activity
1-2	Project Idea Created and Group Member Formation
3	Project Proposal and Committee Formation
4	Project Proposal Meeting Held with the Group and Mentor / Revision of Project Proposal
8	Midterm Report
9	Revision of Our Midterm Report
14	Final Report

10.2.2 Senior Design II

Table 10.3 Senior Design 2 Milestones

Weeks	Completed Activity
1-3	Fabricate PCB and Begin Enclosure Construction
4-8	Integrate DSP Algorithms Onto MCU
9-12	Full System Integration and Testing
15	Showcase Final Functional Prototype

10.3 Work Distributions

Table 10.4 Work Distributions

Electrical Engineering	Responsibilities
------------------------	------------------

Noah Bixby	PCB Fabrication
	Amplifier Design
	Power Supply
	Regulators
Electrical Engineering	Responsibilities
Jordan Riley	Amplifier Design
	Bluetooth Hardware
	USB Connectivity
	Speaker Fabrication
Computer Engineering	Responsibilities
Matias Segura	DSP Algorithm
	Auto-EQ Algorithm
	Embedded Programming & Integration
Computer Engineering	Responsibilities
Xander Levine	GUI Design
	UI/UX Design
	Embedded Programming & Integration

This is a tentative table for each team member's responsibilities. Matias's and Xander's work will be related so the two will be sharing some of the other's responsibilities. The same principal applies for Jordan and Noah, especially with the amplifier design.

Chapter 11 Conclusion

The Smart Auto-Equalization Sound System successfully demonstrates the integration of embedded digital signal processing, custom analog hardware, and user-oriented software into a cohesive audio solution. Using the ESP32-S3, calibrated microphone input, and DSP techniques such as FFT-based analysis and adaptive filtering, the system is capable of analyzing a listening environment and generating a tailored equalization profile. This approach addresses a key limitation of traditional audio systems by eliminating the need for manual tuning and specialized expertise.

Throughout the design process, careful consideration was given to amplifier selection, speaker compatibility, enclosure design, and system constraints such as power, cost, and latency. The implementation of Class D amplifier ICs, along with a custom PCB

integrating both analog and digital subsystems, allowed for an efficient and compact design while maintaining acceptable distortion and performance levels. Additionally, the mobile application provides an accessible and user-friendly method for system control and monitoring.

Testing and evaluation of the prototype validated core system functionality, including microphone data acquisition, DSP processing, and audio playback. While some limitations were encountered—particularly in power delivery and signal conditioning, the system achieved its primary objectives and established a strong foundation for further development.

Overall, this project highlights the feasibility and effectiveness of combining real-time embedded processing with audio system design to create an intelligent, adaptive sound system. Future improvements in hardware optimization, algorithm refinement, and system integration will further enhance performance and bring the system closer to a fully realized consumer-ready product.

Appendices

Appendix A - References:

- [1] *Everything RF*, “What is a Class A Amplifier?,” *Everything RF – The RF & Microwave Community*, 2025. Available: <https://www.everythingrf.com/community/what-is-a-class-a-amplifier>. Accessed: Oct. 6, 2025.
- [2] *Electronics Tutorials*, “Class A Amplifier,” *Electronics-Tutorials.ws*, 2025. Available: https://www.electronics-tutorials.ws/amplifier/amp_6.html. Accessed: Oct. 6, 2025.
- [3] *Analog Devices*, “Class D Audio Amplifiers: What, Why, and How,” *Analog Dialogue*, 2025. Available: <https://www.analog.com/en/resources/analog-dialogue/articles/class-d-audio-amplifiers.html>. Accessed: Oct. 6, 2025.
- [4] B. Cordell, *Designing Audio Power Amplifiers*, 2nd ed. Pearl Hi-Fi, 2019. Available: https://pearl-hifi.com/06_Lit_Archive/14_Books_Tech_Papers/Cordell_Bob/Designing_Audio_Pwr_Amps_Cordell.pdf. Accessed: Oct. 6, 2025.
- [5] *MISCO Speakers*, “Speaker Enclosures: Key Design Considerations,” *MISCO Speakers Information Center*, 2025. Available: <https://info.miscospeakers.com/speaker-enclosures-considerations>. Accessed: Oct. 22, 2025.
- [6] *Audio University Online*, “Speaker Crossovers Explained,” *Audio University Online*, 2025. Available: <https://audiouniversityonline.com/speaker-crossovers/>. Accessed: Oct. 22, 2025.

- [7] *Crutchfield*, “What Is a Crossover and Why Do I Need One?,” *Crutchfield Learning Center*, 2025. Available: <https://www.crutchfield.com/S-w25k3BRXdts/learn/car-what-is-a-crossover.html>. Accessed: Oct. 23, 2025.
- [8] *Orah Technologies*, “Understanding Measurement Microphones: Precision Tools for Acoustic Testing,” *Orah Technologies Blog*, 2025. Available: <https://orah.co/understanding-measurement-microphones-precision-tools-for-acoustic-testing/>. Accessed: Oct. 23, 2025.
- [9] *Ansys*, “What Is a System-on-a-Chip (SoC)?,” *Ansys Blog*, 2025. Available: <https://www.ansys.com/blog/what-is-system-on-a-chip>. Accessed: Oct. 29, 2025.
- [10] *IBM*, “What Is a Microcontroller?,” *IBM Think Blog*, 2025. Available: <https://www.ibm.com/think/topics/microcontroller>. Accessed: Oct. 29, 2025.
- [11] *Microchip Technology Inc.*, “Field Programmable Gate Array (FPGA),” *Microchip Education Glossary*, 2025. Available: <https://www.microchip.com/en-us/education/glossary/fpga>. Accessed: Oct. 29, 2025.
- [12] *Espressif Systems*. *ESP32-S3 Series Datasheet*, Ver. 2.0, 24 Apr. 2025, www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf. Accessed 11 Oct. 2025.
- [13] *D. Lemire*, “Why Do We Even Need SIMD Instructions,” *Daniel Lemire’s Blog*, Aug. 9, 2025. Available: <https://lemire.me/blog/2025/08/09/why-do-we-even-need-simd-instructions>. Accessed: Oct. 30, 2025.
- [14] *Espressif Systems*, *ESP32-S3 Series Technical Reference Manual*, Ver. 1.7, Jun. 9, 2025. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf. Accessed: Oct. 11, 2025.
- [15] *Espressif Systems*, “Espressif DSP Library – ESP32,” *Espressif Documentation*, 2025. Available: <https://docs.espressif.com/projects/esp-dsp/en/latest/esp32/esp-dsp-library.html>. Accessed: Oct. 11, 2025.
- [16] *STMicroelectronics N.V.*, *STM32H735xG Microcontroller Datasheet*, DS13312 Ver. 4.0, Nov. 19, 2023. Available: <https://www.st.com/resource/en/datasheet/stm32h735zg.pdf>. Accessed: Oct. 12, 2025.
- [17] *Espressif Systems Co.*, “ESP-DSP Library,” *Espressif Documentation*, 2025. Available: <https://docs.espressif.com/projects/esp-dsp/en/latest/esp32/index.html>. Accessed: Sept. 3, 2025.

- [18] *Not Black Magic: Bits & Pieces*, “DSP Accelerators,” Sept. 24, 2025. Available: <https://notblackmagic.com/bitsnpieces/dsp-accelerators/#introduction>. Accessed: Oct. 12, 2025.
- [19] *Arm, CMSIS-DSP Software Library*, Ver. 1.10.0, May 2, 2022. Available: https://arm-software.github.io/CMSIS_5/DSP/html/index.html. Accessed: Oct. 13, 2025.
- [20] *NXP Semiconductors, i.MX RT1170 Crossover Processor Data Sheet for Consumer Product*, Rev. 1, May 30, 2025. Available: <https://www.nxp.com/docs/en/data-sheet/IMXRT1170BCEC.pdf>. Accessed: Oct. 13, 2025.
- [21] *MOKOBlue*, “Bluetooth vs Wi-Fi: Which Is Better?,” *MOKOBlue Blog*, 2025. Available: <https://www.mokoblue.com/bluetooth-vs-wifi-which-is-better/>. Accessed: Oct. 29, 2025.
- [22] *HowStuffWorks*, “How Bluetooth Works,” *Electronics – HowStuffWorks*, 2025. Available: <https://electronics.howstuffworks.com/bluetooth.htm>. Accessed: Oct. 28, 2025.
- [23] *Scientific American*, “How Does Bluetooth Work?,” *Scientific American – Ask the Experts*, 2025. Available: <https://www.scientificamerican.com/article/experts-how-does-bluetooth-work/>. Accessed: Oct. 28, 2025.
- [24] *Analog Devices*, “Choose the Ideal Bluetooth Protocol for Your Design,” *Analog Technical Articles*, 2025. Available: <https://www.analog.com/en/resources/technical-articles/choose-the-ideal-bluetooth-protocol-for-your-design.html>. Accessed: Oct. 28, 2025.
- [25] *MakeUseOf*, “Bluetooth 5 vs. aptX Low Latency: What’s the Difference?,” *MakeUseOf*, 2025. Available: <https://www.makeuseof.com/bluetooth-5-vs-aptx-low-latency/>. Accessed: Oct. 28, 2025.
- [26] *Silicon Laboratories Inc., Developing Bluetooth Audio*, 2025. Available: <https://www.silabs.com/documents/public/presentations/Developing-Bluetooth-Audio.pdf>. Accessed: Oct. 28, 2025.
- [28] *Nearstream*, “A Guide to Understanding Lossless Audio,” *Nearstream Blog*, 2025. Available: <https://www.nearstream.us/blog/guide-to-understanding-lossless-audio>. Accessed: Oct. 27, 2025.
- [29] *Eaton Tripp Lite*, “USB Connectivity: Types and Standards,” *Eaton Tripp Lite Products & Resources*, 2025. Available: <https://tripplite.eaton.com/products/usb-connectivity-types-standards>. Accessed: Oct. 27, 2025.
- [30] *Newnex Technology Corp.*, “USB Connector Type Guide,” *Newnex Resources*, 2025. Available: <https://newnex.com/usb-connector-type-guide.php>. Accessed: Oct. 27, 2025.

