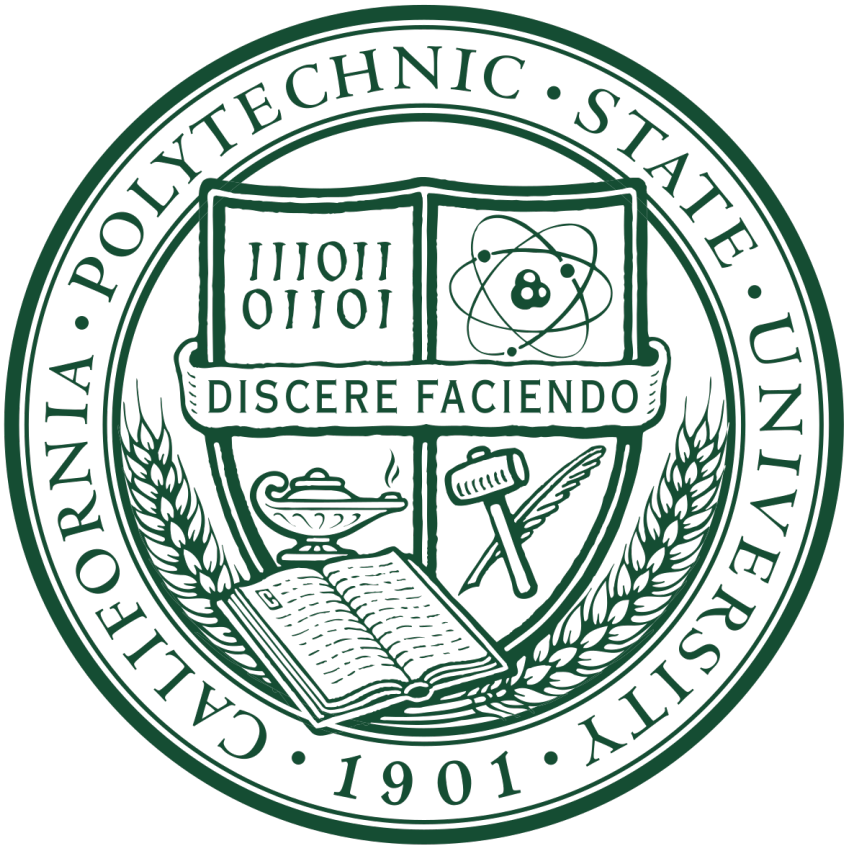
CALIFORNIA POLYTECHNIC STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING



FxLMS-Based ANC in Custom 3D-Printed Headphones

Seth Cherry and Andrew Chookaszian
Dr. Wayne Pilkington

June 5, 2025

Contents

List of Figures

List of Tables

Listings

FxLMS-Based ANC in Custom 3D-Printed Headphones

1st Andrew Chookaszian
Cal Poly San Luis Obispo -)
Electrical Engineering Department
San Luis Obispo, United States
andrew@chookaszian.com
Advisor: Dr. Wayne Pilkington

2nd Seth Cherry
Cal Poly San Luis Obispo -
Electrical Engineering Department
City, Country
seth.d.cherry@gmail.com
Advisor: Dr. Wayne Pilkington

***Abstract*—Abstract – This project presents a pair of mid-range active noise-cancelling (ANC) headphones designed for affordability, performance, and user repairability. The headphones feature two Bluetooth modes (pass-through and quiet), 10 dB active attenuation between 300–500 Hz, 20 dB passive attenuation, and support for 3.5 mm wired audio. Weighing approximately 350g, the device is highly portable and constructed with a modular design that includes replaceable earmuffs, an adjustable headband, and open-source 3D-printable parts for easy customization and repair. The system was successfully built and tested, demonstrating functional ANC capabilities. However, the measured attenuation was below expectations due to mechanical limitations, offering valuable insights for future refinement in structural design.**

***Index Terms*—ANC - active noise cancelling, STL - standard tessellation language, PCB - Printed Circuit Board**

Fig. 1. PloopyCo makes open source 3D printed headphones [1]

I Introduction

This project aims to develop a set of active noise cancelling (ANC) headphones that are affordable, modular, and repairable—serving two key customer groups. The first group includes hobbyists and tech enthusiasts who value a customizable and upgradeable platform. The second group consists of cost-conscious users, particularly those in lower-income brackets, who may be exposed to high levels of daily noise pollution but are priced out of the current ANC headphone market. For this group, an affordable and durable alternative could provide meaningful health and lifestyle benefits.

The product is motivated by the right-to-repair movement, which advocates for greater user control over the maintenance, modification, and longevity of their products. Most commercial ANC headphones, especially high-end models, are difficult or impossible to repair due to proprietary designs and sealed construction. In contrast, this design prioritizes repairability and longevity, offering customers an alternative to disposable electronics.

To meet these customer needs, the product is constrained to a maximum price of $250. This cap ensures accessibility for a broader user base, especially when compared to premium models that exceed $400. Despite this price limitation, the headphones will feature core ANC functionality, Bluetooth connectivity, and passive attenuation comparable to mid-range commercial products. The physical structure of the headphones is entirely 3D-printed, enabling customers to produce replacement parts or custom modifications independently. By releasing the STL files as open-source, users are empowered to maintain and extend the product life-cycle themselves.

The headphones are primarily intended for personal use in public transit, office settings, or other urban environments where noise exposure is common. Although standalone, the system could be incorporated into broader personal audio setups or used as a testbed for DIY electronics communities and embedded DSP developers. Compared to commercial alternatives, this product is unique in its openness, repairability, and cost-efficiency. While premium ANC headphones offer strong performance, they are typically expensive, non-serviceable, and not modifiable. This project offers a viable tradeoff by targeting the most common noise frequencies (300–500 Hz), delivering modularity and accessibility while still achieving meaningful noise reduction.

II Background

When undertaking a project like this, the decision of how to implement active noise cancellation (ANC) was paramount. In order to begin, we needed to determine how to effectively cancel ambient noise in the environment surrounding the head-

phones. While the underlying theory is straightforward—since sound waves can interfere constructively or destructively—we aimed to generate a 180° phase-shifted signal (or "anti-noise") of the recorded noise. When played through the speaker, this anti-noise ideally cancels out the ambient sound, resulting in a muted acoustic environment.

However, implementing this with real-world efficacy requires the use of adaptive filtering techniques, as the noise measured outside the headphones is altered as it propagates through the headphone structure to the user's ears, introducing acoustic delay. Adaptive filtering operates by adjusting filter coefficients in real time based on an observation vector and an error signal. This process drives the system along a performance surface toward a minimum error value. The rate of convergence is controlled by the algorithm's step size, which governs how aggressively the system adapts.

The approach described here is known as the Least Mean Squares (LMS) algorithm, which is one of the simplest and most adaptable algorithms to begin with. While LMS can be effective in simulations, hardware implementations suffer from real-world challenges such as conversion losses and propagation delays through materials. To address this, we implemented the Filtered-x LMS (FxLMS) algorithm, which improves convergence by accounting for the secondary path—the transfer function between the speaker and the error microphone. This adjustment allows FxLMS to better minimize the error signal in physical systems [2], [3].

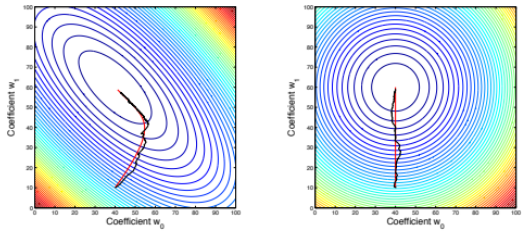


Fig. 2. Visualization of LMS algorithm paths down performance contours with different correlation values [2]

Additionally, for the FxLMS algorithm to function effectively, the passive noise isolation of the headphone enclosure must ensure that the two microphones (reference and error) are measuring separate acoustic environments. If speaker output leaks into the outer microphone, it corrupts the reference signal, causing the algorithm to diverge—or worse, generate constructive interference that amplifies the noise.

This constraint meant that the mechanical design of the headphone housing played a critical role. Since there is limited research into ANC systems using 3D-printed enclosures, especially in consumer-style headphones, we opted for conservative print strategies. We maintained higher infill percentages in the slicer to maximize acoustic isolation. Infill percentage defines how solid a printed object is; for example, a 40% infill results in an object that is 60% hollow. For our design, most components were printed with 70% infill, with a few larger parts printed at slightly lower densities to save material without compromising performance.

III Product Design Engineering Requirements

In order to effectively tackle the design and verification of the end product, Table I was created to quantify the performance of the final product. Within these specifications, certain critical system parameters were identified as being paramount to ensuring system success.

The following are the critical parameters: the FxLMS algorithm estimation time must be fast enough that users are able to use the headphones quickly and painlessly. The headphone cup 3D prints cannot fray or delaminate. A/D converters should have minimal quantization error. The microphone must have an accurate frequency response within the desired range, and the speakers should have an accurate frequency response for good sound quality.

Figure 3 displays the overall inputs and outputs of the system. The inputs include power from a USB cable for charging the internal battery, an input audio signal through AUX, a mechanical mode selection using buttons, and the ambient environment noise. The outputs are a Bluetooth signal to the phone with status information, an anti-noise signal, the desired music signal, and a small amount of leakage noise.

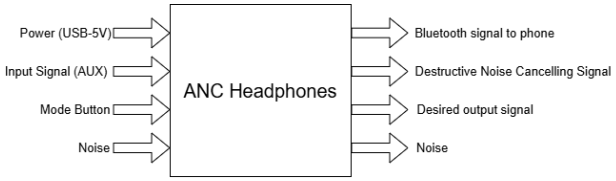


Fig. 3. Level 0 block diagram

The user is able to control the headphones using 3 buttons and a switch. The power switch turns the headphones on and off, the volume up and down buttons control the volume of the music, and the mode button chooses if the headphones are in passive or active noise cancelling mode. Lastly, there is an LED that turns on when the headphones are in noise cancelling mode and off when in passive mode.

If the algorithm takes too long to converge, there would be little incentive to use the headphones for anything beyond novelty. On the other hand, if it tries to converge too quickly, the risk of algorithm divergence increases drastically, potentially causing constructive interference.

If the 3D prints delaminate (i.e., layers of the print come unstuck) or fray too easily, the main justification for using 3D-printed parts—ease of replacement—breaks down, because a part that fails quickly is not meaningfully serviceable.

The rest of the performance requirements are straightforward quantifications used to ensure our system remains compliant with current industry-standard audio practices. The user interfaces on the system are similar to those used in the industry: volume up and down buttons, a mode switch, and a power button. There is also an LED that indicates whether the headphones are currently in noise cancellation mode.

TABLE I
NOISE CANCELLING HEADPHONE DESIGN SPECIFICATIONS

Spec #	Parameter Description	Requirement or Target (Units)	Tolerance	Risk	Compliance	Test Equipment Needed	Source
1	Estimation Time	22.5 (μs)	Max.	H	A, T	Matlab, Oscilloscope	[4]
2	Passive Cancellation Range	20 dB above 501 Hz	Min.	L	T, D	Oscilloscope	[5]
3	Active Cancellation Range	10 dB within 300–500 Hz	Min.	H	A, T	Oscilloscope	[5]–[8]
4	Price	$250	Max.	M	A, S	Calculator	[9]–[11]
5	Flat Frequency Response	20 Hz – 20 kHz	Min./Max.	M	T	Oscilloscope	[12]
6	Repairable	Replaceable Battery/Earmuffs	P/F	L	I, D	N/A	—
7	Comfortable	7 cm tall and 4.5 cm wide	±5%	L	A, I, T, D	Tape Measure	—
8	Adjustable Headband	12 in to 15 in	±5%	H	A, I, T, D, S	Tape Measure	—
9	Weight	350 g	Max.	M	T, D, S	Scale	[9]–[11]
10	Connectivity	3.5 mm Jack (Aux)	P/F	L	A, I, T, D	N/A	[9]–[11]
11	Bluetooth Range	10 m	Min.	M	T, D	Range Finder	[9]–[11]
12	Battery Life	1 hour	Min.	H	T, D, A	Timer	[9]–[11]
13	Modes	3 (Passive, Active)	P/F	M	A, I, T, D	N/A	[9]–[11]

IV System Design - Functional Decomposition (Level 1)

IV-A Overall System Functional Block Diagram

Figure 24 shows the level 1 functional decomposition. The Blue signals show power signals and Green signals show information. All I2S signals are serial communication signals that are 16 bits at 44100 Hz. For the full sized image check Other Documentation within the appendix.

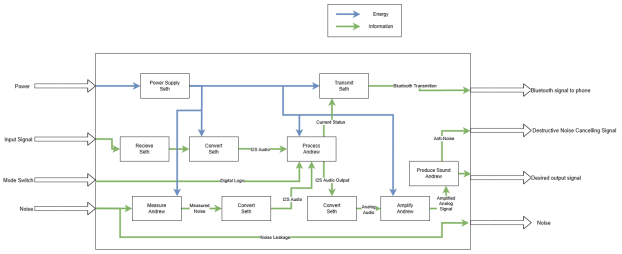


Fig. 4. Level 1 block diagram

IV-B Implementation Concept Block Diagram

Figure 25 shows all major physical components of the system. Within each component, the main functions are broken up with signal flow shown. This is useful for understanding how mechanical, information, and energy signals flow throughout the system and its parts. Each colored box shows a different component with its individual functions broken up. For the full sized image check Other Documentation within the appendix.

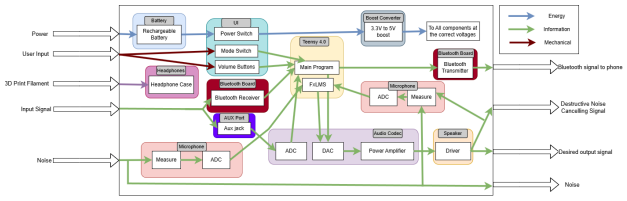


Fig. 5. Implementation Concept Block Diagram

IV-C System Design Specification Partitioning or Performance Budgets

Table II displays each system requirement along with the subsystems that each depends on. This is important to help differentiate which systems are working or not when certain requirements are not met.

V Subsystem Descriptions

V-A Audio Codec

In any personal audio system, both digital-to-analog (DAC) and analog-to-digital (ADC) conversion are typically required. For this design, a DAC was needed to drive speaker output and an ADC to capture microphone input. In addition to data conversion, proper signal conditioning was necessary—specifically, anti-aliasing filters, microphone preamplification, and speaker amplification.

Initial attempts involved discrete components, including LTSpice simulations of anti-aliasing filters using Sallen-Key low-pass architectures. However, to reduce complexity and ensure tighter integration, we transitioned to a codec-based solution. Our first candidate was the TLV320AIC3104IRHBR from Texas Instruments, but due to PCB complexity and development time risk, we prioritized breakout boards.

TABLE II
SYSTEM REQUIREMENTS AND SUBSYSTEM DEPENDENCIES

Spec #	System Requirement	Subsystem / Design Dependency
1	Estimation time should be $\leq$ 22.5 μs.	Dependent on the processing speed of the Teensy 4.0 MCU and its ability to produce output signals in real time.
2	Passive noise cancelling should attenuate signals $>$ 500 Hz by at least 20 dB.	Based on the acoustic characteristics of the 3D printed headphone case and foam lining.
3	Active noise cancelling should attenuate signals between 300–500 Hz by at least 10 dB.	Dependent on the accuracy of the Teensy MCU and the FxLMS algorithm for noise estimation; also depends on the driver's ability to reproduce anti-noise without distortion.
4	Total cost should not exceed $250.	Must ensure selected components are budget-conscious yet maintain sufficient quality to meet performance targets.
5	Frequency response should be flat from 20 Hz to 20 kHz.	Dependent on performance of Bluetooth link, DAC, power amplifier, and speaker driver chain.
6	Repairable earmuffs and replaceable battery.	Design must allow easy removal of earmuffs and include accessible battery housing.
7	Earmuff dimensions: $\geq$ 7 cm tall, 4.5 cm wide.	Selection of correctly sized ear cushions is required.
8	Headband adjustment range: 12–15 cm.	Depends on sourcing a compatible adjustable headband assembly.
9	Total weight $\leq$ 350 g.	Requires choice of lightweight materials, especially for soundproofing elements.
10	Include a 3.5 mm Aux input jack.	Requires integration of a standard aux jack and supporting audio circuitry.
11	Bluetooth connectivity range of at least 10 m.	Select Bluetooth 4.0 or 5.0 modules with proven long-range capability.
12	Battery life of at least 1 hour in ANC mode.	All components must be power-efficient; battery must have sufficient capacity.
13	Support for 2 modes: passive and active.	Use of a mode switch to control operation between the ANC states.

After evaluating several options, we selected the SparkFun Audio Codec Breakout – WM8960 (Qwiic). This device integrates a 16-bit DAC, a 15-bit ADC, and two Class D speaker amplifiers, making it suitable for our application. It communicates via I2S and interfaces with the Teensy microcontroller for both incoming (microphone to Teensy) and outgoing (Teensy to speaker drivers) audio paths. It also sits between the Bluetooth or AUX input and the Teensy for incoming media playback.

Functionality was verified through real-time playback tests. The system was configured for a 44.1 kHz sample rate, and synchronization was confirmed by ensuring glitch-free audio output—any desynchronization would have immediately produced audible artifacts. Thus, the codec was validated as meeting both performance and integration requirements.

V-B Microphones

The main purpose of the microphones is to measure noise in both the ambient environment and the residual error inside the headphones. The Microphones need to have a relatively flat frequency response so they can accurately measure the noise. This means that within the noise-cancelling range of 300 - 500Hz, the frequency response must not deviate more than +-3 dB from a flat response. The microphones need to interface with the microcontroller so the data they collect can be used by the FxLMS algorithm. To easily connect them, digital microphones were chosen so there were no extra ADC's needed, instead they convert the signal to a digital I2S signal internally. Four microphones were required for the final implementation with two on each side, and because the microcontroller chosen only had two I2S ports, Time Division Multiplexing(TDM) needed to be used so all the data could be collected. TDM connects all datalines together and each microphone writes data at a different time in 24 bit slots. Using TDM, all four microphones could be connected to one I2S port, leaving the other port for the audio input and output. The ICS-52000 digital MEMS microphone fit all requirements, but a PCB needed to be made to allow for easy connectivity to the microcontroller.

V-C Custom PCB - Microphone Hub, Mux, and Aux Port

During the development of the ANC headphone system, the design transitioned from analog to digital MEMS microphones to simplify signal routing and eliminate the need for external ADCs. This introduced a new challenge: integrating six I2S digital microphones using only the two I2S ports available on the Teensy 4.0 microcontroller. To overcome this limitation, Time Division Multiplexing (TDM) was selected, allowing multiple microphones to share a single I2S data line by assigning each a unique 24-bit time slot within a shared clock and word select frame.

To implement this solution, a custom PCB was designed (Figure 7) to route and organize the signals for two microphone arrays—one for each headphone cup—into a TDM-compatible configuration. This board also incorporates flat-flex (FPC) connectors for attaching the microphone PCBs, simplifying the physical integration within the headphone enclosure. While the PCB does not perform any active signal conditioning, it plays a critical role in consolidating digital audio and control lines.

In addition to microphone routing, the PCB includes a 3.5 mm auxiliary input jack and an analog multiplexer (mux)

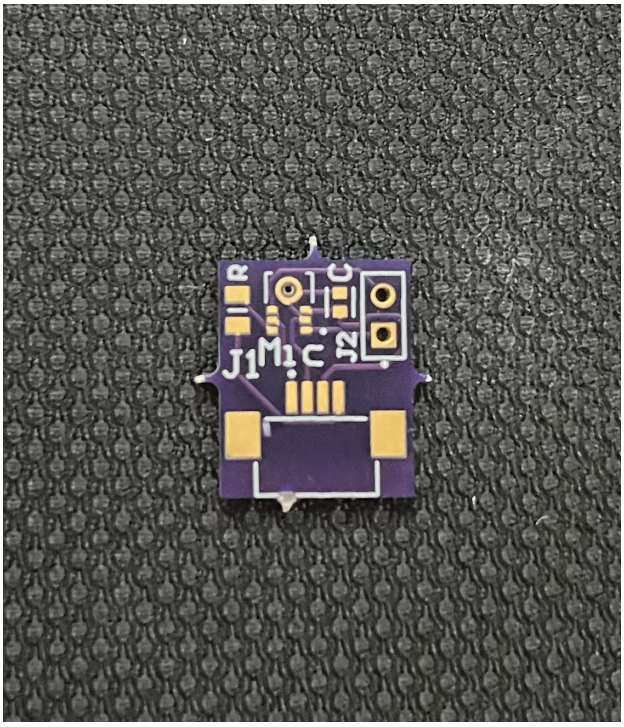


Fig. 6. Microphone PCB

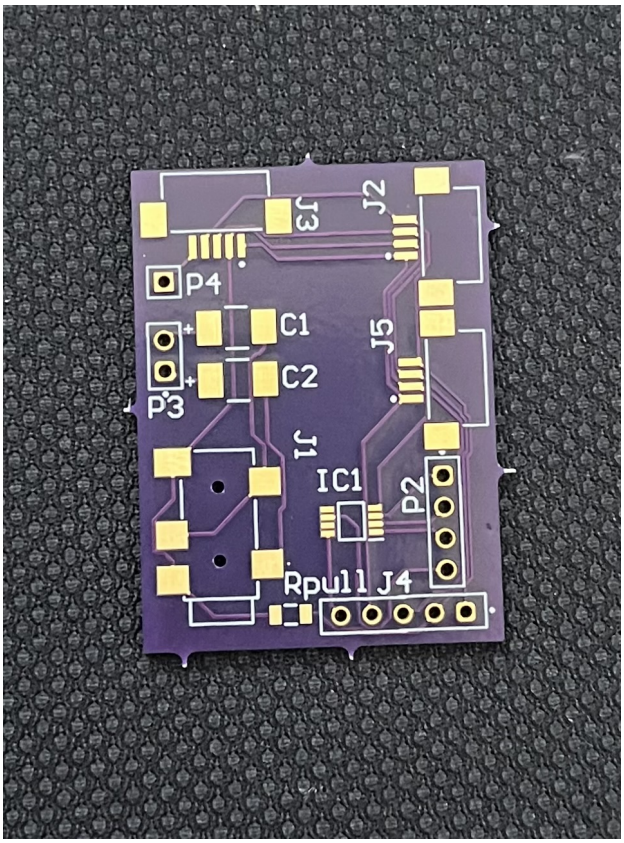


Fig. 7. Connector, Mux, and Aux PCB

IC. The mux allows software-selectable switching between Bluetooth and AUX audio sources, feeding the chosen signal to the system's audio codec. This enables user flexibility without manual reconnections.

The auxiliary input was verified through playback testing, confirming consistent and uncorrupted audio transmission through the codec. Although the microphone routing section is passive, continuity checks were performed across all signal lines to ensure reliable connections. Full system testing later confirmed proper synchronization and data capture via the I2S-TDM configuration, verifying the effectiveness of the PCB's signal layout.

Additional circuit schematics and layout information for this board are provided in Appendix X-C.

V-D Microcontroller

The microcontroller is the part of the system which controls all signal flow and runs the FxLMS algorithm. It is used as a hub where the microphone data and audio data will meet up, get processed, and then get output to the speakers. The main requirements of this component were that it needed to have I2S inputs, as this was the chosen method of communication between the peripherals and controller. It also needed to have a DSP so that the FxLMS algorithm could be efficiently computed. Lastly, it needed to be small enough to fit on the headphones, our requirement was less than 5 x 5 cm. We ended up pickign the Teensy 4.0 Microcontroller as it has two I2S ports, a DSP at 600MHz, and is only 3.6 x 1.8 cm. Other MCUs were considered like the ESP32 which did not have a DSP, or STM32F4 which did not have a fast enough clock.

V-E Bluetooth Receiver

Bluetooth (BT) functionality was a critical requirement to ensure the system's competitiveness with mainstream consumer ANC headphones from manufacturers such as Bose and Sony. Given the widespread use of wireless audio in modern applications, support for BT audio streaming was essential.

The original plan was to develop a custom Bluetooth receiver PCB to integrate with the custom audio codec solution. However, after shifting the design to favor breakout board components for faster development and reduced risk, a prebuilt solution was selected: the AudioB I2S Bluetooth Digital Audio Receiver Module – I2S Controller from TinySine. This module outputs digital audio over I2S and can be configured as an I2S peripheral via SPI commands, enabling seamless integration with the Teensy microcontroller and audio codec.

Unfortunately, due to external geopolitical and logistical disruptions—including international tariff-related delays—the module shipment was significantly delayed and ultimately not received in time for integration. As a result, Bluetooth functionality could not be implemented in the final prototype.

Despite this, the system architecture—including the custom multiplexer circuit and I2S routing—was designed with Bluetooth compatibility in mind. This allows for future integration with minimal redesign, preserving system scalability and alignment with original product requirements.

V-F Physical Headphone Components

To support a modular, repairable, and customizable design, the headphone housing was fabricated using fused deposition modeling (FDM) 3D printing. The development process began with a review of open-source mechanical designs, leading to the selection of the PloopyCo headphone model (Figure 1) for its generously sized earcups and openly licensed STL files.

The base models were modified to better align with system requirements. Specifically, cosmetic indentations on the outer earcups were removed to create flat surfaces for mounting microphones, FPC connectors, and other hardware. These adjustments improved mechanical integration while preserving structural strength.

Following full assembly—including embedded electronics, wiring, mounts, and padding—the final headset weighed approximately 500g, exceeding the original design target of 350g. The primary contributor was the 70% infill density used for the earcups, selected to enhance passive acoustic isolation and rigidity. A subsequent print at 40% infill reduced weight but compromised isolation, demonstrating a clear tradeoff between acoustic performance and wearability.

Despite the excess mass, the printed housing met functional requirements by supporting component installation, protecting internal wiring, and maintaining sufficient acoustic separation between the reference and error microphones. All STL files, including modified versions, are provided in onn the PloopyCo webpage [1] for reproducibility and further development.

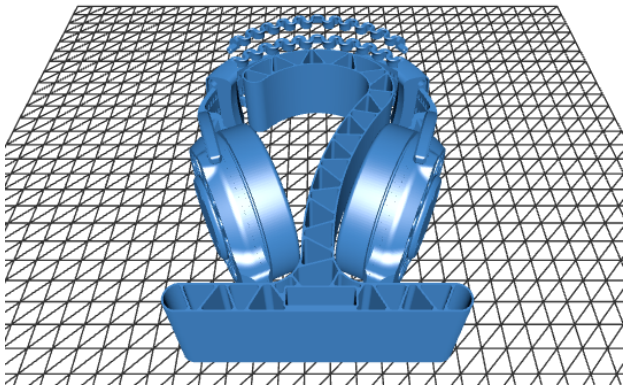


Fig. 8. Headphone Full Mechanical STL

V-G Power Components

The power subsystem consisted of three primary components: a 3.7V, 2000mAh lithium-polymer (LiPo) battery, a USB-C charging interface, and a boost converter. A rechargeable battery was selected to improve usability and align with consumer expectations, as nearly all commercial ANC headphones use integrated, rechargeable power sources. Removable cells were ruled out early in the design phase due to the inconvenience of manual replacement.

The boost converter used in the system accepts an input voltage range of 0.9–5V and provides a regulated 5V output, which powers the Teensy microcontroller, audio codec, speaker amplifiers, and other subsystems. The USB-C port was included to support convenient charging via standard 5V USB power supplies.

During full-system testing—with the ANC algorithm actively processing and music playback enabled—the system achieved a continuous runtime of approximately 2 hours. Although limited compared to commercial offerings, this exceeded the original design goal of 1 hour, demonstrating effective power budgeting and subsystem efficiency under prototype constraints.

V-H Speaker

The speakers main function is to play the music as well as the training noise and anti-noise signals. The requirements for this component included a flat frequency response, especially in the range of noise cancellation; This means it shouldn't deviate more than +- 3dB from flat in the range of 300 - 500 Hz; It also needed to be 16 or 32 ohms, as this is the standard for headphone drivers and the impedance that most headphone amplifiers can power; Laslty, it needed to have a 40mm diameter because the headphones we built had a 40 mm hole. The chosen driver was from Dayton audio, specifically the CE Series CE38MB-32.

V-I FxLMS Algorithm

The filtered-x least mean square algorithm was the chosen adaptive filter to perform active noise canceling. This algorithm requires an estimated secondary path, the path from the output of the adaptive filter to the error microphone including DAC, Amplifier, Speaker, air, and microphone. The estimated path is used as a pre-filter for the ambient noise. The design of the algorithm was based off the implementation as shown in the Simulink Real-Time example provided by MathWorks [13]. The Block diagram of the FxLMS simulation is shown in Figure 9. In Figure 10, the convergence of the algorithm is shown. The yellow signal is the ambient noise signal after it passes through the primary path to the error mic and the green signal is that same error signal summed with the anti-noise signal from the output of the FxLMS adaptive filter. The simulation was run for 10 seconds with a 16 tap filter. The algorithm quickly converged after about 3 seconds and got down to a peak amplitude of about 0.1 whereas the noise has a peak around 2.

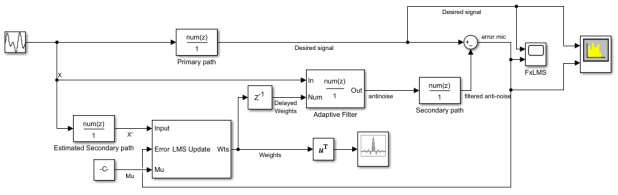


Fig. 9. Simulink FxLMS Simulation Block Diagram

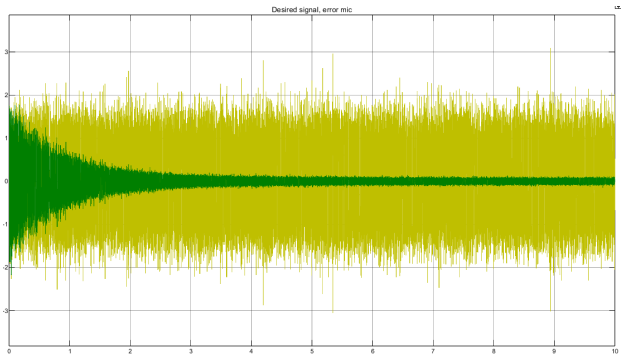


Fig. 10. Simulink FxLMS Simulation Convergence

In Figure 11, the block diagram of the secondary path estimation algorithm is shown. This algorithm uses white noise to train the LMS adaptive filter to match the secondary path filter coefficients. Figure 12 shows this algorithm converging by cancelling out the training noise signal. Figure 13 shows the coefficients that the algorithm converged to when [0.5 0.5 -.3 -.3 -.2 -.2] was used as the secondary path. The output coefficients perfectly converged to the actual coefficients.