- [31] *Analog Devices*, “Single-Ended to Differential Signal Conversion,” *Analog Dialogue*, 2025. Available: <https://www.analog.com/en/resources/analog-dialogue/articles/single-ended-to-differential-signal-conversion.html>. Accessed: Nov. 23, 2025.
- [32] *Texas Instruments*, *Single-Ended to Differential Conversion Using Fully Differential Amplifiers*, Application Report TIDU038, 2014. Available: <https://www.ti.com/lit/ug/tidu038/tidu038.pdf>. Accessed: Nov. 23, 2025.
- [33] Google, “Flutter: Build Apps for Any Screen,” Flutter.dev, 2025. Available: <https://flutter.dev>. Accessed: Oct. 2, 2025.
- [34] Meta Platforms, Inc., “React Native – Learn Once, Write Anywhere,” ReactNative.dev, 2025. Available: <https://reactnative.dev>. Accessed: Oct. 2, 2025.
- [35] Google, “Material Design Guidelines,” Material Design 3 (M3), 2025. Available: <https://m3.material.io>. Accessed: Oct. 2, 2025.
- [36] OASIS Open, “MQTT Version 5.0,” OASIS Documentation, 2025. Available: <https://docs.oasis-open.org/mqtt>. Accessed: Oct. 2, 2025.
- [37] D. Hardt, “The OAuth 2.0 Authorization Framework,” IETF RFC 6749, Oct. 2012. Available: <https://datatracker.ietf.org/doc/html/rfc6749>. Accessed: Oct. 2, 2025.
- [38] OWASP Foundation, “Mobile Security Testing Guide,” OWASP Project Documentation, 2025. Available: <https://owasp.org/www-project-mobile-security-testing-guide>. Accessed: Oct. 2, 2025.
- [39] AudioCheck.net, “White Noise Test Tone,” AudioCheck – Sound Test Reference, 2025. Available: https://www.audiocheck.net/testtones_whitenoise.php. Accessed: Oct. 30, 2025.
- [40] AudioCheck.net, “Pink Noise Test Tone,” AudioCheck – Sound Test Reference, 2025. Available: https://www.audiocheck.net/testtones_pinknoise.php. Accessed: Oct. 30, 2025.
- [41] AudioCheck.net, “Sine Sweep 20 Hz – 20 kHz Test Tone,” AudioCheck – Sound Test Reference, 2025. Available: https://www.audiocheck.net/testtones_sinesweep20-20k.php. Accessed: Oct. 30, 2025.
- [42] Android Authority, “The Different Bluetooth Versions and What They Mean,” Android Authority, 2025. Available: <https://www.androidauthority.com/bluetooth-versions-3488170/>. Accessed: Oct. 29, 2025.
- [43] Tom’s Hardware, “USB Decoded: All the Specs and Version Numbers,” Tom’s Hardware Features, 2025. Available: <https://www.tomshardware.com/features/usb->

[decoded-all-the-specs-and-version-numbers#section-usb-power-standards](#). Accessed: Oct. 29, 2025.

[44] Digi-Key Electronics, “Decoding the USB Standards from 1.0 to 4.0,” Digi-Key Articles Library, 2025. Available: <https://www.digikey.com/en/articles/decoding-the-usb-standards-from-1-to-4>. Accessed: Oct. 29, 2025.

[45] Digi-Key Electronics, “Decoding the USB Standards from 1.0 to 4.0,” Digi-Key Articles Library, 2025. Available: <https://www.digikey.com/en/articles/decoding-the-usb-standards-from-1-to-4>. Accessed: Oct. 29, 2025.

[46] Dart Dev Documentation, “Effective Dart: Style,” Dart Language Documentation – Flutter Dev, 2025. Available: <https://docs.flutter.dev/effective-dart/style>. Accessed: Oct. 29, 2025.

[47] Ready To Work Japan, “Flutter & Dart Naming Conventions,” Ready To Work Japan Articles, 2025. Available: <https://articles.readytowork.jp/flutter-best-practices-and-coding-guidelines-f494b1ad2369>. Accessed: Oct. 29, 2025.

[48] Flutter Documentation, “Automatic Formatting with dart format,” Flutter Dev Tools Documentation, 2025. Available: <https://docs.flutter.dev/tools/formatting>. Accessed: Oct. 29, 2025.

[49] G. S. V. S. Venkat, “Flutter Coding Standards and Best Practices,” Medium, Oct. 23, 2023. Available: <https://medium.com/@venkat.gsvs/flutter-coding-standards-and-best-practices-84de032413e8>. Accessed: Oct. 29, 2025.

[50] IPC, *IPC-2221B: Generic Standard on Printed Board Design*, IPC, 2012. Accessed: Oct. 29, 2025.

[51] IPC, *IPC-2152: Standard for Determining Current-Carrying Capacity in Printed Boards*, IPC, 2009. Accessed: Oct. 29, 2025.

[52] IPC, *IPC-6012D: Qualification and Performance Specification for Rigid Printed Boards*, IPC, 2015. Accessed: Oct. 29, 2025.

[53] Espressif Systems, *ESP32-S3 Hardware Design Guidelines*, Espressif Documentation, 2023. Accessed: Oct. 29, 2025.

[54] Microchip Technology, *BM83 Bluetooth Audio Module Hardware User Guide*, Microchip, 2021. Accessed: Oct. 29, 2025.

[55] Flutter Documentation, “App Architecture Guide,” Flutter Dev Documentation, 2025. Available: <https://docs.flutter.dev/app-architecture/guide>. Accessed: Oct. 29, 2025.

- [56] Flutter Dev Docs, “Performance Best Practices,” Flutter Documentation, 2025. Available: <https://docs.flutter.dev/perf/best-practices>. Accessed: Oct. 29, 2025.
- [57] Acoustical Society of America, “Room Acoustics and Sound Reflection,” ASA Topics in Acoustics, 2025. Available: <https://asa.scitation.org/topic/acoustics>. Accessed: Oct. 29, 2025.
- [58] Audioholics Magazine, “Guide to Speaker Placement for Optimal Sound,” Audioholics Loudspeaker Design Articles, 2025. Available: <https://www.audioholics.com/loudspeaker-design/speaker-placement-guidelines>. Accessed: Oct. 29, 2025.
- [59] Sound on Sound, “Crossover Design and Implementation,” Sound on Sound – Techniques, 2025. Available: <https://www.soundonsound.com/techniques/crossover-design-basics>. Accessed: Oct. 29, 2025.
- [60] Crutchfield, “Speaker Size and Enclosure Design Considerations,” Crutchfield Tech Articles, 2025. Available: <https://www.crutchfield.com/learn/speaker-box-design.html>. Accessed: Oct. 29, 2025.
- [61] Texas Instruments, “Audio Processing Performance Optimization,” TI Application Note SLA447A, 2025. Available: <https://www.ti.com/lit/an/slaa447a/slaa447a.pdf>. Accessed: Oct. 29, 2025.
- [62] NXP Semiconductors, “Low-Latency Audio in Embedded Systems,” NXP Technical Brief, 2025. Available: <https://community.nxp.com/docs/DOC-107753>. Accessed: Oct. 29, 2025.
- [63] ISO, “Equal-Loudness Contours and Psychoacoustics – ISO 226:2003 Standard Overview,” ISO Standards Catalogue, 2025. Available: <https://www.iso.org/standard/34222.html>. Accessed: Oct. 29, 2025.
- [64] Nordic Semiconductor, “Bluetooth Low Energy Latency Performance Analysis,” Nordic Documentation Portal, 2025. Available: <https://infocenter.nordicsemi.com>. Accessed: Oct. 29, 2025.
- [65] Arm Ltd., *ARM Cortex-M Series Technical Reference Manual*, 2025. Available: <https://developer.arm.com/documentation/100230/latest/>. Accessed: Oct. 29, 2025.
- [66] Flutter Documentation, “Platform Channels: Flutter and Native Communication,” Flutter Dev Documentation, 2025. Available: <https://docs.flutter.dev/development/platform-integration/platform-channels>. Accessed: Oct. 29, 2025.

[67] Room EQ Wizard Help, “Graph Smoothing (Variable, Psychoacoustic, ERB),” Room EQ Wizard Documentation, 2025. Available: https://www.roomeqwizard.com/help/help_en-GB/html/graph.html. Accessed: Oct. 30, 2025.

[68] Keysight Technologies, “What Is an FFT (Fast Fourier Transform)?,” Keysight Knowledge Center, 2025. Available: <https://www.keysight.com/used/us/en/knowledge/glossary/oscilloscopes/what-is-a-fft-fast-fourier-transform>. Accessed: Oct. 30, 2025.

[69] Zölzer, Udo. *Digital Audio Signal Processing*, 2nd ed., Wiley, 2008. Accessed: Oct. 30, 2025.

[70] Fiveable, “Wiener Filtering,” Fiveable Inc., Sept. 2025. Available: <https://fiveable.me/advanced-signal-processing/unit-4/wiener-filtering/study-guide/3vYp5KdCY6d8FzSQ>. Accessed: Nov. 22, 2025.

[71] Uncini, Aurelio. *Digital Audio Processing Fundamentals.*, Springer, 2022. Accessed: Oct. 30, 2025.

[72] Iain Explains Signals, Systems, and Digital Comms, “What is Windowing in Digital Processing?” YouTube, 25 Nov 2024. Available: <https://www.youtube.com/watch?v=1Hd72RpMFIQ>. Accessed: Oct. 30, 2025.

[73] Wikipedia, “XLR connector,” Wikipedia, 2025. Available: https://en.wikipedia.org/wiki/XLR_connector. Accessed: Nov. 23, 2025.

[74] Cadence PCB Design Blog, “What Is a Star Ground Layout and Why Do You Need It?,” 2020. Available: <https://resources.pcb.cadence.com/blog/2020-what-is-a-star-ground-layout-and-why-do-you-need-it>. Accessed: Nov. 22, 2025.

[75] ALLPCB, “The Role of Copper Weight in 4-Layer PCBs,” ALLPCB ElectroHub, 2025. Available: <https://www.allpcb.com/allelectrohub/the-role-of-copper-weight-in-4-layer-pcbs-balancing-current-capacity-and-signal-integrity>. Accessed: Nov. 22, 2025.

Appendix B – Copyright permission: (if external diagrams/images used).

- (1) A. I. Brilliance, “From Turing to Today: A Brief History of AI,” *AI Brilliance Blog*, 2024. Available: <https://www.aibrilliance.com/blog/from-turing-to-today-a-brief-history-of-ai>. Accessed: Oct. 4, 2025.

Function	ESP32	ESP32S3	ESP32P4	ANSI Xtensa	ANSI Risc-V
Optimization	O2	O2	O2	O2	O2
Dot Product					
dsps_dotprod_f32 for N=256 points	1047	432	1319	1311	1563
dsps_dotprod_f32 for N=256 points with step 1	1307	1307	1314	2581	1819
dsps_dotprod_s16 for N=256 points	437	307	208	3635	3385
FIR Filters					
dsps_fir_f32 1024 input samples and 256 coefficients	1078691	443671	1606563	1353122	1606552
dsps_fir_f32 1024 samples 256 coeffs and decimation 4	276504	115499	339796	350726	415332

(2)

Appendix C – Datasheet (if necessary):

(1c) STMicroelectronics, “STM32H7 Series High-Performance MCUs,” ST.com, 2025. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html> Accessed: Oct. 4, 2025

(2c) STMicroelectronics, “ST67W611M1 Bluetooth 5.4 Module Datasheet,” ST.com, 2025. Available: <https://www.st.com/en/wireless-transceivers/st67w611m1.html> Accessed: Oct. 5, 2025.

Texas Instruments. LM25118 Wide Voltage Range Buck-Boost Controller Datasheet. Rev. F, Mar. 2018. Texas Instruments, www.ti.com/lit/ds/symlink/lm25118.pdf.

LM25088, LM25088-Q1 Wide Input Range Non-Synchronous Buck Controllers

LMR51625 and LMR51635 Functional Safety FIT Rate, FMD and Pin FMA

TPS54062 4.7-V to 60-V Input, 50-MA Synchronous Step-down Converter with Low IQ.