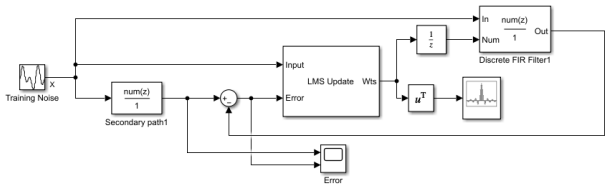


Fig. 11. Simulink Secondary Path Estimation Simulation Block Diagram

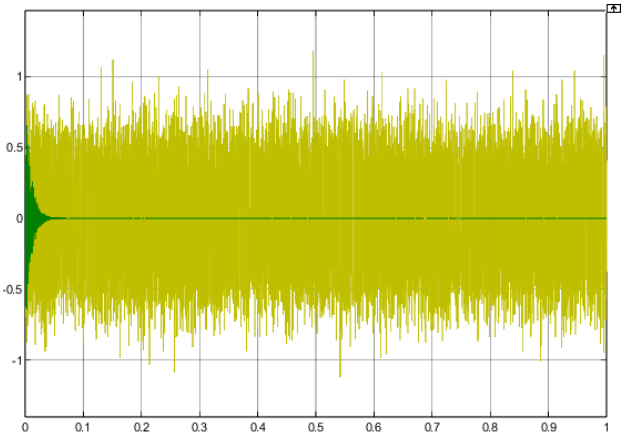


Fig. 12. Simulink Secondary Path Estimation Simulation Convergence

VI Physical Construction and Integration

VI-A Digital

Using the Simulink secondary path estimation and FxLMS algorithms shown in Figures 11 and 9, the block diagram was created in Figure 14. This implementation uses two

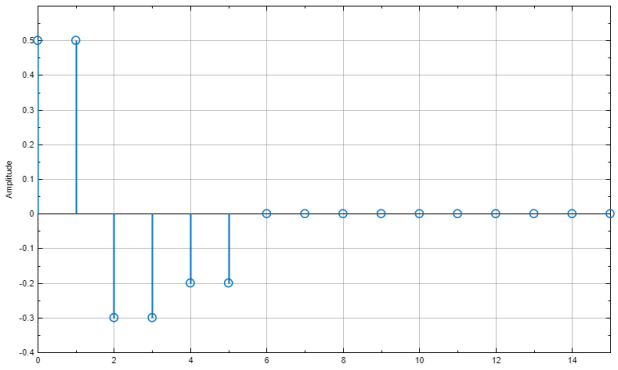


Fig. 13. Simulink Secondary Path Estimation Simulation Coefficients

subsystems that are triggered sequentially. First, the secondary path estimation runs for 30 seconds, then the coefficients that it converges to are input to the inverse filter in the FxLMS algorithm, then the FxLMS algorithm runs for the rest of the time. All the inputs and outputs were left open so they could be controlled by the Teensy MCU. Using the embedded coder app in Simulink, C++ code was created and exported to the Arduino IDE. Then, all the inputs and outputs were connected to their respective components in the physical implementation.

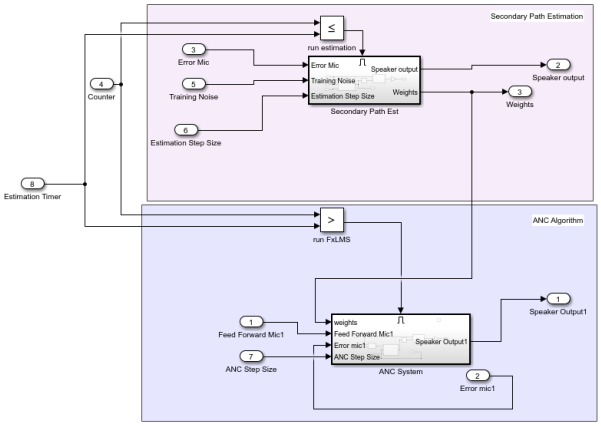


Fig. 14. Simulink Block Diagram System Setup

VI-B Electrical

The electrical components were connected by breadboard for simplicity of development. A close up view of the final connections is shown in Figure 15. All the components were placed on the same ear cup so no long wires were required over the top of the headband. The teensy MCU and usb charging port were placed at the bottom so their usb ports could easily be access for charging and programming. The audio codec was placed next to the teensy because they have 7 ports directly connected together.

VI-C Mechanical

During the assembly phase, the physical construction of the headphone system was based largely on the open-source

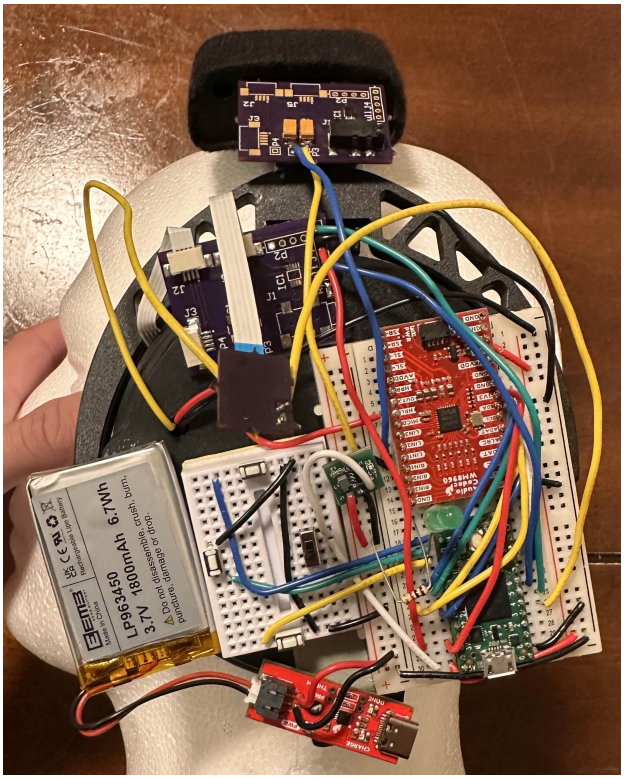


Fig. 15. Close up of Electronics

guidelines provided in the PloopyCo Wiki, which served as the foundation for our mechanical design. However, several modifications were necessary to accommodate the added functionality of passive and active noise cancellation, as well as to support internal component mounting.

The most significant deviation from the original PloopyCo assembly involved the earcup configuration. Unlike PloopyCo's original design—which was intended for standard speaker drivers without passive sealing requirements—our system required enhanced acoustic isolation. To achieve this, the earcup indentations were filled in the CAD models to create a more enclosed geometry. A hole was drilled in the rear of each earcup to allow for wire routing through layers of acoustic foam, enabling secure and acoustically damped connections between internal components and external interfaces.

The final assembly includes:

Earcups: Printed in PLA with 40% infill, each houses a 40mm driver, error microphone, and interior foam padding.

Headband: Adjustable, connects the left and right earcups via mechanical pivots and encloses the reference microphones and wiring channels.

PCBs: Mounted on the outside of each earcup, including the microphone hub, codec breakout board, and power distribution lines.

Aux port and mode switch: Accessible on the lower exterior via breadboard-mounted ports.

There is no dedicated cooling system, as the power draw of the system components (Teensy MCU, WM8960 codec, MEMS microphones) remained well within thermal limits during testing.

Fig. 16. Full System (Side Facing)

VII Integrated System Tests and Results

System integration was evaluated through a combination of functional and performance-based tests that aligned with the defined specifications. Testing focused on verifying active and passive noise attenuation, battery performance, mechanical dimensions, and core usability features.

VII-A Noise Attenuation Test Setup

The primary qualitative metric for system success was noise attenuation performance, both passive and active. Testing was conducted using a pink noise source placed one foot from a 3D-printed styrofoam head model, with a bore hole drilled through the ear canal to house a calibrated measurement microphone. Three test conditions were recorded:

1) Open ear baseline (no headphones)
2) Headphones on with ANC disabled (passive only)
3) Headphones on with ANC enabled (active)

Measurements were collected using MATLAB, with the test code provided in Appendix E. Due to limitations in acoustic isolation in the testing environment, raw data was post-processed using averaging across multiple trials to reduce ambient noise artifacts and emphasize general attenuation trends. Figure 17 presents the resulting frequency response compared to a commercial benchmark (Bose QC Ultra).

VII-B Battery and Mechanical Validation

Battery performance was validated under full load (ANC enabled, audio playback active) and achieved a continuous runtime of **2 hours**, exceeding the design goal of 1 hour. The fully assembled system was also weighed, with a final mass of **500 g**, exceeding the target of 350 g due to high infill density used for structural rigidity.

Dimensional measurements, such as headband length and earmuff size, were taken with a caliper and tape measure.

Certain ergonomics specifications—like total weight and earmuff dimensions—fell outside tolerance, while others, such as headband adjustability and modular part replacement, were met.

VII-C Test Summary

Figure 18 shows the fully integrated system. The overall verification results are presented in Table III, which compares specification targets against measured performance and indicates whether each requirement was met.

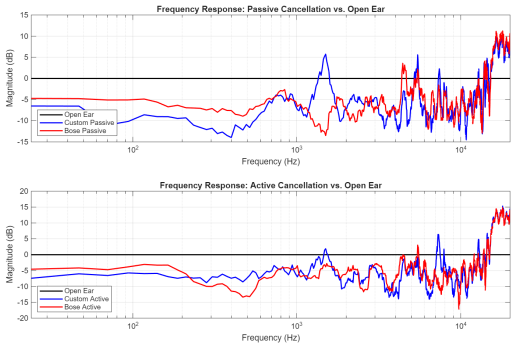


Fig. 17. Passive and Active Attenuation Compared to Bose QC Ultra (Pink Noise Test)

VIII Conclusions

While the final system did not fully meet the original design specifications, the project yielded several valuable lessons that inform future design iterations. As summarized in Table III, the most significant deficiencies were in passive and active noise attenuation.

The system's passive noise cancellation was minimal. This limitation is likely attributable to two factors. First, the use of PLA for the earcup housing provided little inherent acoustic damping. Additionally, the internal honeycomb infill structure typical in FDM 3D printing may have further degraded isolation performance. Second, the foam and seal used for the earpads were insufficient to acoustically isolate the environments of the two microphones (reference and error). As a result, the active noise cancellation algorithm struggled to converge, likely due to signal contamination—i.e., the microphones could not adequately differentiate between ambient noise and the anti-noise signal.

To address these shortcomings, future versions of the system could adopt a hybrid approach. Specifically, a commercially available set of passive noise-cancelling headphones could be modified to integrate the active noise cancellation electronics, benefiting from better sealing and acoustics out of the box. This approach would preserve repairability and modularity goals while significantly improving baseline attenuation performance.

Further enhancements may include redesigning the mechanical housing with materials or techniques that offer superior acoustic insulation, and improving microphone placement and isolation. These changes would provide clearer signal separation, enabling better algorithmic convergence and user-perceived performance—ultimately delivering more value to the intended customer base.

Fig. 18. View of Fully Assembled System (Front Facing)

TABLE III
SYSTEM PERFORMANCE VERIFICATION SUMMARY

Spec #	Parameter Description	Requirement or Target (Units)	Tolerance	Risk	Compliance	Measured Performance	Requirement Met?
1	Estimation Time	22.5 (μs)	Max.	H	A, T	204 μs	N
2	Passive Cancellation Range	20 dB above 501 Hz	Min.	L	T, D	0.86 dB	N
3	Active Cancellation Range	10 dB within 300–500 Hz	Min.	H	A, T	6.16 dB	N
4	Price	\$250	Max.	M	A, S	–	N
5	Components	Flat Frequency Response	P/F	M	T	F	N
6	Repairable	Replaceable Battery / Earmuffs	P/F	L	I, D	P	Y
7	Comfortable	7 cm tall, 4.5 cm wide	$\pm$5%	L	A, I, T, D	10 cm tall, 11 cm wide	N
8	Adjustable Headband	12–15 in	$\pm$5%	H	A, I, T, D, S	13.3 in	Y
9	Weight	350 g	Max.	M	T, D, S	500 g	N
10	Connectivity	3.5 mm Jack (Aux)	P/F	L	A, I, T, D	P	Y
11	Bluetooth Range	10 m	Min.	M	T, D	n/a	n/a
12	Battery Life	1 hour	Min.	H	T, D, A	2 hours	Y
13	Modes	2 (Passive, Active)	P/F	M	A, I, T, D	2	Y

IX Bibliography

References

[1] Ploopy Corporation, "Ploopy headphones open-source repository," https://github.com/ploopyco/headphones/tree/master, n.d., accessed: 2025-06-09.

[2] Carnegie Mellon University, "Adaptive filtering lecture notes," n.d., lecture 27, ECE 491, Carnegie Mellon University. [Online]. Available: https://course.ece.cmu.edu/~ece491/lectures/L27/AdaptiveFilteringChap_ADSP.pdf

[3] M. Stamenovic, "A review of active noise control algorithms towards a user-implementable aftermarket anc system," 2016, available online. [Online]. Available: https://hajim.rochester.edu/ece/sites/zduan/teaching/ece472/projects/2016/Stamenovic_paper.pdf

[4] D. Gauger, M. Feder, K. C. Zangi, E. Weinstein, and A. V. Oppenheim, "Single-sensor active noise cancellation," *IEEE Journals & Magazine*, Aug. 2002, https://ieeexplore.ieee.org/abstract/document/279277.

[5] Y. Song, Y. Gong, and S. M. Kuo, "A robust hybrid feedback active noise cancellation headset," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 4, pp. 607–617, Jul. 2005.

[6] Y. Shiang-Hwua and H. Jwu-Sheng, "Controller design for active noise cancellation headphones using experimental raw data," *IEEE/ASME Transactions on Mechatronics*, vol. 6, no. 4, pp. 483–490, 2001.

[7] H. Møller, "Fundamentals of binaural technology," *Applied Acoustics*, vol. 36, no. 3–4, pp. 171–218, 1992.

[8] K.-H. Chen *et al.*, "Design of an efficient active noise cancellation circuit for in-ear headphones," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Nov. 2014.

[9] Bose, "Quietcomfort ultra headphones — bose," Oct. 2024, https://www.bose.com/p/headphones/bose-quietcomfort-ultra-headphones/QCUH-HEADPHONEARN.html.

[10] Sennheiser, "Momentum 4 wireless," 2024, https://www.amazon.com/Sennheiser-Consumer-Audio-Momentum-Headphones/dp/B0CDH415QV/.

[11] Sony Electronics, "Wh1000xm5/b," 2022, https://electronics.sony.com/audio/headphones/headband/p/wh1000xm5-b.

[12] H. Møller, C. B. Jensen, D. Hammershøi, and M. F. Sørensen, "Design criteria for headphones," Acoustics Laboratory, Aalborg University, Tech. Rep., 1995, https://aes2.org/publications/elibrary-page/?id=10274.

[13] MathWorks, "Active noise control with simulink real-time - matlab & simulink," `https://www.mathworks.com`, Jun. 2017, https://www.mathworks.com/help/audio/ug/active-noise-control-with-simulink.html#d126e12111.

X Appendices

APPENDIX A
ANALYSIS OF SENIOR PROJECT DESIGN

Project Title: FxLMS-Based ANC in Custom 3D-Printed Headphones
Student's Name: Seth Cherry **Student's Signature:** Seth Cherry
Student's Name: Andrew Chookaszian **Student's Signature:** Andrew Chookaszian
Advisor's Name: _____ **Advisor's Initials:** _____ **Date:** _____

Summary of Functional Requirements

The active noise cancellation (ANC) headphone system provides users with both active and passive sound attenuation. Functionally, the system achieves approximately 10 dB of attenuation in the 300–500 Hz range using active feedback, and up to 20 dB of passive attenuation across a broader frequency range. It supports three Bluetooth modes—pass-through, passive, and quiet—allowing users to adapt audio transparency based on their environment. The device also includes a standard 3.5 mm audio input for wired use when battery power is unavailable. With a total weight of approximately 350 grams, the headphones are lightweight and portable. Additionally, the system is designed for ease of maintenance and customization, featuring modular components such as replaceable earmuffs, an adjustable headband, and user-accessible 3D-printable STL files for component replacement.

Primary Constraints

Several constraints significantly influenced the design and development of the ANC headphone system. First, complete background noise cancellation is not physically feasible with consumer-grade ANC technology, which required us to set realistic expectations and focus on a limited frequency range for active attenuation.

In practice, mechanical design factors—such as the seal of the ear pads against the head, the width and density of the acoustic dampening foam, and the infill percentage of 3D-printed components—had a substantial effect on the system's passive attenuation performance. These factors limited the effectiveness of the active feedback and feedforward microphones, as poor isolation between them made it difficult for the algorithm to differentiate the ambient noise from the desired signal. As a result, the ANC algorithm had difficulty converging on physical hardware.

Another constraint arose from our commitment to designing a repairable, modular headphone system rather than adapting an existing commercial model. While modifying a pre-built system with effective passive isolation could have improved performance, it would have conflicted with our design goals around user repairability and customization. Finally, the weight of the system became a constraint. Our modular design required separating components across multiple circuit boards rather than consolidating them on a single PCB. Although this approach supported flexibility and repairability, it increased the overall weight of the headset and presented comfort challenges.

Economic

XI Project Cost and Timeline Summary

XI-A Original Estimated Cost of Component Parts

At the start of the project, the estimated cost for component parts was approximately $113.57. This estimate was based on early prototyping needs and assumed reuse of non-critical parts (e.g., speaker grills) to control cost.

XI-B Actual Final Cost of Component Parts

The final cost of all components—after accounting for changes in quantities, part selections, and additional purchases—is documented in Table IV. The complete and updated bill of materials is included below:

TABLE IV
ORIGINAL BILL OF MATERIALS

Part	Model	Price	Qty	Explanation
MCU	Teensy 4.0	$23.80	2	DSP-capable, compact
Mic	Same Sky Mic	$1.58	10	9.7 mm × 4.5 mm, compact
Speaker	40mm 32Ω driver	$3.49	2	0.5 W audio drivers
Grills	Reused	—	—	From old headphones
Battery	3.7V 1200mAh LiPo	$9.95	1	Wireless power source
Filament	PLA	$16.00	1	3D print housing
Earcups	Bose QC35 Repl.	$8.99	1	Effective passive isolation
Amp	TDA2822D	$1.52	2	Drives 32Ω at 300 mW
Audio CODEC	WM8960 Breakout	$17.95	2	I2S DAC/ADC interface
Bluetooth	AudioB I2S Module	$11.95	1	Wireless audio input
Potentiometer	Stereo Pot	$1.82	1	Volume control
Switch	Tactile 6mm	$4.95	1	Power/mode switch
Converter	COM-15208	$11.57	2	Buck/boost regulator
Aux Jack	SJ2-35863B1	$1.06	2	Wired input option
Knobs	1528-5542-ND	$7.50	1	Potentiometer knobs
Total		**$113.57**		

TABLE V
ALL PARTS FOR ANC HEADPHONES WITH PRICING

Part	Model #	Price per unit	Amount	Total Price
MCU/DSP	PJRC Teensy 4.0 USB Development Board	$23.80	1	$23.80
Mode Switch	Tactile Switch Buttons	$4.95	1	$4.95
Audio Codec	SparkFun Audio Codec Breakout - WM8960 (Qwiic)	$17.95	1	$17.95
Power Switch	Three Way Switch	$5.49	1	$5.49
Battery	103454 Lipo Rechargeable	$11.99	1	$11.99
Boost Converter	1528-4654-ND	$3.95	2	$7.90
Noise Dampening Foam	Polyethylene Foam Roll	$9.99	1	$9.99
Recylce PLA	Protopasta Recycled PLA	$15.00	1	$15.00
Headphone Driver	CE38MB-32	$3.49	2	$6.98
Volume Rocker Switch	R4GBLKBLKEF0	$2.21	1	$2.21
Single 2:1 Data Mux	SN74LVC2G157	$0.38	1	$0.38
100nF SM Capacitor	C0603C104K4RACTU	$0.08	6	$0.48
Digital MEMS Microphone	ICS-52000	$3.49	6	$20.94
25 Ohm SM Resistor	SN74LVC2G161	$0.68	6	$4.08
100K ohm Resistor	RC0603FR-07100KL	$0.10	2	$0.20
Headphone Jack	SJ2-35863B1-SMT-TR	$1.07	2	$2.14
LiPo Charging Port	SparkFun LiPo Charger Plus	$11.50	1	$11.50
FPC 5 Wire 304.8 mm		$1.08	2	$2.16
FPC 5 Wire Connector	527930570	$0.60	4	$2.40
FPC 4 wire Connector	527930470	$0.51	10	$5.10
FPC 4 wire 30 mm(1.18”) cable	151670701	$1.28	2	$2.56
FPC 4 wire 51 mm(2”) cable	151670707	$1.97	2	$3.94
FPC 4 wire 76 mm(3”) cable	151670707	$1.86	2	$3.72
FPC 4 wire 101.6 mm(4”) cable	151670707	$1.70	6	$10.20
220 uF Capacitors		$0.81	4	$3.24
Connector PCB 1		—	—	—
Connector PCB 2		$27.00	—	$27.00
Digital Microphone PCB		$14.80	—	$14.80
Stockinette	Nuanchu Cotton Stockinette Tubular Elastic Bandage	$14.99	1	$14.99
Metal Dowels	3 x 12mm metal dowels	$6.19	1	$6.19
ADC	PCM1802 Audio Stereo A/D Converter ADC Decoder	$10.99	1	$10.99

XI-C Additional Equipment Costs

The most significant equipment expense would have been the 3D printer required to fabricate the headphone casing and internal supports. However, because the team had access to a printer throughout development, no additional capital investment

was required. Without this access, the cost of purchasing a reliable consumer-grade 3D printer would likely exceed $200–$300, substantially increasing the overall budget.

XI-D Original Estimated Development Time

Figure 19 shows the initial project schedule developed at the beginning of EE460. The projected development timeline spanned from September 2024 to late May 2025.

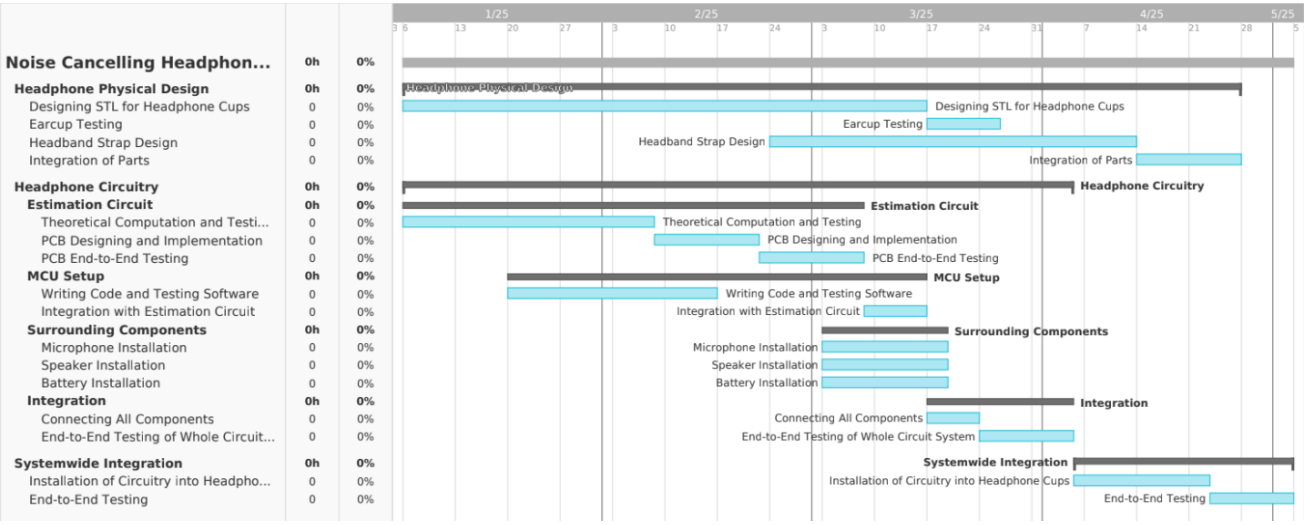


Fig. 19. Original Gantt Chart (from EE460)

XI-E Actual Development Time

The final development timeline is detailed in Table VI. While most milestones remained consistent with the original plan, the final tasks extended into early June 2025 due to testing and iteration needs.

TABLE VI
DETAILED PROJECT SCHEDULE – ANC HEADPHONES

Task ID	Task Name	Description	Start Date	End Date
3728100	Choose Project	Select senior project topic and confirm scope	Sep 24, 2024	Sep 29, 2024
3728109	CTQ Tree	Develop Critical to Quality requirements tree	Sep 30, 2024	Oct 08, 2024
3728101	PRD	Write Product Requirements Document	Oct 01, 2024	Nov 02, 2024
3728107	Engineering Specifications	Define technical specifications	Oct 07, 2024	Oct 15, 2024
3728102	Monte Carlo Analysis	Run statistical analysis on design parameters	Oct 28, 2024	Nov 12, 2024
3728103	ABET Analysis	Complete required ABET outcome mapping	Nov 04, 2024	Nov 26, 2024
3728110	Functional Decomposition	Break down major system functions	Nov 04, 2024	Nov 19, 2024
3728104	Preliminary Design Presentation	Present early system design to advisors	Nov 18, 2024	Dec 07, 2024
3728105	Final Report (EE460)	Complete and submit first term final report	Nov 25, 2024	Dec 14, 2024
3728111	Choose Print Materials	Select PLA types and finalize STL files	Jan 06, 2025	Jan 11, 2025
3728112	Background Research	Collect research on ANC, hardware, design	Jan 13, 2025	Feb 04, 2025
3728114	Printing Parts	3D print housing and mechanical parts	Jan 20, 2025	Apr 15, 2025
3728115	Choosing Components	Finalize selection of electrical components	Jan 27, 2025	Mar 15, 2025
3728113	FxLMS Algorithm (Matlab)	Design and simulate ANC control algorithm	Feb 10, 2025	Mar 11, 2025
3728116	PCB Design	Layout circuit boards for fabrication	Feb 24, 2025	Apr 08, 2025
3728117	Order Components	Place orders for electrical parts	Apr 15, 2025	Apr 27, 2025
3728118	Software Writing	Write firmware and interface code	Apr 21, 2025	May 03, 2025
3729133	Assembly	Assemble the headphone system	Apr 29, 2025	May 10, 2025
3729134	Testing	Validate system performance and safety	May 09, 2025	May 17, 2025
3729135	Iterations & Final Report	Address design issues and submit final report	May 16, 2025	Jun 09, 2025

Environmental

The most significant environmental impacts of our headphone system stem from PCB manufacturing, raw material extraction, and the battery life cycle. These impacts are consistent with those of comparable consumer electronics, particularly due to

the use of silicon, plastic, and metals such as lithium. Extraction processes for these materials are known to cause substantial environmental degradation and habitat loss.

To mitigate some of these impacts, we have explored the use of eco-friendly 3D printing materials, such as recycled PLA (R-PLA) and mineral-filled PLA, which offer reduced environmental footprints compared to conventional plastics. Additionally, the foam used for the ear cups is generally recyclable.

Importantly, our modular design supports component-level repair and replacement, meaning that users do not need to discard the entire device if a single part fails. This feature can significantly reduce electronic waste over the product's life cycle.

Manufacturability

One of the primary challenges associated with manufacturing this product is the limited team size. With only two individuals responsible for assembling each unit, the current production capacity is constrained to a relatively small scale, likely no more than a few thousand units annually. This significantly limits the feasibility of meeting high market demand without substantial workforce expansion.

Additionally, the use of a 3D-printed casing introduces a critical bottleneck in the production process. At present, only one 3D printer is available, which restricts output to approximately two to three units per day. This creates a significant scalability issue if larger batch production is required. Addressing this would necessitate a substantial capital investment—either in acquiring multiple additional printers or in upgrading to higher-speed industrial-grade printing equipment.

These factors combined present a significant barrier to scaling manufacturing operations, and any path toward mass production would require a detailed strategy for resource allocation, equipment procurement, and possibly workforce expansion.

Sustainability

The modular design of our headphone system offers a significant advantage in terms of sustainable resource use. Instead of discarding the entire device when one part fails, users can replace individual components—either through reordering or 3D-printing replacements. This approach minimizes electronic waste and enables upgrades over time by allowing older components to be swapped out for newer versions.

However, several challenges exist for long-term maintenance. If replacement parts become unavailable in the future, users may no longer be able to repair their devices. Additionally, although modularity is intended to support repairs, it may inadvertently lead to increased damage if users attempt disassembly without proper guidance. Furthermore, studies show that many consumers value repairability but are reluctant to carry out repairs themselves [4].

To address this, one proposed upgrade is the establishment of a technician support network. A service model where trained professionals assist with repairs could increase user confidence and reduce waste from improperly handled upgrades or maintenance. This network could be community-based or partnered with local repair organizations.

A challenge in upgrading the design lies in ensuring backward compatibility of future components and encouraging consistent part availability. The success of the system's sustainability model depends on maintaining long-term support and user access to parts and repair tools.

Ethical

One key ethical concern in our headphone design is accessibility and affordability. While we aim to reduce the cost of our ANC system, it remains relatively expensive compared to non-ANC alternatives. This raises concerns about equity, particularly for low-income communities in urban areas where noise pollution is highest. These communities stand to benefit the most from our product, yet may find it financially inaccessible. This presents a tension between utilitarian ethics, which prioritize the greatest good for the greatest number, and duty-based ethics, which emphasize doing what is morally right—such as prioritizing support for the most impacted populations.

Another ethical dimension concerns sustainability and environmental impact. Manufacturing new electronics involves the extraction of rare materials and the creation of electronic waste, which contradicts IEEE's ethical principle to "hold paramount the safety, health, and welfare of the public… and strive to comply with ethical design and sustainable development practices." While our design supports modular repair and upgrades to reduce waste, the act of producing new devices still contributes to resource depletion and environmental strain.

We recognize these ethical challenges and are actively considering solutions. One long-term strategy may involve creating subsidized pricing models or donation programs for high-noise, underserved communities. We are also exploring ways to enhance repairability and longevity, minimizing the environmental footprint and maximizing product life. As development progresses, we aim to refine these strategies to better align with ethical design principles and public welfare.

Health and Safety

As this project involves direct sound output, a key health concern is the risk of hearing damage. To mitigate this, the system must incorporate software and hardware safeguards to prevent audio playback at levels that could cause temporary or permanent hearing loss, either for the user or those nearby. This includes volume-limiting algorithms or hard caps in firmware to prevent unsafe output levels.

The ergonomic design of the headset is also essential for user comfort and safety. The headband and earcups must be designed to avoid excessive pressure, heat buildup, or prolonged discomfort during extended wear, especially in occupational settings.

From an electrical standpoint, the device must adhere to safety standards to prevent overcurrent conditions, short circuits, or electrostatic discharge (ESD) events, particularly during charging. Ensuring proper insulation, grounding, and fail-safe mechanisms is necessary to avoid any risk of electrical injury or device failure.

Additionally, the product offers potential health benefits. In high-noise environments—such as industrial workplaces—the ANC system can reduce exposure to harmful sound levels, helping to prevent long-term hearing loss. It may also enhance focus and productivity by reducing ambient distractions in noisy public or office settings.

Overall, our design aims to balance functionality with a strong commitment to user safety and health protection across all stages of use.

Social and Political

While the primary purpose of our headphone system is consumer-focused entertainment and utility, there are important social concerns related to economic accessibility. The relatively high price point of the product may limit availability to individuals in lower-income communities, who are often the most affected by chronic noise pollution—especially in densely populated urban areas. This creates a disparity between those who need the benefits of noise-canceling technology for health and well-being and those who can afford it.

From a social equity perspective, this raises concerns about how technological solutions to environmental and health-related problems can unintentionally exclude vulnerable populations. In this context, affordability becomes not just a market consideration, but a social responsibility.

Although no direct political controversy is currently associated with the product, the broader issue intersects with public policy around environmental justice and urban planning. Many cities and advocacy groups are increasingly recognizing the disproportionate impact of noise pollution on under-resourced neighborhoods, which could eventually inform regulatory support or incentives for noise-reducing technologies. As such, ensuring that our product can be adapted or subsidized for these communities may align with both social goals and potential future policy frameworks.