Texas Instruments. TPA3116D2 15-W, 30-W, 50-W Filter-Free Class-D Stereo Amplifier Family With AM Avoidance (Rev. G). Texas Instruments, Dec. 2017

TPS54062 4.7-V to 60-V Input, 50-MA Synchronous Step-down Converter with Low IQ.

8-Channel DAC with PLL and Single-Ended Outputs, 192 KHz, 24 Bits.

Advanced BM62/64 Features.

L R S -6 0 0 N 2 S E R I E S.

Appendix D – Software Code (if necessary):

main.dart for application

```
import 'dart:async';
import 'dart:convert';
import 'dart:typed_data';

import 'package:flutter/material.dart';
import 'package:flutter_web_bluetooth/flutter_web_bluetooth.dart';

import 'pages/home_page.dart';
import 'pages/control_page.dart';
import 'pages/settings_page.dart';

const String kServiceUuid = "000000ff-0000-1000-8000-00805f9b34fb";
const String kRxCharUuid = "0000ff01-0000-1000-8000-00805f9b34fb";

void main() {
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({super.key});
  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  int _currentIndex = 0;

  bool _connecting = false;
  bool _connected = false;
  String _status = 'Not connected';

  BluetoothDevice? _device;
  BluetoothService? _eqService;
```

```

BluetoothCharacteristic? _rxChar;

double _volume = 0.5;
final List<double> _eqBands = List<double>.filled(5, 0.0);

bool _bluetoothOn = true;
String _wifiSsid = 'MyWiFi';
String _room = 'Living Room';
double _latencyMs = 50;

@override
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    theme: ThemeData.dark(),
    home: Scaffold(
      body: SafeArea(child: _buildBody()),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentIndex,
        selectedItemColor: Colors.deepPurpleAccent,
        unselectedItemColor: Colors.white54,
        onTap: (i) => setState(() => _currentIndex = i),
        items: const [
          BottomNavigationBarItem(icon: Icon(Icons.home), label:
"Home"),
          BottomNavigationBarItem(icon: Icon(Icons.equalizer), label:
"Control"),
          BottomNavigationBarItem(icon: Icon(Icons.settings), label:
"Settings"),
        ],
      ),
    ),
  );
}

Widget _buildBody() {
  switch (_currentIndex) {
    case 0:
      return HomePage(
        connected: _connected,
        connecting: _connecting,
        status: _status,
        onToggleConnect: _toggleConnect,

```

```

    );
    case 1:
        return ControlPage(
            connected: _connected,
            volume: _volume,
            onVolumeChanged: (v) {
                setState(() => _volume = v);
                _sendVolumeCommand(v);
            },
            eqValues: _eqBands,
            onEqChanged: (i, v) {
                setState(() => _eqBands[i] = v);
                _sendEqCommand(i, v);
            },
            onResetEq: () {
                setState(() {
                    for (int i = 0; i < _eqBands.length; i++) _eqBands[i] = 0.0;
                });
                _sendResetEqCommand();
            },
        );
    case 2:
        return SettingsPage(
            bluetoothOn: _bluetoothOn,
            onBluetooth: (b) => setState(() => _bluetoothOn = b),
            wifiSsid: _wifiSsid,
            onWifiSsid: (s) => setState(() => _wifiSsid = s),
            room: _room,
            onRoom: (r) => setState(() => _room = r),
            latencyMs: _latencyMs,
            onLatency: (l) => setState(() => _latencyMs = l),
        );
    default:
        return const SizedBox.shrink();
    }
}

Future<void> _toggleConnect() async {
    if (_connected) {
        await _disconnectWeb();
    } else {
        await _connectWeb();
    }
}

```

```

}

Future<void> _connectWeb() async {
  if (_connecting) return;

  setState(() {
    _connecting = true;
    _status = 'Requesting device...';
  });

  BluetoothDevice device;

  try {
    final opts =
RequestOptionsBuilder.acceptAllDevices(optionalServices: [kServiceUuid]);
    device = await FlutterWebBluetooth.instance.requestDevice(opts);
  } catch (e) {
    debugPrint("User canceled chooser.");
    setState(() {
      _connecting = false;
      _status = "User canceled";
    });
    return;
  }

  debugPrint("Device chosen: ${device.name}");

  try {
    await device.connect();
  } catch (e) {
    debugPrint("connect() failed: $e");
    setState(() {
      _connecting = false;
      _status = "Connect failed";
    });
    return;
  }

  // FIXED: Real boolean check
  if (device.gatt?.connected != true) {
    debugPrint("GATT not connected.");
    setState(() {
      _connecting = false;

```

```

        _status = "Failed GATT connect";
    });
    return;
}

List<BluetoothService> services;
try {
    services = await device.discoverServices();
} catch (e) {
    debugPrint("discoverServices failed: $e");
    setState(() {
        _connecting = false;
        _status = "Service discover fail";
    });
    return;
}

BluetoothService? eqService;
for (final s in services) {
    if (s.uuid.toLowerCase() == kServiceUuid.toLowerCase()) {
        eqService = s;
    }
}
if (eqService == null) {
    setState(() {
        _connecting = false;
        _status = "EQ service missing";
    });
    return;
}

BluetoothCharacteristic? rx;
try {
    rx = await eqService.getCharacteristic(kRxCharUuid);
} catch (e) {
    debugPrint("RX char fail: $e");
    setState(() {
        _connecting = false;
        _status = "RX missing";
    });
    return;
}

```

```

        setState(() {
            _device = device;
            _eqService = eqService;
            _rxChar = rx;
            _connected = true;
            _connecting = false;
            _status = "Connected";
        });

        debugPrint("BLE CONNECTED SUCCESSFULLY");
    }

    Future<void> _disconnectWeb() async {
        try {
            _device?.disconnect();
        } catch (_) {}

        setState(() {
            _connected = false;
            _device = null;
            _rxChar = null;
            _eqService = null;
            _status = "Disconnected";
        });

        debugPrint("Disconnected.");
    }

    bool get _canSend => _connected && _rxChar != null;

    Future<void> _writeString(String text) async {
        debugPrint("BLE -> ESP32: $text");

        if (!_canSend) return;

        try {
            final data = Uint8List.fromList(utf8.encode(text));
            await _rxChar!.writeValueWithoutResponse(data);
            setState(() => _status = "Sent: $text");
        } catch (e) {
            debugPrint("SEND ERROR: $e");
        }
    }
}

```

```

Future<void> _sendEqCommand(int band, double value) async {
  await _writeString("EQ:$band:${value.toStringAsFixed(1)}");
}

Future<void> _sendResetEqCommand() async {
  await _writeString("EQ:RESET");
}

Future<void> _sendVolumeCommand(double value) async {
  await _writeString("VOL:${(value * 100).toInt()}");
}
}

```

eq_visualizer.dart for mobile application

```

import 'package:flutter/material.dart';

class EqVisualizer extends StatelessWidget {
  const EqVisualizer({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 120,
      decoration: BoxDecoration(
        color: Color(0xFF141722),
        borderRadius: BorderRadius.circular(12),
      ),
      child: const Center(
        child: Text("Waiting for data..."),
      ),
    );
  }
}

```

eq_sliders.dart for mobile application

```

import 'package:flutter/material.dart';

```

```

class EqSliders extends StatelessWidget {
  final List<double> values;
  final bool enabled;
  final void Function(int, double) onChange;
  final VoidCallback onResetPressed;

  const EqSliders({
    super.key,
    required this.values,
    required this.enabled,
    required this.onChange,
    required this.onResetPressed,
  });

  static const labels = ["60 Hz", "250 Hz", "1 kHz", "4 kHz", "12 kHz"];

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(
          height: 220,
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: List.generate(5, (i) {
              return Column(
                children: [
                  Text(labels[i]),
                  Expanded(
                    child: RotatedBox(
                      quarterTurns: -1,
                      child: Slider(
                        min: -12,
                        max: 12,
                        value: values[i],
                        onChanged: enabled ? (v) => onChange(i, v) : null,
                      ),
                    ),
                  Text("${values[i].toStringAsFixed(1)} dB")
                ],
              );
            })
          ),
        ],
      );
  }
}

```

```

        )),
      ),
    ),
    ElevatedButton(
      onPressed: enabled ? onResetPressed : null,
      child: Text("Reset EQ"),
    )
  ],
);
}
}

```

home_page.dart for mobile application

```

import 'package:flutter/material.dart';

class HomePage extends StatelessWidget {
  final bool connected;
  final bool connecting;
  final String status;
  final VoidCallback onToggleConnect;

  const HomePage({
    super.key,
    required this.connected,
    required this.connecting,
    required this.status,
    required this.onToggleConnect,
  });

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(24),
      child: Column(
        children: [
          const Text("Smart Auto-EQ", style: TextStyle(fontSize: 24)),
          const SizedBox(height: 20),
          Icon(
            connected ? Icons.bluetooth_connected :
Icons.bluetooth_disabled,

```

```

        size: 60,
      ),
      const SizedBox(height: 10),
      Text(status),
      const SizedBox(height: 20),
      ElevatedButton(
        onPressed: connecting ? null : onToggleConnect,
        child: Text(connected ? "Disconnect" : "Connect"),
      ),
    ],
  ),
);
}
}

```

control_page.dart for mobile application

```

import 'package:flutter/material.dart';
import '../widgets/eq_sliders.dart';

class ControlPage extends StatelessWidget {
  final bool connected;
  final double volume;
  final ValueChanged<double> onVolumeChanged;
  final List<double> eqValues;
  final void Function(int, double) onEqChanged;
  final VoidCallback onResetEq;

  const ControlPage({
    super.key,
    required this.connected,
    required this.volume,
    required this.onVolumeChanged,
    required this.eqValues,
    required this.onEqChanged,
    required this.onResetEq,
  });

  @override
  Widget build(BuildContext context) {
    final enabled = connected;

```

```

return SingleChildScrollView(
  padding: const EdgeInsets.all(16),
  child: Column(
    children: [
      const Text("Master Volume", style: TextStyle(fontSize: 18)),
      Slider(
        value: volume,
        onChanged: enabled ? onVolumeChanged : null,
      ),
      Text("${(volume * 100).toInt()}%"),

      const SizedBox(height: 20),
      const Text("Graphic EQ ±12 dB", style: TextStyle(fontSize: 18)),
      Container(
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: Color(0xFF141722),
          borderRadius: BorderRadius.circular(12),
        ),
        child: EqSliders(
          values: eqValues,
          enabled: enabled,
          onChange: onEqChanged,
          onResetPressed: onResetEq,
        ),
      ),
      if (!connected)
        const Padding(
          padding: EdgeInsets.only(top: 12),
          child: Text("Not connected", style: TextStyle(color:
Colors.red)),
        )
    ],
  ),
);
}
}

```

main.cpp ESP32 code for BLE connectivity

```

// main.cpp -- ESP32-S3 BLE UART
extern "C" {
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#include "esp_system.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "esp_bt.h"
#include "esp_bt_main.h"
#include "esp_gap_ble_api.h"
#include "esp_gatts_api.h"
#include "esp_gatt_common_api.h"
}

#include <string.h>    // for memcpy

static const char *TAG = "SD1_BLE_UART";

/* ----- BLE CONFIG ----- */
static const char *DEVICE_NAME = "esp32s3";

// Option B layout: 16-bit service + characteristics
static const uint16_t GATTS_SERVICE_UUID = 0x00FF;
static const uint16_t GATTS_CHAR_UUID_RX = 0xFF01;    // phone writes here
static const uint16_t GATTS_CHAR_UUID_TX = 0xFF02;    // ESP notifies here

// service + RX (decl,val) + TX (decl,val,cccd)
#define GATTS_NUM_HANDLE 6
#define PROFILE_APP_ID 0

/* Attribute index definitions */
enum {
    IDX_SVC,

    IDX_CHAR_RX,
    IDX_CHAR_VAL_RX,

    IDX_CHAR_TX,
    IDX_CHAR_VAL_TX,
    IDX_CHAR_TX_CFG,

```

```

        IDX_NB,
    };

    static uint16_t gatt_handle_table[IDX_NB];

    /* Standard UUIDs */
    static const uint16_t primary_service_uuid =
        ESP_GATT_UUID_PRI_SERVICE;
    static const uint16_t character_declaration_uuid =
        ESP_GATT_UUID_CHAR_DECLARE;
    static const uint16_t client_character_cfg_uuid =
        ESP_GATT_UUID_CHAR_CLIENT_CONFIG;

    /* Characteristic properties */
    static const uint8_t char_prop_write =
        ESP_GATT_CHAR_PROP_BIT_WRITE | ESP_GATT_CHAR_PROP_BIT_WRITE_NR;
    static const uint8_t char_prop_notify =
        ESP_GATT_CHAR_PROP_BIT_NOTIFY;

    /* Initial values */
    static uint8_t rx_char_val[20] = {0};
    static uint8_t tx_char_val[20] = {0};
    static uint8_t tx_cccd_val[2] = {0x00, 0x00}; // notifications disabled
    by default

    /* GATT attribute database */
    static esp_gatts_attr_db_t gatt_db[IDX_NB] = {

        // Service declaration
        [IDX_SVC] = {
            {ESP_GATT_AUTO_RSP},
            {
                ESP_UUID_LEN_16,
                (uint8_t *)&primary_service_uuid,
                ESP_GATT_PERM_READ,
                sizeof(uint16_t),
                sizeof(uint16_t),
                (uint8_t *)&GATTS_SERVICE_UUID
            }
        },

        // RX characteristic declaration (Write / Write NR)
        [IDX_CHAR_RX] = {

```

```

        {ESP_GATT_AUTO_RSP},
        {
            ESP_UUID_LEN_16,
            (uint8_t *)&character_declaration_uuid,
            ESP_GATT_PERM_READ,
            sizeof(uint8_t),
            sizeof(uint8_t),
            (uint8_t *)&char_prop_write
        }
    },

    // RX characteristic value
    [IDX_CHAR_VAL_RX] = {
        {ESP_GATT_AUTO_RSP},
        {
            ESP_UUID_LEN_16,
            (uint8_t *)&GATTS_CHAR_UUID_RX,
            ESP_GATT_PERM_WRITE,
            sizeof(rx_char_val),
            sizeof(rx_char_val),
            (uint8_t *)rx_char_val
        }
    },

    // TX characteristic declaration (Notify)
    [IDX_CHAR_TX] = {
        {ESP_GATT_AUTO_RSP},
        {
            ESP_UUID_LEN_16,
            (uint8_t *)&character_declaration_uuid,
            ESP_GATT_PERM_READ,
            sizeof(uint8_t),
            sizeof(uint8_t),
            (uint8_t *)&char_prop_notify
        }
    },

    // TX characteristic value
    [IDX_CHAR_VAL_TX] = {
        {ESP_GATT_AUTO_RSP},
        {
            ESP_UUID_LEN_16,
            (uint8_t *)&GATTS_CHAR_UUID_TX,

```

```

        ESP_GATT_PERM_READ,
        sizeof(tx_char_val),
        sizeof(tx_char_val),
        (uint8_t *)tx_char_val
    }
},

// TX CCCD (Client Characteristic Configuration Descriptor)
[IDX_CHAR_TX_CFG] = {
    {ESP_GATT_AUTO_RSP},
    {
        ESP_UUID_LEN_16,
        (uint8_t *)&client_character_cfg_uuid,
        ESP_GATT_PERM_READ | ESP_GATT_PERM_WRITE,
        sizeof(uint16_t),
        sizeof(tx_cccd_val),
        (uint8_t *)tx_cccd_val
    }
},
};

/* ----- RAW ADVERTISING PAYLOAD -----
- Flags:      0x02, 0x01, 0x06
- Name:       "esp32s3"
- 16-bit UUID: 0x00FF
*/
static uint8_t adv_raw_data[] = {
    // Flags
    0x02, 0x01, 0x06,
    // Complete Local Name: "esp32s3"
    0x08, 0x09, 'e','s','p','3','2','s','3',
    // Complete List of 16-bit Service UUIDs: 0x00FF (little-endian: 0xFF
    0x00)
    0x03, 0x03, 0xFF, 0x00,
};

/* Advertising parameters */
static esp_ble_adv_params_t adv_params = {
    .adv_int_min      = 0x20,
    .adv_int_max      = 0x40,
    .adv_type          = ADV_TYPE_IND,
    .own_addr_type     = BLE_ADDR_TYPE_PUBLIC,
    .channel_map       = ADV_CHNL_ALL,

```

```

        .adv_filter_policy = ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY,
    };

    /* State */
    static esp_gatt_if_t g_gatts_if      = ESP_GATT_IF_NONE;
    static uint16_t      g_conn_id       = 0;
    static bool          g_notify_enabled = false;

    /* ----- GAP EVENT HANDLER ----- */
    static void gap_event_handler(esp_gap_ble_cb_event_t event,
    esp_ble_gap_cb_param_t *param)
    {
        switch (event) {

            case ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT:
                ESP_LOGI(TAG, "Raw advertising data set, starting advertising");
                esp_ble_gap_start_advertising(&adv_params);
                break;

            case ESP_GAP_BLE_ADV_START_COMPLETE_EVT:
                if (param->adv_start_cmpl.status == ESP_BT_STATUS_SUCCESS) {
                    ESP_LOGI(TAG, "Advertising started successfully");
                } else {
                    ESP_LOGE(TAG, "Failed to start advertising: 0x%x",
                        param->adv_start_cmpl.status);
                }
                break;

            case ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT:
                ESP_LOGI(TAG, "Advertising stopped");
                break;

            case ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT:
                ESP_LOGI(TAG, "Conn params updated: min_int=%d max_int=%d",
                    param->update_conn_params.min_int,
                    param->update_conn_params.max_int);
                break;

            default:
                break;
        }
    }
}

```

```

/* ----- GATTS EVENT HANDLER ----- */
static void gatts_event_handler(esp_gatts_cb_event_t event,
                                esp_gatt_if_t gatts_if,
                                esp_ble_gatts_cb_param_t *param)
{
    if (event == ESP_GATTS_REG_EVT) {
        g_gatts_if = gatts_if;
    }

    switch (event) {

    case ESP_GATTS_REG_EVT: {
        ESP_LOGI(TAG, "REG_EVT app_id=%d", param->reg.app_id);

        // Set device name (shows in scan results)
        esp_err_t name_ret = esp_ble_gap_set_device_name(DEVICE_NAME);
        if (name_ret != ESP_OK) {
            ESP_LOGE(TAG, "esp_ble_gap_set_device_name failed: 0x%x",
name_ret);
        }

        // Configure raw advertising payload
        esp_err_t adv_ret = esp_ble_gap_config_adv_data_raw(adv_raw_data,
sizeof(adv_raw_data));
        if (adv_ret != ESP_OK) {
            ESP_LOGE(TAG, "esp_ble_gap_config_adv_data_raw failed: 0x%x",
adv_ret);
            // NOTE: do NOT abort here; we still want firmware to run and
log.
        }

        // Create GATT attribute table
        esp_err_t tbl_ret = esp_ble_gatts_create_attr_tab(gatt_db,
gatts_if, IDX_NB, PROFILE_APP_ID);
        if (tbl_ret != ESP_OK) {
            ESP_LOGE(TAG, "esp_ble_gatts_create_attr_tab failed: 0x%x",
tbl_ret);
        }
        break;
    }

    case ESP_GATTS_CREAT_ATTR_TAB_EVT:
        if (param->add_attr_tab.status != ESP_GATT_OK) {

```

```

        ESP_LOGE(TAG, "Create attr table failed, error 0x%x",
                  param->add_attr_tab.status);
        break;
    }
    if (param->add_attr_tab.num_handle != IDX_NB) {
        ESP_LOGE(TAG, "Attr table handle num mismatch (%d != %d)",
                  param->add_attr_tab.num_handle, IDX_NB);
        break;
    }
    memcpy(gatt_handle_table, param->add_attr_tab.handles,
           sizeof(gatt_handle_table));
    ESP_LOGI(TAG, "GATT attribute table created, starting service");
    esp_ble_gatts_start_service(gatt_handle_table[IDX_SVC]);
    break;

case ESP_GATTS_CONNECT_EVT:
    g_conn_id = param->connect.conn_id;
    ESP_LOGI(TAG, "Client connected, conn_id=%d", g_conn_id);
    break;

case ESP_GATTS_DISCONNECT_EVT:
    ESP_LOGI(TAG, "Client disconnected, restarting advertising");
    g_notify_enabled = false;
    esp_ble_gap_start_advertising(&adv_params);
    break;

case ESP_GATTS_WRITE_EVT: {
    uint16_t handle = param->write.handle;
    uint16_t len     = param->write.len;
    uint8_t *data    = param->write.value;

    if (handle == gatt_handle_table[IDX_CHAR_VAL_RX]) {
        // Data from phone (RX characteristic)
        ESP_LOGI(TAG, "RX write (%d bytes)", len);
        ESP_LOG_BUFFER_HEXDUMP(TAG, data, len, ESP_LOG_INFO);

        // TODO (for Option B): forward this to BM83 UART later
    }
    else if (handle == gatt_handle_table[IDX_CHAR_TX_CFG]) {
        // CCCD write (enable/disable notifications)
        if (len == 2) {
            uint16_t cfg = (data[1] << 8) | data[0];
            g_notify_enabled = (cfg & 0x0001);
        }
    }
}

```

```

        ESP_LOGI(TAG, "Notifications %s",
                  g_notify_enabled ? "ENABLED" : "DISABLED");
    }
}
break;
}

default:
    break;
}
}

/* ----- APP MAIN ----- */
extern "C" void app_main(void)
{
    esp_err_t ret;

    ESP_LOGI(TAG, "Starting ESP32-S3 BLE UART (device \"%s\")",
DEVICE_NAME);

    // Initialize NVS (required by BT stack)
    ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }

    // Free classic BT memory (we only use BLE)
    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT))
;

    // Init and enable BT controller in BLE mode
    esp_bt_controller_config_t bt_cfg =
BT_CONTROLLER_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_bt_controller_init(&bt_cfg));
    ESP_ERROR_CHECK(esp_bt_controller_enable(ESP_BT_MODE_BLE));

    // Init and enable Bluedroid
    ESP_ERROR_CHECK(esp_bluedroid_init());
    ESP_ERROR_CHECK(esp_bluedroid_enable());

    // Register BLE callbacks