Development

One of the main tools learned in the development of this project was adaptive filters. These filters were at the center of this project and no classes taken had taught about them, therefore, an independent study was required to understand how they work and how to develop them.

During PCB development, Altium was chosen because it is commenly used in the commercial sector, therefore it was seen as an important skill to learn. Using online instruction, the basics of Altium were covered, and the PCBs were able to be developed.

Simulink was an essential tool during this project which allowed for the creation of the FxLMS algorithm. However, the algorithm developed in Simulink needed to be converted to code so it could be interfaced with the Arduino program. This is where the embedded coder app was used to directly convert the Simulink algorithm to c++ code, then interfaced with the program.

Engineering Standards

Since this project evolved into a more research-focused effort than originally anticipated, the final design is currently closer to a prototype than a commercially manufacturable product. However, that does not mean we have neglected to consider the relevant engineering standards that would apply in a streamlined production scenario.

First, we would need to ensure compliance with audio performance standards, such as ANSI/CTA-2054 – Personal Sound Amplification Products (PSAPs) Performance Criteria, to ensure safe and effective sound output. Additionally, we would need to meet electrical and wireless safety standards, including IEC 62133 for lithium-ion battery safety and IEEE 802.15.1 for Bluetooth wireless communication.

Finally, to prevent electromagnetic interference with other devices, our design would need to conform to the IEC 61000 series of electromagnetic compatibility (EMC) standards. While the product is still in a developmental stage, these standards provide a clear roadmap for future iterations intended for consumer use.

XI-F Parts List and Costs

TABLE VII
ALL PARTS FOR ANC HEADPHONES WITH PRICING

Part	Model #	Price per unit	Amount	Total Price
MCU/DSP	PJRC Teensy 4.0 USB Development Board	$23.80	1	$23.80
Mode Switch	Tactile Switch Buttons	$4.95	1	$4.95
Audio Codec	SparkFun Audio Codec Breakout - WM8960 (Qwiic)	$17.95	1	$17.95
Power Switch	Three Way Switch	$5.49	1	$5.49
Battery	103454 Lipo Rechargeable	$11.99	1	$11.99
Boost Converter	1528-4654-ND	$3.95	2	$7.90
Noise Dampening Foam	Polyethylene Foam Roll	$9.99	1	$9.99
Recylce PLA	Protopasta Recycled PLA	$15.00	1	$15.00
Headphone Driver	CE38MB-32	$3.49	2	$6.98
Volume Rocker Switch	R4GBLKBLKEF0	$2.21	1	$2.21
Single 2:1 Data Mux	SN74LVC2G157	$0.38	1	$0.38
100nF SM Capacitor	C0603C104K4RACTU	$0.08	6	$0.48
Digital MEMS Microphone	ICS-52000	$3.49	6	$20.94
25 Ohm SM Resistor	SN74LVC2G161	$0.68	6	$4.08
100K ohm Resistor	RC0603FR-07100KL	$0.10	2	$0.20
Headphone Jack	SJ2-35863B1-SMT-TR	$1.07	2	$2.14
LiPo Charging Port	SparkFun LiPo Charger Plus	$11.50	1	$11.50
FPC 5 Wire 304.8 mm		$1.08	2	$2.16
FPC 5 Wire Connector	527930570	$0.60	4	$2.40
FPC 4 wire Connector	527930470	$0.51	10	$5.10
FPC 4 wire 30 mm(1.18”) cable	151670701	$1.28	2	$2.56
FPC 4 wire 51 mm(2”) cable	151670707	$1.97	2	$3.94
FPC 4 wire 76 mm(3”) cable	151670707	$1.86	2	$3.72
FPC 4 wire 101.6 mm(4”) cable	151670707	$1.70	6	$10.20
220 uF Capacitors		$0.81	4	$3.24
Connector PCB 1		—	—	—
Connector PCB 2		$27.00	—	$27.00
Digital Microphone PCB		$14.80	—	$14.80
Stockinette	Nuanchu Cotton Stockinette Tubular Elastic Bandage	$14.99	1	$14.99
Metal Dowels	3 x 12mm metal dowels	$6.19	1	$6.19
ADC	PCM1802 Audio Stereo A/D Converter ADC Decoder	$10.99	1	$10.99

XI-G Project Schedule – Original Time Estimates & Actual Dates Achieved

XI-H PCB Layout

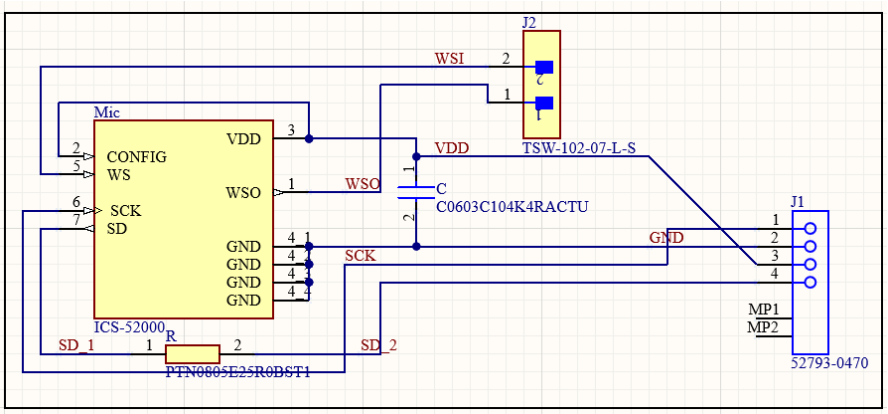


Fig. 20. Microphone Connection Hub Schematic

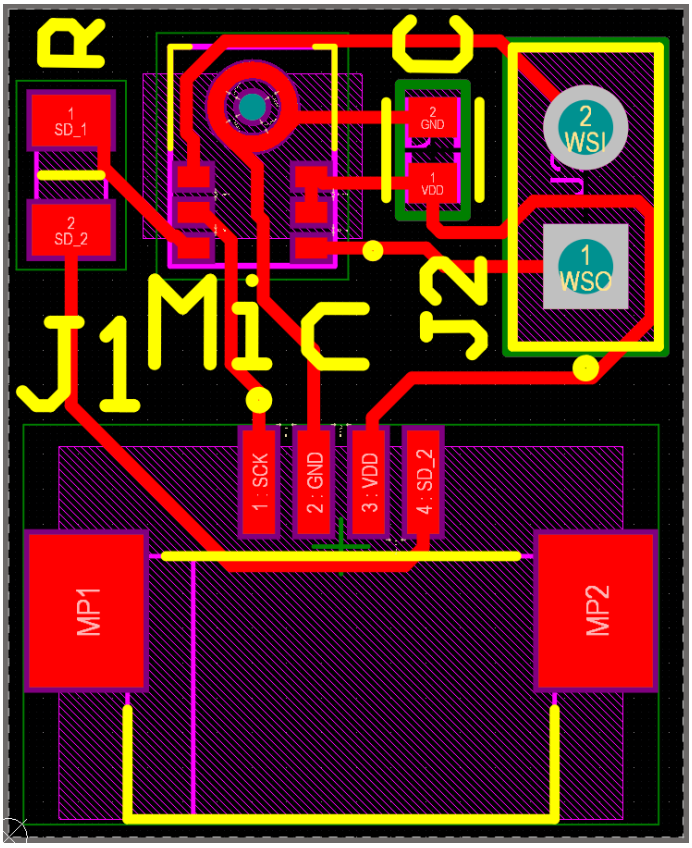


Fig. 21. Microphone Hub PCB Layout

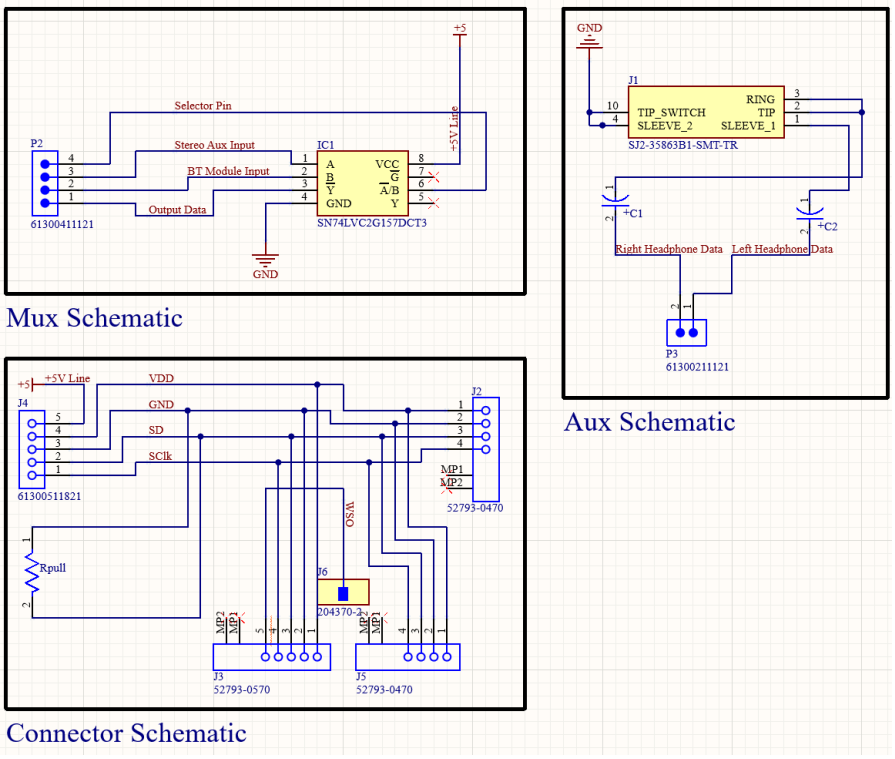


Fig. 22. Connector PCB Schematic

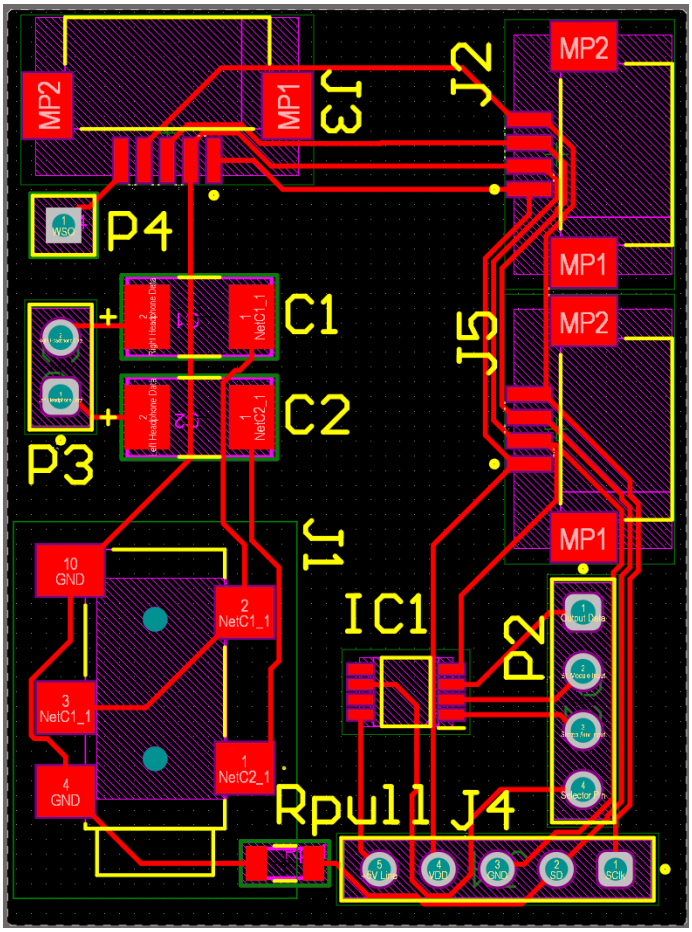


Fig. 23. Connector PCB Layout with Aux and MUX Routing

XI-I Program Listings

1) ANC Impulse Suppression

Listing 1. ANC Impulse Suppression Code

```
#include <Audio.h>
#include <Wire.h>
#include <SparkFun_WM8960_Arduino_Library.h>
#include "ANCBlockProcessor.h"

// --- Pin definitions ---
const int VOL_UP_PIN = 14;
const int VOL_DOWN_PIN = 16;
const int MODE_SWITCH_PIN = 15;
const int MODE_LED = 17;

// --- Global state ---
ANCMode currentMode = MODE_ON;
float volume = 1.0;
bool lastModeButtonState = HIGH;
bool lastVolUpState = HIGH;
bool lastVolDownState = HIGH;

unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;  // ms

bool lastButtonState = HIGH;
bool currentButtonState = HIGH;


WM8960 codec;
ANCBlockProcessor ancProcessor;

// --- Audio Routing ---
AudioInputTDM2 tdm2;
AudioInputI2S i2sIn;  // Music from AUX (Line In)
AudioOutputI2S i2s1;
AudioMixer4 mixer;

AudioConnection patchFF(tdm2, 0, ancProcessor, 0);    // Feedforward mic
AudioConnection patchErr(tdm2, 2, ancProcessor, 1);   // Error mic
AudioConnection patchANC(ancProcessor, 0, mixer, 0);  // ANC output
AudioConnection patchMusic(i2sIn, 0, mixer, 1);       // Music AUX input
AudioConnection patchOut(mixer, 0, i2s1, 0);          // Output L
// AudioConnection patchOutR(mixer, 0, i2s1, 1);         // Output R

void setup() {
  Serial.begin(115200);
  Wire.begin();
  delay(100);

  if (!codec.begin()) {
    Serial.println("Codec not detected.");
    while (1)
      ;
  }

```

```
// --- Input path: route Line In 1 (AUX) to I2S output ---
codec.enableVREF();
codec.enableVMID();
codec.connectLMN1();  // LINPUT1 to left mixer
// codec.connectRMN1();  // RINPUT1 to right mixer
codec.enableLMIC();
// codec.enableRMIC();
codec.disableLINMUTE();
// codec.disableRINMUTE();
codec.setLMICBOOST(WM8960_MIC_BOOST_GAIN_0DB);
// codec.setRMICBOOST(WM8960_MIC_BOOST_GAIN_0DB);
codec.connectLMIC2B();  // route to boost mixer
// codec.connectRMIC2B();
codec.enableAINL();
// codec.enableAINR();

// --- Output path ---
codec.disableLB2LO();
// codec.disableRB2RO();
codec.enableLD2LO();
// codec.enableRD2RO();
codec.setLB2LOVOL(WM8960_OUTPUT_MIXER_GAIN_0DB);
// codec.setRB2ROVOL(WM8960_OUTPUT_MIXER_GAIN_0DB);
codec.enableLOMIX();
// codec.enableROMIX();
codec.enableHeadphones();
codec.enableOUT3MIX();

// --- Clocking ---
codec.enablePLL();
codec.setPLLPRESCALE(WM8960_PLLPRESCALE_DIV_2);
codec.setSMD(WM8960_PLL_MODE_FRACTIONAL);
codec.setCLKSEL(WM8960_CLKSEL_PLL);
codec.setSYSCLKDIV(WM8960_SYSCLK_DIV_BY_2);
codec.setBCLKDIV(4);
codec.setDCLKDIV(WM8960_DCLKDIV_16);
codec.setPLLN(7);
codec.setPLLK(0x86, 0xC2, 0x26);
codec.enablePeripheralMode();

codec.enableAdcLeft();
// codec.enableAdcRight();
codec.enableDacLeft();
// codec.enableDacRight();
codec.disableDacMute();
codec.setHeadphoneVolumeDB(-10.0);

AudioMemory(60);

pinMode(VOL_UP_PIN, INPUT_PULLUP);
pinMode(VOL_DOWN_PIN, INPUT_PULLUP);
pinMode(MODE_SWITCH_PIN, INPUT_PULLUP);
pinMode(MODE_LED,OUTPUT);
digitalWrite(MODE_LED,LOW);

mixer.gain(0, 1.0);     // ANC signal (fixed)
```

```
  mixer.gain(1, volume);  // Music signal (variable)
  mixer.gain(2, 0);
  mixer.gain(3, 0);

  ancProcessor.setMode(currentMode);
  digitalWrite(MODE_LED,HIGH);
  ancProcessor.setVolume(volume);

  Serial.println("Secondary Path Estimation Starting");
}

void loop() {
  // --- Read button states ---
  bool modePressed = digitalRead(MODE_SWITCH_PIN) == LOW;
  bool volUpPressed = digitalRead(VOL_UP_PIN) == LOW;
  bool volDownPressed = digitalRead(VOL_DOWN_PIN) == LOW;

  bool reading = digitalRead(MODE_SWITCH_PIN);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();  // reset debounce timer
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != currentButtonState) {
      currentButtonState = reading;

      if (currentButtonState == LOW) {  // button press detected
        currentMode = (currentMode == MODE_ON) ? MODE_OFF : MODE_ON;
        ancProcessor.setMode(currentMode);
        Serial.printf("ANC Mode: %s\n", currentMode == MODE_ON ? "ON" : "OFF");
        if(currentMode==MODE_ON){
          digitalWrite(MODE_LED,HIGH);
        }
        else{
          digitalWrite(MODE_LED,LOW);
        }
      }
    }
  }

  lastButtonState = reading;


  // --- Volume Up ---
  if (volUpPressed && lastVolUpState == HIGH) {
    volume = min(volume + 0.1f, 1.0f);
    mixer.gain(1, volume);
    ancProcessor.setVolume(volume);
    Serial.printf("Volume: %.2f\n", volume);
  }
  lastVolUpState = !volUpPressed ? HIGH : LOW;

  // --- Volume Down ---
  if (volDownPressed && lastVolDownState == HIGH) {
    volume = max(volume - 0.1f, 0.0f);
```

```
    mixer.gain(1, volume);
    ancProcessor.setVolume(volume);
    Serial.printf("Volume: %.2f\n", volume);
  }
  lastVolDownState = !volDownPressed ? HIGH : LOW;
}
```