```

```

    ESP_ERROR_CHECK(esp_ble_gap_register_callback(gap_event_handler));
    ESP_ERROR_CHECK(esp_ble_gatts_register_callback(gatts_event_handler));
    ESP_ERROR_CHECK(esp_ble_gatts_app_register(PROFILE_APP_ID));

    // Optional: set MTU to something larger than default
    ESP_ERROR_CHECK(esp_ble_gatt_set_local_mtu(247));

    ESP_LOGI(TAG, "Initialization complete. Waiting for BLE
connections...");
    // Everything else is handled in callbacks.
}

```

main.cpp for microphone demo

```

#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_err.h"

#include "esp_adc/adc_continuous.h"
#include "driver/gpio.h"
#include "hal/adc_types.h"
#include "hal/gpio_types.h"
#include "soc/gpio_num.h"

static const char *TAG = "ADC";

// ADC sampling parameters
#define SAMPLE_RATE_HZ      44100           // 44.1kHz sampling rate
#define ADC_READ_LEN        1024           // DMA read buffer size in
bytes
#define CHANNEL_TO_USE      ADC_CHANNEL_5   // ADC1_CH3 -> GPIO4 on S3
module
#define ADC_UNIT            ADC_UNIT_1     // ADC 1
#define ADC_ATTEN           ADC_ATTEN_DB_12 // use DB_11 for S3

// Output format parsing helpers

```

```

#define MY_ADC_OUTPUT_TYPE      ADC_DIGI_OUTPUT_FORMAT_TYPE2
#define MY_ADC_GET_CHANNEL(p)   ((p)->type2.channel)
#define MY_ADC_GET_DATA(p)      ((p)->type2.data)

static TaskHandle_t s_task_handle = NULL;

// -----
// -----
// ----- ISR CALLBACK -----
// -----
// -----
static IRAM_ATTR bool cont_cb(adc_continuous_handle_t handle, const
adc_continuous_evt_data_t *edata, void *user_data)
{
    // Notify the main task that conversion data is ready
    BaseType_t mustYield = pdFALSE;
    vTaskNotifyGiveFromISR(s_task_handle, &mustYield);
    return (mustYield == pdTRUE);
}

// -----
// -----
// ----- MAIN -----
// -----
// -----
extern "C" void app_main(void)
{
    gpio_config_t gcfg = {
        .pin_bit_mask   = (1ULL << GPIO_NUM_6),    // Use shifting to
mask GPIO pin 6
        .mode            = GPIO_MODE_INPUT,          // Input as we are
gathering data
        .pull_up_en      = GPIO_PULLUP_DISABLE,     // Disable pull-up
        .pull_down_en    = GPIO_PULLDOWN_DISABLE,   // Disable pull-down
        .intr_type        = GPIO_INTR_DISABLE       // Disable Interrupts
for now
    };

    // Apply GPIO Configuration
    gpio_config(&gcfg);

```

```

s_task_handle = xTaskGetCurrentTaskHandle();

// ADC continuous handle configuration
adc_continuous_handle_t adc_handle = NULL;

adc_continuous_handle_cfg_t handle_cfg = {
    .max_store_buf_size = 4096,
    .conv_frame_size = 1024,
};
handle_cfg.flags.flush_pool = 1;
ESP_ERROR_CHECK(adc_continuous_new_handle(&handle_cfg, &adc_handle));

// Pattern config for single channel (ADC1_CH5 -> GPIO6)
adc_digi_pattern_config_t pat = {
    .atten      = ADC_ATTEN_DB_12,
    .channel    = ADC_CHANNEL_5,
    .unit       = ADC_UNIT,
    .bit_width  = ADC_BITWIDTH_12,
};

adc_continuous_config_t config = {
    .pattern_num    = 1,
    .adc_pattern    = &pat,
    .sample_freq_hz = SAMPLE_RATE_HZ,
    .conv_mode      = ADC_CONV_SINGLE_UNIT_1,
    .format         = MY_ADC_OUTPUT_TYPE,
};

// Apply Continuous Read configuration
ESP_ERROR_CHECK(adc_continuous_config(adc_handle, &config));

// Register callback and start ADC
adc_continuous_evt_cbs_t cbs = {
    .on_conv_done  = cont_cb,
    .on_pool_ovf   = cont_cb
};

ESP_ERROR_CHECK(adc_continuous_register_event_callbacks(adc_handle,
&cbs, NULL));
ESP_ERROR_CHECK(adc_continuous_start(adc_handle));

// Allocate DMA read buffer on heap

```

```

    uint8_t *dma_buf = (uint8_t*)heap_caps_malloc(ADC_READ_LEN,
MALLOC_CAP_8BIT);
    if (!dma_buf) {
        ESP_LOGE(TAG, "Failed to allocate dma_buf");
        return;
    }
    memset(dma_buf, 0, ADC_READ_LEN);

    ESP_LOGI(TAG, "Start ADC");

    while (true) {
        // Wait for a conversion-done notification from ISR
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY);

        // Read available bytes
        uint32_t out_len = 0;
        esp_err_t r = adc_continuous_read(adc_handle, dma_buf,
ADC_READ_LEN, &out_len, 0);
        if (r == ESP_ERR_TIMEOUT) {
            continue;
        }

        const size_t pkt_bytes = SOC_ADC_DIGI_RESULT_BYTES; // For the S3
it's 4 bytes
        size_t pkt_count = out_len / pkt_bytes;

        for (size_t i = 0; i < pkt_count; ++i) {
            adc_digi_output_data_t *p = (adc_digi_output_data_t*)(dma_buf
+ i * pkt_bytes);
            uint16_t raw = MY_ADC_GET_DATA(p);        // 12-bit ADC code
            float out = (3300.0 / 4095.0) * raw ;    // Compute Vdata using
conversion formula from Datasheet
            ESP_LOGI(TAG, "Data[%d] = %f", i, out); // Print the data
            vTaskDelay(10);                        // Wait 10 ticks for next print
        }
        vTaskDelay(100000);                        // Delay to look at previous data
    }
}

```

main.cpp for BM83, DAC, & Amplifier demonstration

```

#include "driver/i2s_common.h"
#include "driver/i2s_types.h"
#include "esp_log.h"
#include "driver/i2s_std.h"
#include "freertos/projdefs.h"
#include "hal/i2s_types.h"
#include "freertos/FreeRTOS.h"
#include "esp_task.h"
#include "portmacro.h"
#include "soc/clk_tree_defs.h"
#include "soc/gpio_num.h"
#include <stdint.h>

static const char *TAG = "I2S_DEMO";
static i2s_chan_handle_t mcu_rx = NULL;
static i2s_chan_handle_t mcu_tx = NULL;

#define SAMPLE_RATE      48000                      // Sampling Rate (Matched
with BM83)
#define BUFFER_FRAMES    8192                      // frames (stereo frames)
captured and transmitted
#define BYTES_PER_SAMPLE 2                          // 16-bit => 2 bytes per channel
sample
#define CHANNELS          2
#define FRAME_SIZE_BYTES (BYTES_PER_SAMPLE * CHANNELS) // 4 bytes per frame
(16-bit stereo)
#define BUFFER_BYTES      (BUFFER_FRAMES * FRAME_SIZE_BYTES)

// -----
// -----
// ----- I2S CONNECTION FOR BT TO MCU -----
// -----
// -----

static void i2s_bt_to_mcu_to_dac_init ()
{
    i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0,
I2S_ROLE_MASTER);

    ESP_ERROR_CHECK(i2s_new_channel(&chan_cfg, &mcu_tx, &mcu_rx));

    i2s_std_clk_config_t clk_config = I2S_STD_CLK_DEFAULT_CONFIG(SAMPLE_RATE);
    clk_config.mclk_multiple = I2S_MCLK_MULTIPLE_384;

    i2s_std_config_t std_cfg = {
        .clk_cfg = clk_config,
        .slot_cfg =
I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_24BIT,
I2S_SLOT_MODE_STEREO),
        .gpio_cfg = {
            .mclk = I2S_GPIO_UNUSED,
            .bclk = GPIO_NUM_12,

```

```

        .ws    = GPIO_NUM_7,
        .dout  = GPIO_NUM_6,
        .din   = GPIO_NUM_10,
    },
};

    // Initialize standard sender channel only
    ESP_ERROR_CHECK(i2s_channel_init_std_mode(mcu_rx, &std_cfg));
    ESP_ERROR_CHECK(i2s_channel_enable(mcu_rx));
    ESP_ERROR_CHECK(i2s_channel_init_std_mode(mcu_tx, &std_cfg));
    ESP_ERROR_CHECK(i2s_channel_enable(mcu_tx));
} // end of i2s_bt_to_mcu_init

// -----
// -----
// ----- MAIN -----
// -----
// -----

extern "C" void app_main()
{
    i2s_bt_to_mcu_to_dac_init();           // Initialize module connections

    uint8_t *bt_buff = (uint8_t *)calloc(1, BUFFER_BYTES);    // Allocate
space to store data coming from BT module
    assert(bt_buff);

    size_t bytes_read = 0;
    size_t wrote = 0;

    // Read bytes from the Bluetooth Module (MCU acts as Receiver) and
echo/send it to the DAC (MCU acts as Transmitter)
    while (1)
    {
        i2s_channel_read(mcu_rx, bt_buff, BUFFER_BYTES, &bytes_read,
portMAX_DELAY);

        size_t bytes_to_w = bytes_read;
        uint8_t *p = bt_buff;

        //ESP_LOGI(TAG, "Read [%d] to DAC", (unsigned)bytes_read);

        while (bytes_to_w > 0)
        {
            //ESP_LOGI(TAG, "Bytes to write: [%d]",
(unsigned)bytes_to_w);
            esp_err_t r = i2s_channel_write(mcu_tx, p, bytes_to_w,
&wrote, portMAX_DELAY);
            //ESP_LOGI(TAG, "Wrote [%d] to DAC", (unsigned)wrote);

            if (r != ESP_OK){
                ESP_LOGE(TAG, "I2S threw ERROR: %d", r);
            }
        }
    }
}

```

```

        break;
    }

    if (wrote == 0)
    {
        vTaskDelay(pdMS_TO_TICKS(1));
        continue;
    }
    bytes_to_w -= wrote;           // If written -> OK, then
decrease counter                // Increase pointer
    p += wrote;
to buffer
    } // end of inner while loop
    } // end of main while loop

    free(bt_buff);

    return;
} // end of main

```

Appendix E – LLM AI Prompts & Outcomes:

[1a] Prompt: *Flutter build fails with: Timeout waiting to lock build logic queue... Owner PID: ##### and a lock at android\gradle\noVersion\buildLogic.lock. What are Gradle daemons, why does this happen, and what exact Windows steps fix it quickly?"*

[1a] Answer with ChatGPT 5: *What Gradle daemons are:*

Gradle daemons are background Java processes that keep the Android/Gradle build environment "warm" to speed up future builds. If a daemon ends abnormally or is left behind, it may leave a lock file (for example: android.gradle\noVersion\buildLogic.lock) that blocks new builds and triggers the message: "Timeout waiting to lock build logic queue... Owner PID: #####".

Why the error occurs:

- *A previous build ended mid-task (IDE closed, system sleep/hibernate, crash, or reboot).*
- *Two builds ran at the same time from different terminals/IDEs.*
- *A Gradle or Android Gradle Plugin mismatch caused a daemon to hang.*
- *Antivirus, backup, or sync tools briefly locked files inside the gradle folder.*

Quick remediation on Windows:

1. *Stop Gradle daemons for the project: open a terminal in the android subfolder and run "gradlew --status" followed by "gradlew --stop".*
2. *End any leftover Java processes: open Task Manager → Details, and end java.exe or javaw.exe entries labeled as Gradle Daemon (or matching the Owner PID shown in the error).*

3. *Remove the stale lock and clean the project: delete the file `android.gradle\noVersion\buildLogic.lock`, then from the project root run “flutter clean”.*
4. *Rebuild: run “flutter pub get” and then “flutter run”.*

Prevention:

Avoid closing the IDE or terminal while Gradle is building; do not start multiple builds at once; prevent the computer from sleeping during builds; keep Flutter, the Android Gradle Plugin, and the Gradle wrapper at the versions recommended by the Flutter channel. If lock issues recur, it is safe to run “gradlew --stop” before starting a fresh build.

[1b] Prompt: *Please explain what a Gradle is*

[1b] Answer with ChatGPT 5: *Gradle is the build system that Android projects use.*

In plain terms: it’s the tool that assembles your Android app. When you run your Flutter app on Android, Flutter hands work off to Gradle, which:

- *downloads and manages dependencies (libraries, plug-ins)*
- *runs the Android Gradle Plugin to compile Java/Kotlin code*
- *packages resources and assets*
- *produces the APK/AAB you install on the emulator/phone*
- *caches results and keeps a background “daemon” process to make later builds faster*

You’ll see Gradle files like `build.gradle` (or `build.gradle.kts`) in the `android/` folder, and the “Gradle wrapper” (`gradlew`) that ensures everyone builds with the same Gradle version. The “stale Gradle daemon” issue you hit is just that background process getting stuck and holding a lock until you stop it and clean things up.

[2a] Prompt: *What is the IVDD for the TPA3116D2DADR at 50% duty cycle*

[2a] Answer with ChatGPT 5: *Since “IVDD at 50% duty cycle” is not directly given, you may:*

1. *Estimate current draw via output power and efficiency assumptions (since Class-D has > 90% efficiency typical in datasheet). [Texas Instruments+1](#)*
2. *Measure it on your own in your setup: At 50% duty cycle (i.e., approximate 50% power output), measure supply current from the main PVCC rail and any internal logic supply rails.*
3. *Contact the manufacturer (Texas Instruments) or review detailed electrical characteristics full table in the datasheet revision “Rev G” to see if IVDD appears in a more complete table.*

[2b] Prompt: *If the power generator is producing 34.2 to 39.6 volts and our sub-woofer amplifier needs 36 volts do we need a regulator or what other options can we use*

[2b] Answer with ChatGPT 5:

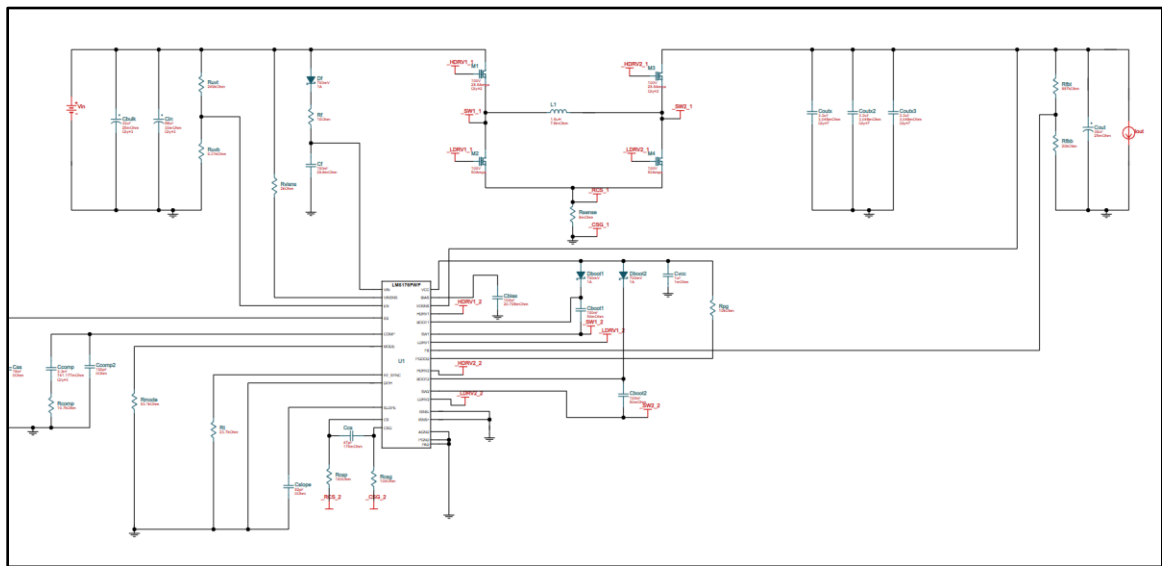
A Graph ChatGPT Laying Out All Suggested Options to My Query

Option	Type / Method	Operation Range	Maintains 36 V Stable?	Efficiency	Pros	Cons	Recommended Use Case
1. Buck-Boost DC-DC Converter (Recommended)	Active switching regulator that steps voltage up or down	30 – 45 V input → 36 V output	Yes (both under and over input conditions)	~90–95%	Works across full input range, efficient, compact modules available (e.g., LTC3780)	Slightly higher cost, adds small switching noise	When voltage fluctuates above & below 36 V; ideal for audio systems needing consistent power
2. Buck (Step-Down) Converter	Switching regulator that only steps voltage down	≥ 36 V input → 36 V output	Partially (only when input > 36 V)	~92–97%	Cheap, efficient, simple	No boost; output drops if generator < 36 V	When voltage rarely dips below 36 V and amp tolerates brief undervoltage
3. Linear Regulator	Series pass transistor maintaining constant output	> 38 V input → 36 V output	Yes (only if input > 36 V)	~60–85%	Simple, low noise	Inefficient, dissipates heat (e.g., 20 W+ at 6 A), large heatsink	Low-power or test setups, not for subwoofer amps
4. Generator Adjustment + LC Filter + TVS	Tune generator output + add filtering and transient suppression	Set output ~36 V $\pm 5\%$	Only if generator stable	~100%	No extra converter, minimal parts	Needs generator with adjustable regulation; no protection from dips	When generator voltage can be finely controlled and stable

Option	Type / Method	Operation Range	Maintains 36 V Stable?	Efficiency	Pros	Cons	Recommended Use Case
5. Combination Approach	Slight generator tuning + buck converter	34–40 V nominal	Mostly	~92%	Reduces converter stress and heat	More setup complexity	When you can slightly tune generator and still want stable regulation

[2c] Prompt: *What do sw1_2, ldrv1_2, csg_1, rcs_1, sw2_2, sw2_1, hdrv2_1, lvdr2_1, hdrv1_1, sw1_1, ldrv1_1, rcs_2, csg_2, mean in this schematic*

(referencing this schematic)



[2c] Answer with ChatGPT 5:

ChatGPT 5's Response

Signal	Typical Meaning	Description
SW1_1 / SW1_2 / SW2_1 / SW2_2	<i>Switch Nodes</i>	The connection between the high-side and low-side MOSFETs for each phase. They carry the pulsating waveform that feeds the inductor. SW1_x → Phase 1 node; SW2_x → Phase 2 node.
HDRV1_1 / HDRV2_1	<i>High-Side Gate Drive Outputs</i>	These are outputs from the driver IC controlling the high-side MOSFET gates for phase 1 and phase 2. They switch between VIN and SWx.
LDRV1_1 / LDRV1_2 / LDRV2_1 / LDRV2_2	<i>Low-Side Gate Drive Outputs</i>	Drive the low-side MOSFETs in each half-bridge (connected to ground).
CSG_1 / CSG_2	<i>Current Sense Ground / Negative Input</i>	These are the negative (return) pins of differential current sense amplifiers or sense resistor connections. Often labeled CSG (Current Sense Ground) in controllers.
RCS_1 / RCS_2	<i>Current Sense Resistor / Positive Input</i>	Correspond to the voltage across current sense resistors for each phase (used to regulate current or detect load). “RCS” might refer to the resistor or to the node that senses its voltage.
LDRV1_2 / LDRV2_1 / etc.	<i>Duplicate Low-Side Drives</i>	These are usually mirrored outputs for multi-FET configurations per phase or parallel FETs. The “_1” and “_2” suffixes often correspond to duplicated components in the same phase (for load sharing).

[3a] Prompt: Based on the following information, can I use the channels in parallel to process one signal?

The two 12-bit buffered DAC channels can be used to convert two digital signals into two analog voltage signal outputs.

This dual digital interface supports the following features:

- two DAC converters: one for each output channel
- 8-bit or 12-bit monotonic output
- left or right data alignment in 12-bit mode

- synchronized update capability
- noise-wave generation
- triangular-wave generation
- dual DAC channel independent or simultaneous conversions
- DMA capability for each channel including DMA underrun error detection
- external triggers for conversion
- input voltage reference VREF+ or internal VREFBUF reference.

The DAC channels are triggered through the timer update outputs that are also connected to different DMA streams.

[3a] Answer with ChatGPT 5:

Short answer: **Yes — you can use both DAC channels in parallel to process one signal**, but *how* you do it matters a lot. You have two useful modes:

1. **Duplicate/drive-strength mode (same sample on both DACs simultaneously)** both DACs output the same value at the same time, then you combine the analog outputs with proper circuitry (summing op-amp or resistor network) if you need more drive or a different voltage swing. Use the DAC's *synchronized update* so both channels change together.
2. **Interleaved/time-interleaving mode (alternate samples to increase effective sample rate)** feed odd samples to DAC1 and even samples to DAC2 and trigger them alternately. Combine the analog outputs with a summing/hold arrangement or low-pass filter to reconstruct a single high-rate waveform. You must trigger channels independently and precisely (timers/DMA per channel) so the timing offset is well controlled.

Below I'll cover practical constraints, a checklist, and a short implementation plan you can follow.

Key constraints & gotchas

- **You cannot simply tie the two DAC outputs together** — that risks contention and incorrect voltages. Always combine outputs through an op-amp summer/buffer or passive resistor network into a buffer op-amp.

- **Settling time / update glitch** — DAC outputs have finite settling time and update glitch energy. For interleaving, the DAC settling time must be significantly shorter than the interleaved sample period, otherwise samples will be corrupted.
- **Timing accuracy** — interleaving requires deterministic, phase-accurate triggers. Use separate timers (or one timer with compare channels) with phase offset, or staggered timer events. The datasheet phrase “dual DAC channel independent or simultaneous conversions” indicates independent triggering is supported.
- **Reference selection** — both channels must use the same VREF (external VREF+ or internal VREFBUF) to avoid scale mismatch.
- **Gain/offset mismatch** — each DAC will have slightly different offset/gain. For precise reconstruction you must calibrate (measure per-channel offset/gain and correct in the digital domain).
- **DMA streams** — you have DMA per channel and DMA underrun detection — use one DMA stream per DAC for interleaving or synchronized DMA transfers for duplication.
- **Filtering / reconstruction** — after analog summation or interleaving, you will need a reconstruction/anti-imaging low-pass filter to remove step images and smooth the waveform.
- **Noise/triangle generators** — those built-in generators are channel-local and not helpful for combining channels into one high-rate signal (unless you have a specific waveform synthesis idea).