2) ANC Implementation.h

Listing 2. ANC System Implmentation V2.h

```
//
// Academic License - for use in teaching, academic research, and meeting
// course requirements at degree granting institutions only.  Not for
// government, commercial, or other organizational use.
//
// File: ANC_system_Implementation_v2.h
//
// Code generated for Simulink model 'ANC_system_Implementation_v2'.
//
// Model version                  : 1.39
// Simulink Coder version         : 9.9 (R2023a) 19-Nov-2022
// C/C++ source code generated on : Tue Jun  3 20:14:31 2025
//
// Target selection: ert.tlc
// Embedded hardware selection: ARM Compatible->ARM Cortex-M
// Code generation objectives:
//    1. Execution efficiency
//    2. RAM efficiency
// Validation result: Not run
//
#ifndef RTW_HEADER_ANC_system_Implementation_v2_h_
#define RTW_HEADER_ANC_system_Implementation_v2_h_
#include "rtwtypes.h"
#include <stddef.h>
#ifndef struct_dsp_simulink_LMSUpdate
#define struct_dsp_simulink_LMSUpdate

struct dsp_simulink_LMSUpdate
{
  int32_T isInitialized;
  boolean_T TunablePropsChanged;
  real_T LeakageFactor;
  real_T Weights[64];
  real_T xBuffer[64];
};

#endif                                 // struct_dsp_simulink_LMSUpdate

extern "C"
{
  static real_T rtGetNaN(void);
  static real32_T rtGetNaNF(void);
}                                      // extern "C"

#define NOT_USING_NONFINITE_LITERALS   1

extern "C"
{
  extern real_T rtInf;
  extern real_T rtMinusInf;
  extern real_T rtNaN;
  extern real32_T rtInfF;
  extern real32_T rtMinusInfF;
```

```
  extern real32_T rtNaNF;
  static void rt_InitInfAndNaN(size_t realSize);
  static boolean_T rtIsInf(real_T value);
  static boolean_T rtIsInfF(real32_T value);
  static boolean_T rtIsNaN(real_T value);
  static boolean_T rtIsNaNF(real32_T value);
  struct BigEndianIEEEDouble {
    struct {
      uint32_T wordH;
      uint32_T wordL;
    } words;
  };

  struct LittleEndianIEEEDouble {
    struct {
      uint32_T wordL;
      uint32_T wordH;
    } words;
  };

  struct IEEESingle {
    union {
      real32_T wordLreal;
      uint32_T wordLuint;
    } wordL;
  };
}                                      // extern "C"

extern "C"
{
  static real_T rtGetInf(void);
  static real32_T rtGetInfF(void);
  static real_T rtGetMinusInf(void);
  static real32_T rtGetMinusInfF(void);
}                                      // extern "C"

// Class declaration for model ANC_system_Implementation_v2
class ANC_system final
{
  // public data and function members
 public:
  // Block signals and states (default storage) for system '<Root>'
  struct DW {
    dsp_simulink_LMSUpdate obj;        // '<S2>/Est LMS Update'
    dsp_simulink_LMSUpdate obj_p;      // '<S1>/ANC LMS Update'
    real_T DiscreteFIRFilter1_states[63];// '<S2>/Discrete FIR Filter1'
    real_T Delay_DSTATE[64];           // '<S1>/Delay'
    real_T AdaptiveFilter_states[63];  // '<S1>/Adaptive Filter'
    real_T EstimatedSecondarypath_states[63];// '<S1>/Estimated Secondary path'
    int32_T EstimatedSecondarypath_circBuf;// '<S1>/Estimated Secondary path'
  };

  // External inputs (root inport signals with default storage)
  struct ExtU {
    real_T FeedForwardMic1;            // '<Root>/Feed Forward Mic1'
    real_T Errormic1;                  // '<Root>/Error mic1'
```

```
    real_T ErrorMic;                   // '<Root>/Error Mic'
    real_T Counter;                    // '<Root>/Counter'
    real_T TrainingNoise;              // '<Root>/Training Noise'
    real_T EstimationStepSize;         // '<Root>/Estimation Step Size'
    real_T ANCStepSize;                // '<Root>/ANC Step Size'
    real_T EstimationTimer;            // '<Root>/Estimation Timer'
  };

  // External outputs (root outports fed by signals with default storage)
  struct ExtY {
    real_T SpeakerOutput1;             // '<Root>/Speaker Output1'
    real_T Speakeroutput;              // '<Root>/Speaker output'
    real_T Weights[64];                // '<Root>/Weights'
  };

  // Copy Constructor
  ANC_system(ANC_system const&) = delete;

  // Assignment Operator
  ANC_system& operator= (ANC_system const&) & = delete;

  // Move Constructor
  ANC_system(ANC_system &&) = delete;

  // Move Assignment Operator
  ANC_system& operator= (ANC_system &&) = delete;

  // External inputs
  ExtU rtU;

  // External outputs
  ExtY rtY;

  // model initialize function
  void initialize();

  // model step function
  void step();

  // Constructor
  ANC_system();

  // Destructor
  ~ANC_system();

  // private data and function members
 private:
  // Block states
  DW rtDW;
};

//-
//  The generated code includes comments that allow you to trace directly
//  back to the appropriate location in the model.  The basic format
//  is <system>/block_name, where system is the system number (uniquely
//  assigned by Simulink) and block_name is the name of the block.
```

```
166 //
167 //  Use the MATLAB hilite_system command to trace the generated code back
168 //  to the model.  For example,
169 //
170 //  hilite_system('<S3>')    - opens system 3
171 //  hilite_system('<S3>/Kp') - opens and selects block Kp which resides in S3
172 //
173 //  Here is the system hierarchy for this model
174 //
175 //  '<Root>' : 'ANC_system_Implementation_v2'
176 //  '<S1>'   : 'ANC_system_Implementation_v2/ANC System'
177 //  '<S2>'   : 'ANC_system_Implementation_v2/Secondary Path Est'
178
179 #endif                                 // RTW_HEADER_ANC_system_Implementation_v2_h_
180
181 //
182 // File trailer for generated code.
183 //
184 // [EOF]
185 //
```