Practical checklist (before you try)

1. Read the MCU DAC datasheet section for:
 - DAC settling time and glitch specs.
 - Maximum update rate per channel.
 - Whether simultaneous update forces identical update timing (it does — used for duplicate mode).
2. Decide mode: **duplicate** (same sample simultaneously) or **interleave** (alternate samples).
3. Hardware: design an analog combiner:
 - For duplicate: simple resistor average followed by buffer op-amp to prevent loading.
 - For interleave use a small summing resistor network and a low-pass reconstruction filter + buffer; alternatively use a fast sample-and-hold or op-amp that can be configured as a multiplexer/buffer.
4. Configure timers and DMA:
 - Duplicate: use one DMA feeding both channels or feed each channel the same stream and use synchronized update.
 - Interleave: configure separate DMA streams and timers (or same timer with two events with 50% phase offset). Ensure precise phasing.
5. Calibration: measure ADC/oscilloscope output and correct offset/gain mismatch in the sample feed.

6. Verify on scope: check timing alignment, glitches, and reconstructed waveform quality.

Example implementation plan (interleaving)

- Prepare your sample buffer: split full-rate sequence into two buffers (even samples $\rightarrow$ DAC1, odd samples $\rightarrow$ DAC2).
- Configure DAC channels for 12-bit right-aligned mode (or whichever you prefer). Use same VREF.
- Set up Timer1 $\rightarrow$ trigger DAC1 at $F_s/2$, Timer2 $\rightarrow$ trigger DAC2 at $F_s/2$ but phase-shifted by half the period (or use Timer with two compare channels separated by half period).
- Configure DMA stream A to move buffer1 $\rightarrow$ DAC1 and DMA stream B buffer2 $\rightarrow$ DAC2. Enable DMA underrun detection so you catch starvation.
- On the analog side, sum both DAC outputs through matched resistors into a rail-to-rail op-amp buffer and then a 1-pole (or higher order) reconstruction filter with cutoff $<$ Nyquist of the effective output.
- Calibrate per-channel gain/offset by outputting known values and measuring the combined analog output; apply digital corrections.

When duplicate mode is useful

- When you want the same analog waveform on two destinations (e.g., two analog inputs of different subsystems).
- When you need more drive: feed each output to an op-amp configured as non-inverting buffers and tie their outputs only at a buffered node (not directly).

Short recommended circuit ideas

- **Resistive summer + buffer:** DAC1 $\rightarrow R \rightarrow$ node; DAC2 $\rightarrow R \rightarrow$ same node; node $\rightarrow$ op-amp buffer (unity) $\rightarrow$ reconstruction filter. Use R values sized to avoid heavy loading of DAC outputs (e.g., 10k–100k then buffer).
- **Op-amp summing amplifier:** use precision op-amp configured as summing amp so you can apply per-channel weighting if desired.

Final notes / recommendations

- If your goal is simply higher resolution, don't try to combine two DACs in analog. Instead, increase bit depth in the digital domain or use a single higher-resolution DAC. Combining two DACs to increase resolution is fragile due to mismatch and noise.
- If your goal is higher sample rate, interleaving is feasible *if* timing and settling constraints are satisfied.

- Verify everything on the bench (oscilloscope + load) and plan for per-channel calibration.

If you tell me **which STM32 family / exact MCU** you're using and **what your end goal is** (higher sample rate, more drive, alternate waveform sources, higher resolution, etc.), I'll sketch a concrete timer+DMA configuration and a simple op-amp circuit you can build.

[3b] Prompt: Which one of these is the best considering we do not necessarily need 24-bit resolution, and we want at least 4 channels? Also let me know if any of these are not suitable. DAC8555IPW AD1934WBSTZ-RL AD5664ARMZ-REEL7 DAC8574IPW

[3b] ChatGPT 5 Response:

Chapter 12 Detailed comparison (why each is/isn't suitable for your use case)

Chapter 13 AD1934 (Analog Devices) audio part, 8 channels

- **What it is:** 8 DAC channels in a single chip, made for multichannel audio (pro/AV). Accepts digital audio formats used in audio systems (I²S/TDM/word-clock/MCLK) and has audio-centric features (oversampling, filters, good SNR). [Analog Devices+1](#)
- **Why it fits:** You have an STM32H735ZG with SAI/TDM — AD1934 can be driven directly by that interface, simplifies timing, provides audio-grade digital filters and clocking options, and gives spare channels (you only need 3 now).
- **Tradeoffs:** It's 24-bit by design (fine even if you only use 16-bit data), requires proper MCLK/BCLK/WS clocking and analog output filtering, and is a more audio-centric (slightly higher integration) part than a simple voltage DAC.

Verdict: *Highly suitable* and probably the best fit if you want an audio-optimized, multichannel solution.

Chapter 14 DAC8555IPW (Texas Instruments) 4-ch, 16-bit, SPI, low-glitch voltage outputs

- **What it is:** 4-channel, 16-bit buffered voltage-output DAC with SPI/QSPI interface, very low glitch behavior, rail-to-rail outputs, designed for precision multichannel voltage outputs. [Texas Instruments+1](#)
- **Why it fits:** It satisfies your “≥4 channels” and 16-bit requirement; SPI interface can be fast enough to stream audio if you implement a high-speed SPI routine (and if timing/synchronization constraints are managed). Low glitch is good for audio artifacts.

- **Tradeoffs / cautions:** Not an audio (I²S/TDM) DAC — it lacks audio oversampling filters, PLLs for clock jitter management, and common audio-format conveniences. The MCU must generate channel updates (synchronous multi-channel timing and DMA usage needed). Also check aggregate SPI bandwidth and MCU CPU/DMA load for your sample rate and channels.

Verdict: *Suitable* if you plan MCU-driven PCM via SPI and can handle the timing/DMA and analog reconstruction yourself. Good choice if you want a straightforward, 16-bit voltage DAC.

Chapter 15 AD5664ARMZ (Analog Devices) 4-ch, 16-bit, SPI, external VREF

- **What it is:** AD's nanoDAC family: quad, 16-bit, buffered voltage outputs, SPI interface (requires external reference to set full-scale). [Analog Devices+1](#)
- **Why it fits:** Matches the DAC8555 channel count and resolution. It's low power, tiny package, guaranteed monotonic, and is a very common choice for multichannel voltage output tasks.
- **Tradeoffs / cautions:** Like DAC8555, it's a general-purpose voltage DAC, not an audio I²S/TDM part. You'll need an accurate external reference (unless you accept lower absolute accuracy) and manage update timing from the MCU.

Verdict: *Also suitable* for MCU-driven 16-bit audio if you want AD's nanoDAC features — choose it if you prefer ADI parts and can provide VREF.

Chapter 16 DAC8574IPW (Texas Instruments) — 4-ch, 16-bit, I²C

- **What it is:** Quad 16-bit buffered voltage DAC with I²C interface and on-chip reference option. [Texas Instruments+1](#)
- **Why might it be tempting:** Simple two-wire bus, internal reference, rail-to-rail outputs. Good for control voltages or slow updates.
- **Why it's not great for streaming audio:** I²C bandwidth and protocol overhead make it a poor choice for continuous, synchronous audio samples on four channels at full audio sample rates (e.g., 48 kHz). Timing and channel alignment are cumbersome. Unless you intend very low sample-rate updates (e.g., control/slow signals), avoid I²C DACs for PCM audio streaming.

Verdict: *Not recommended* for real-time multichannel PCM audio streaming. Use only for control or very slow update use-cases.

Chapter 17 Recommendation (concrete)

Given you **don't need 24-bit**, but you **do want ≥ 4 channels** and you'll be using the STM32's ADC for the mic:

1. If you want **audio-format convenience, best audio timing, and lowest integration pain** → choose **AD1934** (8ch audio DAC). It's the cleanest way to stream multichannel audio from STM32 SAI/TDM with minimal MCU timing gymnastics. [Analog Devices+1](#)
 2. If you prefer a **simpler 16-bit voltage DAC (MCU updates channels over SPI)** and want to keep hardware simpler/cheaper, pick **DAC8555IPW** (TI) or **AD5664ARMZ** (ADI). Between those:
 - **DAC8555**: low-glitch, robust, no external VREF required (use internal architecture), good if you prefer TI and want very low transient artifacts. [Texas Instruments+1](#)
 - **AD5664**: small package, low power, requires external VREF (gives you control over FS accuracy). [Analog Devices+1](#)
 3. **Avoid DAC8574** for streaming audio — PC is not appropriate for synchronized PCM audio across multiple channels at typical audio sample rates. Use it only for slow updates. [Texas Instruments](#)
-

Chapter 18 Practical notes if you pick a voltage-DAC (DAC8555 / AD5664)

- You'll have to implement synchronous updates for the 4 channels, so channel sample alignment is correct — use MCU DMA + SPI at a sufficiently high clock rate. Calculate required SPI clock: e.g., for 4 channels $\times$ 48 kHz $\times$ 16 bits $\approx$ 3.072 Mb/s plus framing overhead — so SPI at e.g. 10–20 MHz should be fine.
- Implement an analog reconstruction filter per channel (RC or LC) and follow recommended layout/decoupling to minimize noise.
- Watch settling time specs (DAC856/5664 have different settling times). AD5664 has $\sim 4\ \mu\text{s}$ settling; DAC8555 $\sim 12\ \mu\text{s}$ in some modes — verify if you plan very high sample rates.

[4a] ChatGPT 5 Prompt:

Give me a list of bluetooth modules that my group can use that has an integrated antenna and has one of the A2DP, aptX, AAC protocols that allow us to stream music to the system

Answer:

Great — since your group is working on a music-streaming system and needs a Bluetooth audio module **with an integrated antenna** and support for streaming audio via **A2DP** (and ideally the higher-quality codecs like **aptX** or **AAC**), here's a set of recommended modules **plus** some key selection criteria you should evaluate.

You (Jordan) can pick one that fits your board design (power budget, interface, audio outputs, etc.). Because you have audio electronics experience (subwoofers, amplifiers, etc.), you'll appreciate the nuance in codec and interface support.

✅ What to check / specification checklist

When evaluating a Bluetooth audio module for your streaming system, ensure it meets:

1. **Profiles:** A2DP (Advanced Audio Distribution Profile) for stereo audio streaming. [Wikipedia+2Infineon Community+2](#)
 2. **Codecs:** Support for at least SBC (mandatory for A2DP) but preferably **aptX** and/or **AAC** for better fidelity. [Microchip Technology Incorporated+3Mouser+3Ezurio+3](#)
 3. **Integrated antenna:** Look for modules with built-in PCB chip or “on-board” antenna (eliminating need for external antenna or RF tuning).
 - This simplifies your mechanical/PCB layout and keeps complexity lower for a senior-design-type implementation.
 4. **Audio output interface:** Analog vs digital (I²S, PCM). If you are integrating into your amplifier/subwoofer system, you'll want to match the output type (e.g., differential line out, stereo analog, or I²S to external DAC).
 5. **Certifications / RF compliance:** If this eventually be productized, modular certification helps.
 6. **Control interface:** UART, I²C, or SPI for controlling module (pairing, configuration) — since you're used to microcontrollers, you'll appreciate a module with a clean command set.
 7. **Power / footprint:** Since you've done speaker buildouts, you'll want something that fits your available space, power rails (5 V vs 3.3 V), and fits the audio path cleanly.
-

🔧 Recommended Modules

Here are eight modules that match your criteria (**integrated antenna + A2DP + aptX or AAC support**). I'll highlight key features for each so you can compare.

Dayton Audio KAB-23 Bluetooth 5.0 aptX HD Receiver Board

\$19.98 Parts Express + others

Microchip BM64 Bluetooth Stereo Audio Module

\$14.40 DigiKey

TS3008 Bluetooth 5.0 aptX/AAC Audio Module

\$8.75 OzRobotics.com

FSC-BT1026D aptX/AAC Bluetooth Audio Receiver Module

\$7.50 OzRobotics.com

TS64215 Bluetooth 4.2 aptX Audio Module

\$7.65 OzRobotics.com

CSR8670 Bluetooth V5.0 Audio Module AAC/aptX

\$20.49 Walmart - Seller

Bluetooth 5.0 aptX Audio Module TS8675

\$18.15 OzRobotics.com

Bluetooth 5.1 aptX Audio Module TS3031

\$8.95 OzRobotics.com

Here's a summary of each:

- Dayton Audio KAB-23 Bluetooth 5.0 aptX HD Receiver Board: A receiver board rather than just a module — integrates Bluetooth 5.0 + aptX HD + built-in antenna. Good if you want a board you can drop into your system quicker.
- Microchip BM64 Bluetooth Stereo Audio Module: A highly capable module supports A2DP, AAC, and has integrated antenna as noted in the datasheet (“stand-alone module with onboard antenna and BLUETOOTH stack”). [Mouser](#)
- TS3008 Bluetooth 5.0 aptX/AAC Audio Module: Based on Qualcomm QCC3008, supports A2DP v1.3.1, aptX, AAC, and has PCB antenna. [TINYSINE AUDIO](#)
- FSC-BT1026D aptX/AAC Bluetooth Audio Receiver Module: A dual-mode (Bluetooth classic + BLE) audio module supporting A2DP, aptX, AAC. Good codec support for high fidelity streaming. [Oz Robotics](#)

- TS64215 Bluetooth 4.2 aptX Audio Module: A more cost-sensitive option: Bluetooth 4.2 but still supports aptX; useful if budget is constrained and you accept slightly older spec.
- CSR8670 Bluetooth V5.0 Audio Module AAC/aptX: Good mid/high-end SOC audio module; strong codec support including AAC & aptX; good for your audio-electronics domain.
- Bluetooth 5.0 aptX Audio Module TS8675: Another module emphasising Bluetooth 5.0 with aptX support—nice if you want more futureproofing.
- Bluetooth 5.1 aptX Audio Module TS3031: Latest spec (5.1) with aptX support; if you're designing something to be state-of-the-art, this is strong.

Recommendation & Selection Tips for Your Project

Since you (Jordan) already have hands-on experience with audio electronics (subwoofers, amplifier classes A/AB, etc.), you'll want to match the module to your system design. Here's how I'd pick:

- If you want **fast integration** with less RF layout risk: pick a module with onboard antenna and already certified (e.g., BM64 or TS3008).
- If you aim for **high audio fidelity** (you mentioned audio electronics, THD, etc.): favour the modules supporting aptX (or aptX HD) and AAC.
- If you're **budget-conscious** (student project / senior design): maybe pick the older Bluetooth 4.2 aptX option (TS64215) but accept trade-offs in range or futureproofing.
- Ensure your **audio output** aligns: Does the module give analog stereo output (line-out) or digital (I²S)? If you go to your amplifier, choose accordingly.
- Also check how the module interfaces with your microcontroller or host (UART commands, ready-made breakout, etc.).
- Note the **antenna placement** and PCB footprint: integrated antenna is easier, but you'll still need to consider clearance around the antenna for good RF performance (since you're building the system housing).
- Power rail: some modules expect 3.3 V, some 5 V; match to your system's power rails and ensure your power supply can handle audio plus module draw.

If you like, I can **pull together full datasheet links, pinouts, breakout-board options**, and even **filter by footprint/price** (for your budget and PCB size constraints) for your team. Would you like me to dig those?

[4b] ChatGPT 5 Prompt:

Can you place an emphasis on parts available on DigiKey

Answer:

Great — focusing on modules available via DigiKey (which you and your team can order easily) that **meet your audio-streaming criteria** (integrated or on-board antenna viable, A2DP support, optional aptX/AAC) here are **top picks** along with commentary tailored for your senior-design audio system. I'll highlight their key specs (that matter for your system design) and implications for your amplifier/subwoofer build.

 Recommended DigiKey-available modules

IOT747 IDC777-1 Bluetooth 5.3 Audio Module

\$31.27 DigiKey

IOT747 IDC767-1 Bluetooth 5.3 Audio Module

\$31.27 DigiKey

Feasycom FSC-BT1026D Bluetooth 5.1 Audio Module

\$14.46 DigiKey

Feasycom FSC-BT1026D (alt listing)

\$14.46 DigiKey

Seeed Bluetooth Module (basic)

\$12.90 DigiKey + others

Murata LBEE5ZZ2XS-846 Bluetooth RF TXRX Module

\$31.78 DigiKey

Silicon Labs SLWSTK6101D Bluetooth Module

\$186.50 DigiKey

FDK HY0020 Bluetooth LE V5.4 Module

\$14.04 DigiKey

Here are brief summaries:

- IOT747 IDC777-1 Bluetooth 5.3 Audio Module: High-end audio module supporting aptX Lossless, aptX HD, AAC, plus A2DP Sink/Source. Certified stack. Excellent if your system aims for near “audiophile” wireless streaming. [DigiKey+1](#)
 - IOT747 IDC767-1 Bluetooth 5.3 Audio Module: Similar family to the above; audio-optimized, aptX, AAC support; gives you high performance. [IOT747](#)
 - Feasycom FSC-BT1026D Bluetooth 5.1 Audio Module: More mid-tier but still meets aptX & AAC and A2DP profile; good balance of cost and features for your project. [FCC Report](#)
 - Feasycom FSC-BT1026D (alt listing): Same part basically; inclusion just underscores availability via DigiKey.
 - Seeed Bluetooth Module (basic): More budget-oriented; you’d need to check codec support (aptX/AAC) carefully, likely not full high-end.
 - Murata LBEE5ZZ2XS-846 Bluetooth RF TXRX Module: Great RF module but doesn’t emphasise high-quality audio codecs; only choose if you plan to add your own audio path and accept default SBC/A2DP.
 - Silicon Labs SLWSTK6101D Bluetooth Module: Evaluation/starter kit category; might be overkill for simple streaming module.
 - FDK HY0020 Bluetooth LE V5.4 Module: Bluetooth LE focus — likely doesn’t support aptX/AAC for A2DP stereo streaming (LE Audio is evolving, but for your current stereo system you probably want Classic A2DP + aptX/AAC).
-

My Recommendations for Your Team

Given your system (you’ve built subwoofers, amps, you’re designing an audio streaming front-end) here are what I’d emphasize:

Pick modules with aptX and/or AAC support

Since you care about fidelity beyond just “wireless sound”, aim for modules that explicitly list: aptX (or aptX HD/Lossless) **or** AAC codec support. For example, the Feasycom FSC-BT1026D lists aptX, aptX HD, aptX Low Latency, and AAC. [FCC Report+1](#)

This ensures you’re not stuck with the basic SBC codec (which all A2DP devices support, per the spec). [Wikipedia+1](#)

Since you want to stream music to your system, this module becomes the “front end” between phone/tablet and your amplifier/subwoofer.

Integrated antenna & RF layout considerations

For RF performance, integrated (on-board) antennas simplify the PCB & enclosure design. The IDC777/IDC767 mention “Integrated Antenna, Small form factor” in their datasheet. [Linkwave - Driving IoT Innovation](#)

The Feasycom part has single-ended antenna connection or built-in balun. You’ll need to check footprint/CAD and make sure your system’s metal enclosure doesn’t block the antenna. The datasheet shows the antenna block. [FCC Report](#)

Since you already do amplifier buildouts, you’ll appreciate paying attention to RF clearance, ground plane under antenna, etc.

Audio interface to your amp/subwoofer

Check how the module outputs audio: analog (line out) vs digital (I²S/PCM). For your system: if you have a DAC or amplifier already, you might prefer analog line-out stereo. If you have a high-end DAC and want full flexibility, digital interface is better.

For instance, IDC777 supports both analog & digital interfaces (I²S, PCM). [Linkwave - Driving IoT Innovation](#)

Ensure the module’s audio output can drive your amplifier input (gain, signal level, noise floor) and matches your routing practices (balanced/unbalanced, etc.).

Budget vs discipline

Since this is a senior-design project, you’ll want to balance cost vs performance.

- If you pick the high-end IDC777/IDC767 modules, you may pay a higher BOM cost (say ~\$30 each). But you get “future-proof” capability.
- If budget is tighter, the FSC-BT1026D gives a strong trade-off and still supports aptX/AAC.
- If cost is really limited, you can pick a more basic module (Seeed’s or Murata’s) but you may be limited to SBC or mono audio or simpler interface.

Certification and production readiness

Because these modules are pre-certified (Bluetooth/FCC/CE) you reduce the burden of RF certification in your project, which is helpful. For example, the IDC777 datasheet lists FCC, RED, MIC, KCC certifications. [Linkwave - Driving IoT Innovation](#)

Since your school project may not require full product cert, but good to note if you want to scale to a prototype production.

Interface/Control-wise

Make sure the module has a convenient host interface (UART or I²C) for configuration (pairing, controlling audio path, volume, status). The IDC777 uses “AudioAgent”

firmware with UART commands. [MikroElektronika Downloads](#)

You (Jordan) being comfortable with microcontrollers should appreciate having a clean command set to integrate into your control logic (e.g., switching sources, tracking connection status, volume control, LED indicator).

✓ **My Top Pick for *Your* Project**

Given your team is building a music-streaming system for sub-woofer/amp, and you have good audio electronics background, here's what I'd pick:

- **Primary candidate:** Feasycom FSC-BT1026D — gives aptX/AAC, A2DP, dual mode, good price, good features.
 - **If you want premium:** IOT747 IDC777-1 — for “audiophile-wireless streaming” scenario, especially if you use high-end amp/sub and want high fidelity.
 - **Ensure audio path:** Choose whichever supports analog line out stereo or digital interface your existing amp/DAC expects.
 - **Budget fallback:** Choose one of the simpler modules with A2DP only (SBC) if audio fidelity is less critical.
-

If you like, I can **pull together a full comparison table** of ~5 modules from DigiKey (with part numbers, cost, footprint, antenna type, codec support, audio interface) and highlight **which PCB/footprint to pick** for your team's layout (so you can drop into your PCB and not redo mechanical). Do you want me to generate that table?