3) ANC System Implementation.cpp

Listing 3. ANC System Implmentation V2.cpp

```
//
// Academic License - for use in teaching, academic research, and meeting
// course requirements at degree granting institutions only.  Not for
// government, commercial, or other organizational use.
//
// File: ANC_system_Implementation_v2.cpp
//
// Code generated for Simulink model 'ANC_system_Implementation_v2'.
//
// Model version                  : 1.39
// Simulink Coder version         : 9.9 (R2023a) 19-Nov-2022
// C/C++ source code generated on : Tue Jun  3 20:14:31 2025
//
// Target selection: ert.tlc
// Embedded hardware selection: ARM Compatible->ARM Cortex-M
// Code generation objectives:
//    1. Execution efficiency
//    2. RAM efficiency
// Validation result: Not run
//
#include "ANC_system_Implementation_v2.h"
#include <cstring>
#include <cmath>
#include "rtwtypes.h"
#include <stddef.h>
#define NumBitsPerChar                 8U

extern "C"
{
  real_T rtInf;
  real_T rtMinusInf;
  real_T rtNaN;
  real32_T rtInfF;
  real32_T rtMinusInfF;
  real32_T rtNaNF;
}

extern "C"
{
  //
  // Initialize rtNaN needed by the generated code.
  // NaN is initialized as non-signaling. Assumes IEEE.
  //
  static real_T rtGetNaN(void)
  {
    size_t bitsPerReal{ sizeof(real_T) * (NumBitsPerChar) };

    real_T nan{ 0.0 };

    if (bitsPerReal == 32U) {
      nan = rtGetNaNF();
    } else {
      union {
```

```
54          LittleEndianIEEEDouble bitVal;
55          real_T fltVal;
56        } tmpVal;
57
58        tmpVal.bitVal.words.wordH = 0xFFF80000U;
59        tmpVal.bitVal.words.wordL = 0x00000000U;
60        nan = tmpVal.fltVal;
61      }
62
63      return nan;
64    }
65
66    //
67    // Initialize rtNaNF needed by the generated code.
68    // NaN is initialized as non-signaling. Assumes IEEE.
69    //
70    static real32_T rtGetNaNF(void)
71    {
72      IEEESingle nanF{ { 0.0F } };
73
74      nanF.wordL.wordLuint = 0xFFC00000U;
75      return nanF.wordL.wordLreal;
76    }
77  }
78
79  extern "C"
80  {
81    //
82    // Initialize the rtInf, rtMinusInf, and rtNaN needed by the
83    // generated code. NaN is initialized as non-signaling. Assumes IEEE.
84    //
85    static void rt_InitInfAndNaN(size_t realSize)
86    {
87      (void) (realSize);
88      rtNaN = rtGetNaN();
89      rtNaNF = rtGetNaNF();
90      rtInf = rtGetInf();
91      rtInfF = rtGetInfF();
92      rtMinusInf = rtGetMinusInf();
93      rtMinusInfF = rtGetMinusInfF();
94    }
95
96    // Test if value is infinite
97    static boolean_T rtIsInf(real_T value)
98    {
99      return (boolean_T)((value==rtInf || value==rtMinusInf) ? 1U : 0U);
100   }
101
102   // Test if single-precision value is infinite
103   static boolean_T rtIsInfF(real32_T value)
104   {
105     return (boolean_T)(((value)==rtInfF || (value)==rtMinusInfF) ? 1U : 0U);
106   }
107
108   // Test if value is not a number
109   static boolean_T rtIsNaN(real_T value)
```

```
110   {
111     boolean_T result{ (boolean_T) 0 };
112
113     size_t bitsPerReal{ sizeof(real_T) * (NumBitsPerChar) };
114
115     if (bitsPerReal == 32U) {
116       result = rtIsNaNF((real32_T)value);
117     } else {
118       union {
119         LittleEndianIEEEDouble bitVal;
120         real_T fltVal;
121       } tmpVal;
122
123       tmpVal.fltVal = value;
124       result = (boolean_T)((tmpVal.bitVal.words.wordH & 0x7FF00000) ==
125                            0x7FF00000 &&
126                            ( (tmpVal.bitVal.words.wordH & 0x000FFFFF) != 0 ||
127                              (tmpVal.bitVal.words.wordL != 0) ));
128     }
129
130     return result;
131   }
132
133   // Test if single-precision value is not a number
134   static boolean_T rtIsNaNF(real32_T value)
135   {
136     IEEESingle tmp;
137     tmp.wordL.wordLreal = value;
138     return (boolean_T)( (tmp.wordL.wordLuint & 0x7F800000) == 0x7F800000 &&
139                         (tmp.wordL.wordLuint & 0x007FFFFF) != 0 );
140   }
141 }
142
143 extern "C"
144 {
145   //
146   // Initialize rtInf needed by the generated code.
147   // Inf is initialized as non-signaling. Assumes IEEE.
148   //
149   static real_T rtGetInf(void)
150   {
151     size_t bitsPerReal{ sizeof(real_T) * (NumBitsPerChar) };
152
153     real_T inf{ 0.0 };
154
155     if (bitsPerReal == 32U) {
156       inf = rtGetInfF();
157     } else {
158       union {
159         LittleEndianIEEEDouble bitVal;
160         real_T fltVal;
161       } tmpVal;
162
163       tmpVal.bitVal.words.wordH = 0x7FF00000U;
164       tmpVal.bitVal.words.wordL = 0x00000000U;
165       inf = tmpVal.fltVal;
```

```
166     }
167
168     return inf;
169   }
170
171   //
172   // Initialize rtInfF needed by the generated code.
173   // Inf is initialized as non-signaling. Assumes IEEE.
174   //
175   static real32_T rtGetInfF(void)
176   {
177     IEEESingle infF;
178     infF.wordL.wordLuint = 0x7F800000U;
179     return infF.wordL.wordLreal;
180   }
181
182   //
183   // Initialize rtMinusInf needed by the generated code.
184   // Inf is initialized as non-signaling. Assumes IEEE.
185   //
186   static real_T rtGetMinusInf(void)
187   {
188     size_t bitsPerReal{ sizeof(real_T) * (NumBitsPerChar) };
189
190     real_T minf{ 0.0 };
191
192     if (bitsPerReal == 32U) {
193       minf = rtGetMinusInfF();
194     } else {
195       union {
196         LittleEndianIEEEDouble bitVal;
197         real_T fltVal;
198       } tmpVal;
199
200       tmpVal.bitVal.words.wordH = 0xFFF00000U;
201       tmpVal.bitVal.words.wordL = 0x00000000U;
202       minf = tmpVal.fltVal;
203     }
204
205     return minf;
206   }
207
208   //
209   // Initialize rtMinusInfF needed by the generated code.
210   // Inf is initialized as non-signaling. Assumes IEEE.
211   //
212   static real32_T rtGetMinusInfF(void)
213   {
214     IEEESingle minfF;
215     minfF.wordL.wordLuint = 0xFF800000U;
216     return minfF.wordL.wordLreal;
217   }
218 }
219
220 // Model step function
221 void ANC_system::step()
```

```
{
  real_T rtb_DelayedWeights[64];
  real_T tmp[63];
  real_T absx;
  real_T rtb_antinoise;
  real_T zCurr;
  int32_T b_exponent;
  int32_T i;
  int32_T k;

  // Outputs for Enabled SubSystem: '<Root>/Secondary Path Est' incorporates:
  //   EnablePort: '<S2>/Enable'

  // RelationalOperator: '<Root>/run estimation' incorporates:
  //   Inport: '<Root>/Counter'
  //   Inport: '<Root>/Estimation Timer'

  if (rtU.Counter <= rtU.EstimationTimer) {
    // UnitDelay: '<S2>/Unit Delay' incorporates:
    //   Delay: '<S1>/Delay'

    std::memcpy(&rtb_DelayedWeights[0], &rtY.Weights[0], sizeof(real_T) << 6U);

    // DiscreteFir: '<S2>/Discrete FIR Filter1' incorporates:
    //   Delay: '<S1>/Delay'
    //   Inport: '<Root>/Training Noise'

    rtb_antinoise = rtU.TrainingNoise;
    for (k = 0; k < 1; k++) {
      absx = 0.0;

      // load input sample
      for (i = 0; i < 63; i++) {
        // shift state
        zCurr = rtb_antinoise;
        rtb_antinoise = rtDW.DiscreteFIRFilter1_states[i];
        rtDW.DiscreteFIRFilter1_states[i] = zCurr;

        // compute one tap
        absx += rtb_DelayedWeights[i] * zCurr;
      }

      // compute last tap
      // store output sample
      rtb_antinoise = rtb_DelayedWeights[i] * rtb_antinoise + absx;
    }

    // End of DiscreteFir: '<S2>/Discrete FIR Filter1'

    // Outport: '<Root>/Speaker output' incorporates:
    //   Gain: '<S2>/Multiply'

    rtY.Speakeroutput = -rtb_antinoise;

    // MATLABSystem: '<S2>/Est LMS Update' incorporates:
    //   Inport: '<Root>/Error Mic'
```

```
278     //   Inport: '<Root>/Estimation Step Size'
279     //   Inport: '<Root>/Training Noise'
280
281     if (rtDW.obj.LeakageFactor != 1.0) {
282       if (rtDW.obj.isInitialized == 1) {
283         rtDW.obj.TunablePropsChanged = true;
284       }
285
286       rtDW.obj.LeakageFactor = 1.0;
287     }
288
289     if (rtDW.obj.TunablePropsChanged) {
290       rtDW.obj.TunablePropsChanged = false;
291     }
292
293     std::memcpy(&tmp[0], &rtDW.obj.xBuffer[0], 63U * sizeof(real_T));
294     for (i = 0; i < 63; i++) {
295       rtDW.obj.xBuffer[(i + 2) - 1] = tmp[i];
296     }
297
298     rtDW.obj.xBuffer[0] = rtU.TrainingNoise;
299     for (i = 0; i < 64; i++) {
300       rtb_antinoise = rtU.EstimationStepSize * rtDW.obj.xBuffer[i] *
301         rtU.ErrorMic + rtDW.obj.LeakageFactor * rtDW.obj.Weights[i];
302       rtDW.obj.Weights[i] = rtb_antinoise;
303
304       // Outport: '<Root>/Weights' incorporates:
305       //   Inport: '<Root>/Error Mic'
306       //   Inport: '<Root>/Estimation Step Size'
307
308       rtY.Weights[i] = rtb_antinoise;
309     }
310
311     // End of MATLABSystem: '<S2>/Est LMS Update'
312   }
313
314   // End of RelationalOperator: '<Root>/run estimation'
315   // End of Outputs for SubSystem: '<Root>/Secondary Path Est'
316
317   // Outputs for Enabled SubSystem: '<Root>/ANC System' incorporates:
318   //   EnablePort: '<S1>/Enable'
319
320   // RelationalOperator: '<Root>/run FxLMS' incorporates:
321   //   DiscreteFir: '<S1>/Adaptive Filter'
322   //   Inport: '<Root>/Counter'
323   //   Inport: '<Root>/Estimation Timer'
324
325   if (rtU.Counter > rtU.EstimationTimer) {
326     // DiscreteFir: '<S1>/Adaptive Filter' incorporates:
327     //   Delay: '<S1>/Delay'
328     //   Inport: '<Root>/Feed Forward Mic1'
329
330     rtb_antinoise = rtU.FeedForwardMic1;
331     for (k = 0; k < 1; k++) {
332       absx = 0.0;
333
```

```
      // load input sample
      for (i = 0; i < 63; i++) {
        // shift state
        zCurr = rtb_antinoise;
        rtb_antinoise = rtDW.AdaptiveFilter_states[i];
        rtDW.AdaptiveFilter_states[i] = zCurr;

        // compute one tap
        absx += rtDW.Delay_DSTATE[i] * zCurr;
      }

      // compute last tap
      // store output sample
      rtb_antinoise = rtDW.Delay_DSTATE[i] * rtb_antinoise + absx;
    }

    k = 1;

    // Outport: '<Root>/Speaker Output1' incorporates:
    //   DiscreteFir: '<S1>/Adaptive Filter'
    //   Gain: '<S1>/Multiply'

    rtY.SpeakerOutput1 = -rtb_antinoise;

    // DiscreteFir: '<S1>/Estimated Secondary path' incorporates:
    //   Inport: '<Root>/Feed Forward Mic1'
    //   MATLABSystem: '<S2>/Est LMS Update'
    //   Outport: '<Root>/Weights'

    rtb_antinoise = rtU.FeedForwardMic1 * rtY.Weights[0];
    for (i = rtDW.EstimatedSecondarypath_circBuf; i < 63; i++) {
      rtb_antinoise += rtDW.EstimatedSecondarypath_states[i] * rtY.Weights[k];
      k++;
    }

    for (i = 0; i < rtDW.EstimatedSecondarypath_circBuf; i++) {
      rtb_antinoise += rtDW.EstimatedSecondarypath_states[i] * rtY.Weights[k];
      k++;
    }

    // End of DiscreteFir: '<S1>/Estimated Secondary path'

    // MATLABSystem: '<S1>/ANC LMS Update' incorporates:
    //   Inport: '<Root>/ANC Step Size'
    //   Inport: '<Root>/Error mic1'

    if (rtDW.obj_p.LeakageFactor != 1.0) {
      if (rtDW.obj_p.isInitialized == 1) {
        rtDW.obj_p.TunablePropsChanged = true;
      }

      rtDW.obj_p.LeakageFactor = 1.0;
    }

    if (rtDW.obj_p.TunablePropsChanged) {
      rtDW.obj_p.TunablePropsChanged = false;
```

```
    }

    std::memcpy(&tmp[0], &rtDW.obj_p.xBuffer[0], 63U * sizeof(real_T));
    for (i = 0; i < 63; i++) {
      rtDW.obj_p.xBuffer[(i + 2) - 1] = tmp[i];
    }

    rtDW.obj_p.xBuffer[0] = rtb_antinoise;
    for (i = 0; i < 64; i++) {
      absx = rtDW.obj_p.xBuffer[i];
      rtb_DelayedWeights[i] = absx * absx;
    }

    rtb_antinoise = rtb_DelayedWeights[0];
    for (k = 0; k < 63; k++) {
      rtb_antinoise += rtb_DelayedWeights[k + 1];
    }

    absx = std::abs(rtDW.obj_p.xBuffer[0]);
    if (std::isinf(absx) || std::isnan(absx)) {
      absx = (rtNaN);
    } else if (absx < 4.4501477170144028E-308) {
      absx = 4.94065645841247E-324;
    } else {
      std::frexp(absx, &b_exponent);
      absx = std::ldexp(1.0, b_exponent - 53);
    }

    rtb_antinoise += absx;
    for (i = 0; i < 64; i++) {
      absx = rtU.ANCStepSize * rtDW.obj_p.xBuffer[i] * rtU.Errormic1 /
        rtb_antinoise + rtDW.obj_p.LeakageFactor * rtDW.obj_p.Weights[i];
      rtDW.obj_p.Weights[i] = absx;

      // Update for Delay: '<S1>/Delay' incorporates:
      //   Inport: '<Root>/ANC Step Size'
      //   Inport: '<Root>/Error mic1'

      rtDW.Delay_DSTATE[i] = absx;
    }

    // End of MATLABSystem: '<S1>/ANC LMS Update'

    // Update for DiscreteFir: '<S1>/Estimated Secondary path' incorporates:
    //   Inport: '<Root>/Feed Forward Mic1'

    // Update circular buffer index
    rtDW.EstimatedSecondarypath_circBuf--;
    if (rtDW.EstimatedSecondarypath_circBuf < 0) {
      rtDW.EstimatedSecondarypath_circBuf = 62;
    }

    // Update circular buffer
    rtDW.EstimatedSecondarypath_states[rtDW.EstimatedSecondarypath_circBuf] =
      rtU.FeedForwardMic1;

```

```
    // End of Update for DiscreteFir: '<S1>/Estimated Secondary path'
  }

  // End of RelationalOperator: '<Root>/run FxLMS'
  // End of Outputs for SubSystem: '<Root>/ANC System'
}

// Model initialize function
void ANC_system::initialize()
{
  // Registration code

  // initialize non-finites
  rt_InitInfAndNaN(sizeof(real_T));

  // SystemInitialize for Enabled SubSystem: '<Root>/Secondary Path Est'
  // Start for MATLABSystem: '<S2>/Est LMS Update'
  rtDW.obj.LeakageFactor = 1.0;
  rtDW.obj.isInitialized = 1;
  rtDW.obj.TunablePropsChanged = false;

  // End of SystemInitialize for SubSystem: '<Root>/Secondary Path Est'

  // SystemInitialize for Enabled SubSystem: '<Root>/ANC System'
  // Start for MATLABSystem: '<S1>/ANC LMS Update'
  rtDW.obj_p.LeakageFactor = 1.0;
  rtDW.obj_p.isInitialized = 1;
  rtDW.obj_p.TunablePropsChanged = false;

  // End of SystemInitialize for SubSystem: '<Root>/ANC System'

  // SystemInitialize for Enabled SubSystem: '<Root>/Secondary Path Est'
  // InitializeConditions for MATLABSystem: '<S2>/Est LMS Update'
  std::memset(&rtDW.obj.xBuffer[0], 0, sizeof(real_T) << 6U);
  std::memset(&rtDW.obj.Weights[0], 0, sizeof(real_T) << 6U);

  // End of SystemInitialize for SubSystem: '<Root>/Secondary Path Est'

  // SystemInitialize for Enabled SubSystem: '<Root>/ANC System'
  // InitializeConditions for MATLABSystem: '<S1>/ANC LMS Update'
  std::memset(&rtDW.obj_p.xBuffer[0], 0, sizeof(real_T) << 6U);
  std::memset(&rtDW.obj_p.Weights[0], 0, sizeof(real_T) << 6U);

  // End of SystemInitialize for SubSystem: '<Root>/ANC System'
}

// Constructor
ANC_system::ANC_system():
  rtU(),
  rtY(),
  rtDW()
{
  // Currently there is no constructor body generated.
}

// Destructor
```

```
// Currently there is no destructor body generated.
ANC_system::~ANC_system() = default;

//
// File trailer for generated code.
//
// [EOF]
//
```

4) ANC Block Processor

Listing 4. ANC Block Processor.h

```
#ifndef ANC_BLOCK_PROCESSOR_H
#define ANC_BLOCK_PROCESSOR_H

#include <AudioStream.h>
#include <Audio.h>
#include "ANC_system_Implementation_v2.h"
#include <cmath>

#define ESTIMATION_TIME_SEC 30.0f
#define SAMPLE_RATE 44100.0f
#define LOWPASS_CUTOFF_HZ 1000.0f
#define Estimation_step_size 0.05f
#define ANC_step_size 0.0005f

enum ANCMode {
  MODE_OFF,
  MODE_ON
};

class ANCBlockProcessor : public AudioStream {
public:
  ANCBlockProcessor()
    : AudioStream(2, inputQueueArray) {
    anc.initialize();
    startTime = millis();
    hasSwitched = false;
    currentMode = MODE_ON;
    volume = 1.0f;

    anc.rtU.EstimationStepSize = Estimation_step_size;
    anc.rtU.ANCStepSize = ANC_step_size;
    anc.rtU.EstimationTimer = ESTIMATION_TIME_SEC;

    float RC1000 = 1.0f / (2.0f * 3.14159f * LOWPASS_CUTOFF_HZ);
    lowpassAlpha = 1.0f / (1.0f + RC1000 * SAMPLE_RATE);

    float RC300 = 1.0f / (2.0f * 3.14159f * 300.0f);
    lowpassAlpha300Hz = 1.0f / (1.0f + RC300 * SAMPLE_RATE);

    errorMicFiltered = 0.0f;
    ffLowFiltered = 0.0f;
  }

  void setMode(ANCMode mode) {
    currentMode = mode;
  }

  void setVolume(float vol) {
    volume = vol;
  }

  virtual void update() {
    audio_block_t *inFF = receiveReadOnly(0);
```

```
54      audio_block_t *inErr = receiveReadOnly(1);
55      if (!inFF || !inErr) {
56        if (inFF) release(inFF);
57        if (inErr) release(inErr);
58        return;
59      }
60
61      audio_block_t *outBlock = allocate();
62      if (!outBlock) {
63        release(inFF);
64        release(inErr);
65        return;
66      }
67
68      float timeSec = (millis() - startTime) / 1000.0f;
69      float ffMic, errMic, outSample = 0.0f;
70
71      float lowEnergy = 0.0f;
72      float highEnergy = 0.0f;
73
74      for (int i = 0; i < AUDIO_BLOCK_SAMPLES; i++) {
75        ffMic = (float)inFF->data[i] / 32768.0f;
76        errMic = (float)inErr->data[i] / 32768.0f;
77
78        // Low-pass filter on error mic
79        errorMicFiltered = lowpassAlpha * errMic + (1.0f - lowpassAlpha) *
              errorMicFiltered;
80
81        // Filterbank: Low-pass and high-pass on ffMic
82        ffLowFiltered = lowpassAlpha300Hz * ffMic + (1.0f - lowpassAlpha300Hz) *
              ffLowFiltered;
83        float ffHighFiltered = ffMic - ffLowFiltered;
84
85        // Energy accumulation
86        lowEnergy += ffLowFiltered * ffLowFiltered;
87        highEnergy += ffHighFiltered * ffHighFiltered;
88
89        anc.rtU.Counter = timeSec;
90
91        if (timeSec < ESTIMATION_TIME_SEC) {
92          // anc.rtU.EstimationStepSize = 0.1f / (1.0f + 0.05f * timeSec);
93          // anc.rtU.EstimationStepSize = max(anc.rtU.EstimationStepSize, 0.001f);
94
95          float noiseAmplitude = 0.5f; // Tune as needed
96          anc.rtU.TrainingNoise = noiseAmplitude * random(-1000, 1000) / 1000.0f;
97
98          anc.rtU.ErrorMic = errorMicFiltered;
99          anc.step();
100         outSample = anc.rtY.Speakeroutput;
101       } else {
102         anc.rtU.TrainingNoise = 0.0f;
103         anc.rtU.FeedForwardMic1 = ffMic;
104         anc.rtU.Errormic1 = errorMicFiltered;
105         anc.rtU.ErrorMic = errorMicFiltered;
106
107         if (!hasSwitched) {
```

```
108           hasSwitched = true;
109           Serial.println(">> Estimation time complete. FxLMS ready.");
110           Serial.println("Final estimated secondary path weights:");
111           for (int j = 0; j < 64; j++) {
112             Serial.print("W[");
113             Serial.print(j);
114             Serial.print("] = ");
115             Serial.println(anc.rtY.Weights[j], 6);
116           }
117         }
118
119         if (currentMode == MODE_ON) {
120           // Apply scene-based switching using energy ratio
121           float energyRatio = lowEnergy / (highEnergy + 1e-6f); // avoid divide-by-
                zero
122           bool allowAdaptation = (energyRatio > 2.0f); // Tune this threshold
123
124           anc.rtU.ANCStepSize = allowAdaptation ? ANC_step_size : 0.0f;
125           anc.step();
126           outSample = anc.rtY.SpeakerOutput1;
127         } else {
128           outSample = 0.0f;
129         }
130       }
131
132       outSample = constrain(outSample, -1.0f, 1.0f);
133       outBlock->data[i] = (int16_t)(outSample * 32767.0f);
134     }
135
136     transmit(outBlock, 0);
137     release(outBlock);
138     release(inFF);
139     release(inErr);
140   }
141
142 private:
143   ANC_system anc;
144   audio_block_t *inputQueueArray[2];
145   unsigned long startTime;
146   bool hasSwitched;
147   ANCMode currentMode;
148   float volume;
149
150   // Filter states
151   float errorMicFiltered;
152   float lowpassAlpha;
153   float lowpassAlpha300Hz;
154   float ffLowFiltered;
155 };
156
157 #endif
```

5) rtwtypes

Listing 5. rtwtypes.h

```
//
// Academic License - for use in teaching, academic research, and meeting
// course requirements at degree granting institutions only.  Not for
// government, commercial, or other organizational use.
//
// File: rtwtypes.h
//
// Code generated for Simulink model 'ANC_system_Implementation_v2'.
//
// Model version                  : 1.34
// Simulink Coder version         : 9.9 (R2023a) 19-Nov-2022
// C/C++ source code generated on : Tue Jun  3 09:36:17 2025
//
// Target selection: ert.tlc
// Embedded hardware selection: ARM Compatible->ARM Cortex-M
// Code generation objectives:
//    1. Execution efficiency
//    2. RAM efficiency
// Validation result: Not run
//

#ifndef RTWTYPES_H
#define RTWTYPES_H

// Logical type definitions
#if (!defined(__cplusplus))
#ifndef false
#define false                          (0U)
#endif

#ifndef true
#define true                           (1U)
#endif
#endif

//=======================================================================*
//  Target hardware information
//    Device type: ARM Compatible->ARM Cortex-M
//    Number of bits:     char:   8    short:   16    int:  32
//                        long:  32    long long:  64
//                        native word size:  32
//    Byte ordering: LittleEndian
//    Signed integer division rounds to: Zero
//    Shift right on a signed integer as arithmetic shift: on
// =======================================================================

//=======================================================================*
//  Fixed width word size data types:                                     *
//    int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers     *
//    uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers   *
//    real32_T, real64_T           - 32 and 64 bit floating point numbers *
// =======================================================================
typedef signed char int8_T;
```

```
54 typedef unsigned char uint8_T;
55 typedef short int16_T;
56 typedef unsigned short uint16_T;
57 typedef int int32_T;
58 typedef unsigned int uint32_T;
59 typedef long long int64_T;
60 typedef unsigned long long uint64_T;
61 typedef float real32_T;
62 typedef double real64_T;
63
64 //===========================================================================*
65 //  Generic type definitions: boolean_T, char_T, byte_T, int_T, uint_T,       *
66 //                            real_T, time_T, ulong_T, ulonglong_T.           *
67 // ===========================================================================
68 typedef double real_T;
69 typedef double time_T;
70 typedef unsigned char boolean_T;
71 typedef int int_T;
72 typedef unsigned int uint_T;
73 typedef unsigned long ulong_T;
74 typedef unsigned long long ulonglong_T;
75 typedef char char_T;
76 typedef unsigned char uchar_T;
77 typedef char_T byte_T;
78
79 //===========================================================================*
80 //  Min and Max:                                                             *
81 //    int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers        *
82 //    uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers      *
83 // ===========================================================================
84 #define MAX_int8_T                     ((int8_T)(127))
85 #define MIN_int8_T                     ((int8_T)(-128))
86 #define MAX_uint8_T                    ((uint8_T)(255U))
87 #define MAX_int16_T                    ((int16_T)(32767))
88 #define MIN_int16_T                    ((int16_T)(-32768))
89 #define MAX_uint16_T                   ((uint16_T)(65535U))
90 #define MAX_int32_T                    ((int32_T)(2147483647))
91 #define MIN_int32_T                    ((int32_T)(-2147483647-1))
92 #define MAX_uint32_T                   ((uint32_T)(0xFFFFFFFFU))
93 #define MAX_int64_T                    ((int64_T)(9223372036854775807LL))
94 #define MIN_int64_T                    ((int64_T)(-9223372036854775807LL-1LL))
95 #define MAX_uint64_T                   ((uint64_T)(0xFFFFFFFFFFFFFFFFULL))
96
97 // Block D-Work pointer type
98 typedef void * pointer_T;
99
100 #endif                                 // RTWTYPES_H
101
102 //
103 // File trailer for generated code.
104 //
105 // [EOF]
106 //
```

6) Secondary path Characterization program

Listing 6. ANCCharacterization.ino

```
#include <Audio.h>
#include <Wire.h>
#include <SparkFun_WM8960_Arduino_Library.h>


// --- Constants ---
const int stabilization_time = 300;  // in milliseconds
const int min_freq = 20;
const int max_freq = 21000;
const float amplitude = 1;
const int step_per_octave = 20;  // log scale steps

WM8960 codec;

// --- Audio Components ---
AudioSynthWaveform waveform;  // Sine wave generator
AudioInputTDM2     tdm2;
AudioAnalyzeRMS    rmsTest;

AudioOutputI2S i2s1;           // Output to codec

// --- Audio Connections ---
AudioConnection patchOut(waveform, 0, i2s1, 0);   // Sine to L channel
AudioConnection patchOut2(waveform, 0, i2s1, 1);  // Sine to R channel
AudioConnection patchTest(tdm2, 2, rmsTest, 0);

void setup() {
  Serial.begin(115200);
  Wire.begin();
  delay(100);

  if (!codec.begin()) {
    Serial.println("Codec␣not␣detected.");
    while (1)
      ;
  }

  // --- WM8960 Configuration ---
  codec.enableVREF();
  codec.enableVMID();
  codec.connectLMN1();  // Route mic input
  codec.enableLMIC();
  codec.disableLINMUTE();
  codec.setLMICBOOST(WM8960_MIC_BOOST_GAIN_0DB);
  codec.connectLMIC2B();
  codec.enableAINL();
  codec.enableLD2LO();
  codec.setLB2LOVOL(WM8960_OUTPUT_MIXER_GAIN_0DB);
  codec.enableLOMIX();
  codec.enableHeadphones();
  codec.enableOUT3MIX();

  codec.enablePLL();
```

```
54    codec.setPLLPRESCALE(WM8960_PLLPRESCALE_DIV_2);
55    codec.setSMD(WM8960_PLL_MODE_FRACTIONAL);
56    codec.setCLKSEL(WM8960_CLKSEL_PLL);
57    codec.setSYSCLKDIV(WM8960_SYSCLK_DIV_BY_2);
58    codec.setBCLKDIV(4);
59    codec.setDCLKDIV(WM8960_DCLKDIV_16);
60    codec.setPLLN(7);
61    codec.setPLLK(0x86, 0xC2, 0x26);
62    codec.enablePeripheralMode();
63
64    codec.enableAdcLeft();
65    codec.enableDacLeft();
66    codec.disableDacMute();
67    codec.setHeadphoneVolumeDB(0.0);
68
69    AudioMemory(60);
70
71    delay(100);
72
73    waveform.begin(WAVEFORM_SINE);
74    waveform.amplitude(amplitude);
75    waveform.frequency(min_freq);
76  }
77
78  void loop() {
79    Serial.println("freq, avg_rms");
80
81    for (float freq = min_freq; freq <= max_freq; freq *= pow(2.0, 1.0 /
          step_per_octave)) {
82      // Serial.print("Testing frequency: ");
83      // Serial.println(freq);
84
85      waveform.frequency(freq);
86      delay(stabilization_time); // Allow output and microphones to stabilize
87
88      float sum = 0.0;
89      int count = 0;
90
91      while (count < 20) {
92        if (rmsTest.available()) {
93          float val = rmsTest.read();
94          // Serial.println(val, 8);
95          sum += val;
96          count++;
97          delay(20);
98        }
99      }
100     float avgRMS = sum / 5.0;
101
102     Serial.print(freq, 2);
103     Serial.print(", ");
104     Serial.println(avgRMS, 6);
105   }
106
107   while (true) {
108     // Stop after sweep (remove this to loop)
```

```
109         waveform.amplitude(0);
110     }
111 }
```

7) Cancellation Data Function

```
% Passive and Active Cancellation Graphs
% Averaging

cust_pass_avg = (AbsoluteAmplitudeCustomPassive4 + ...
                 AbsoluteAmplitudeCustomPassive3 + ...
                 AbsoluteAmplitudeCustomPassive2 + ...
                 AbsoluteAmplitudeCustomPassive1) / 4;

cust_act_avg = (AbsoluteAmplitudeCustomActive2 + ...
                AbsoluteAmplitudeCustomActive1) / 2;



% Defining Vars
freq_range = Frequency_Hz_;
open_ear_norm = AbsoluteAmplitudeNorm ./ AbsoluteAmplitudeNorm;
cust_pass_norm = (cust_pass_avg - AbsoluteAmplitudeNorm )./ AbsoluteAmplitudeNorm;
cust_act_norm = (cust_act_avg - AbsoluteAmplitudeNorm) ./ AbsoluteAmplitudeNorm;
bose_pass_norm = (AbsoluteAmplitudeBosePassiveCancellation - AbsoluteAmplitudeNorm) ./ Absolute
bose_act_norm = (AbsoluteAmplitudeBoseActiveCancellation - AbsoluteAmplitudeNorm) ./ AbsoluteA

% Convert to dB
open_ear_dB = 20 * log10(abs(open_ear_norm));
cust_pass_dB = 20 * log10(abs(cust_pass_norm));
cust_act_dB = 20 * log10(abs(cust_act_norm));
bose_pass_dB = 20 * log10(abs(bose_pass_norm));
bose_act_dB = 20 * log10(abs(bose_act_norm));

% Smoothing window size (adjust as needed: 5 21   typical)
windowSize = 15;

% Apply smoothing to dB curves
cust_pass_dB_smooth = movmean(cust_pass_dB, windowSize);
cust_act_dB_smooth  = movmean(cust_act_dB,  windowSize);
bose_pass_dB_smooth = movmean(bose_pass_dB, windowSize);
bose_act_dB_smooth  = movmean(bose_act_dB,  windowSize);

% Define frequency masks by index
idx_300_500 = find(freq_range >= 300 & freq_range <= 500);
idx_501_up  = find(freq_range > 500);

% Compute mean attenuation (already in dB)
avg_act_300_500 = mean(cust_act_dB(idx_300_500));
avg_pass_501up  = mean(cust_pass_dB(idx_501_up));

% Display results
fprintf('Custom Active (300  500  Hz): %.2f dB average attenuation\n', avg_act_300_500);
fprintf('Custom Passive (>500 Hz): %.2f dB average attenuation\n', avg_pass_501up);
```

```
% Create Figure with Two Subplots
figure;

% --- Plot 1: Passive Comparison ---
subplot(2,1,1);
semilogx(freq_range, open_ear_dB, 'k-', ...
         freq_range, cust_pass_dB_smooth, 'b-', ...
         freq_range, bose_pass_dB_smooth, 'r-', ...
         'LineWidth', 1.5);
title('Frequency Response: Passive Cancellation vs. Open Ear');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Open Ear', 'Custom Passive', 'Bose Passive', 'Location', 'SouthWest');
xlim([24 20000]);
grid on;

% --- Plot 2: Active Comparison ---
subplot(2,1,2);
semilogx(freq_range, open_ear_dB, 'k', ...
         freq_range, cust_act_dB_smooth, 'b', ...
         freq_range, bose_act_dB_smooth, 'r', ...
         'LineWidth', 1.5);
title('Frequency Response: Active Cancellation vs. Open Ear');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Open Ear', 'Custom Active', 'Bose Active', 'Location', 'SouthWest');
xlim([24 20000]);
grid on;
```

XI-J Other Documentation

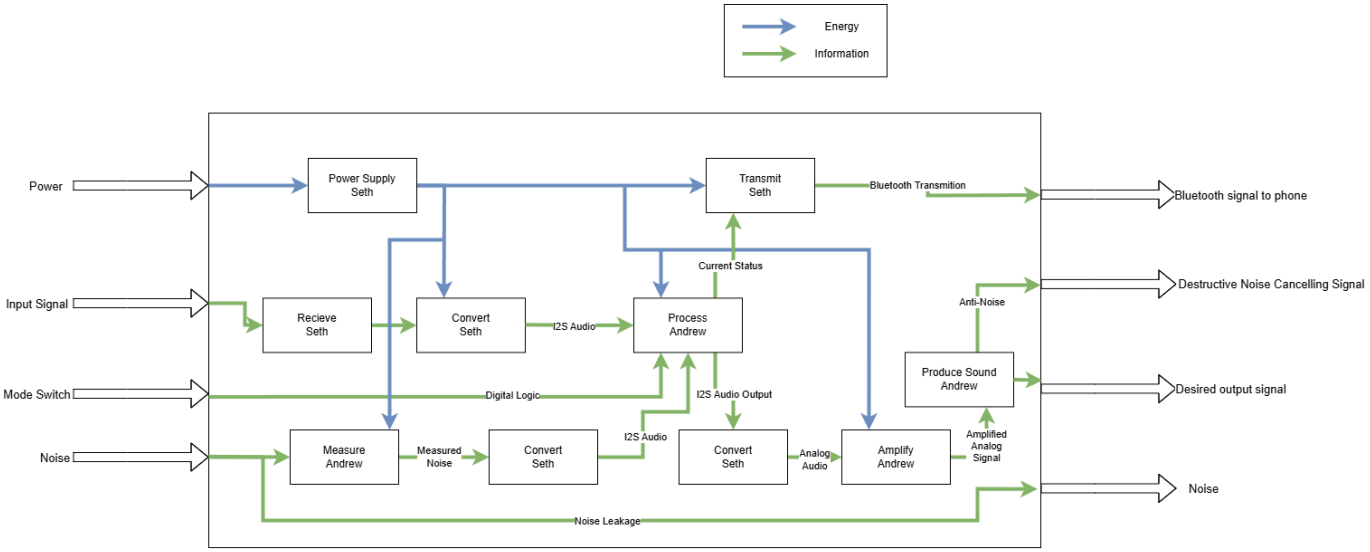


Fig. 24. Level 1 block diagram

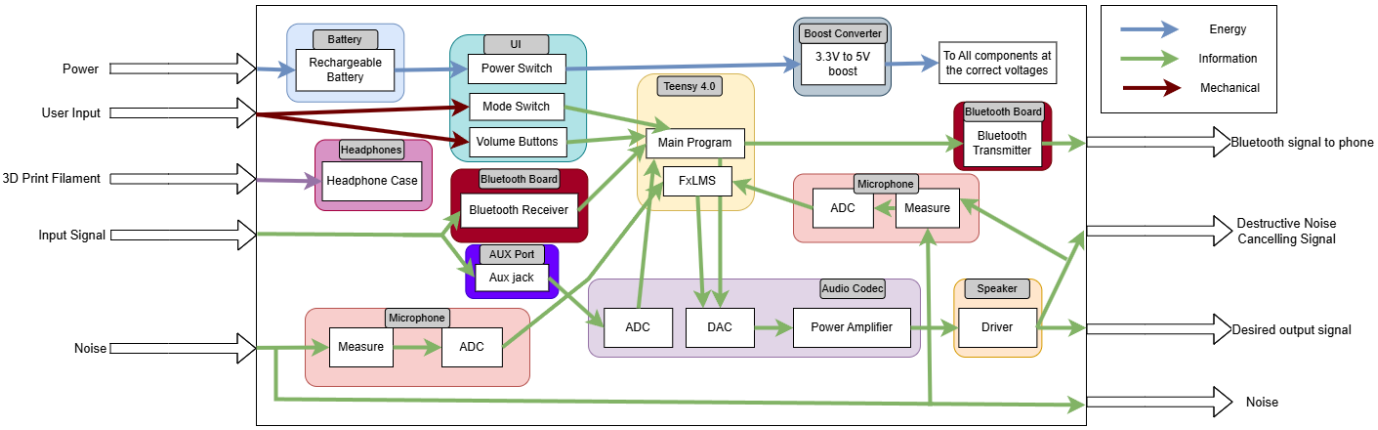


Fig. 25. Implementation Concept Block Diagram