

# MISTER MIXER

## Group 14



Srinilai Kankipati, Computer Engineering  
Kevin Kurian, Computer Engineering  
Michael Meyers, Electrical Engineering  
Juni Yeom, Computer Science  
Joshua Yu, Computer Engineering

### **Board Committee Members**

John Aedo  
Mark Maddox  
Saleem Sahawneh  
Enxia Zhang

November 10, 2024

# Contents

|          |                                                                              |           |
|----------|------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Executive Summary</b>                                                     | <b>1</b>  |
| <b>2</b> | <b>Project Description</b>                                                   | <b>2</b>  |
| 2.1      | Motivation and Background . . . . .                                          | 2         |
| 2.2      | Existing Product/Past Project/Prior Related Work . . . . .                   | 2         |
| 2.2.1    | USB Audio Interfaces . . . . .                                               | 5         |
| 2.3      | Goals and Objectives . . . . .                                               | 6         |
| 2.3.1    | Basic Goals . . . . .                                                        | 6         |
| 2.3.2    | Stretch Goals . . . . .                                                      | 7         |
| 2.3.3    | Objectives . . . . .                                                         | 7         |
| 2.4      | Key Engineering Specifications . . . . .                                     | 8         |
| 2.5      | House of Quality . . . . .                                                   | 8         |
| 2.6      | Hardware Block Diagram . . . . .                                             | 9         |
| 2.7      | Software Architecture Diagram . . . . .                                      | 11        |
| <b>3</b> | <b>Research</b>                                                              | <b>12</b> |
| 3.1      | Hardware Research and Comparisons . . . . .                                  | 13        |
| 3.1.1    | Audio Signals . . . . .                                                      | 13        |
| 3.1.2    | Audio Connectors . . . . .                                                   | 14        |
| 3.1.3    | Preamplifiers . . . . .                                                      | 15        |
| 3.1.4    | Signal Converters (A/D & D/A) . . . . .                                      | 18        |
| 3.1.5    | Choosing The Processor Type: SBCs, MCUs, FPGAs . . . . .                     | 24        |
| 3.1.6    | Choosing the Single-Board Computer . . . . .                                 | 26        |
| 3.1.7    | Choosing the Microcontroller . . . . .                                       | 28        |
| 3.1.8    | Clock . . . . .                                                              | 32        |
| 3.1.9    | Flash Memory . . . . .                                                       | 32        |
| 3.1.10   | Power Supply . . . . .                                                       | 32        |
| 3.2      | Software Research and Comparison . . . . .                                   | 38        |
| 3.2.1    | Audio DSP Effects . . . . .                                                  | 38        |
| 3.2.2    | FFT Signals: Importance and Application . . . . .                            | 40        |
| 3.2.3    | Comparison of Communication Protocols . . . . .                              | 41        |
| 3.2.4    | Analysis of Protocols for Raspberry Pi to Mobile Data Transmission . . . . . | 43        |
| 3.2.5    | Wireless Data Level Protocols . . . . .                                      | 49        |
| 3.2.6    | Real-Time Processing Considerations . . . . .                                | 52        |
| 3.2.7    | User Interface Frameworks, Comparison and Selection . . . . .                | 54        |

|          |                                                           |            |
|----------|-----------------------------------------------------------|------------|
| 3.2.8    | Automixing Methods, Comparison, and Selection . . . . .   | 57         |
| <b>4</b> | <b>Standards and Design Constraints</b>                   | <b>72</b>  |
| 4.1      | Standards . . . . .                                       | 73         |
| 4.2      | Design Constraints . . . . .                              | 75         |
| 4.2.1    | Economic & Time Constraints . . . . .                     | 76         |
| 4.2.2    | Hardware Constraints . . . . .                            | 77         |
| 4.2.3    | Datasets . . . . .                                        | 78         |
| 4.2.4    | Social . . . . .                                          | 78         |
| <b>5</b> | <b>Comparison of ChatGPT with Other Similar Platforms</b> | <b>78</b>  |
| 5.1      | Limitations, Pros, and Cons . . . . .                     | 79         |
| 5.2      | Examples . . . . .                                        | 81         |
| 5.2.1    | Automation of Boilerplate Tasks . . . . .                 | 81         |
| 5.2.2    | Information Retrieval . . . . .                           | 82         |
| 5.3      | Conclusion . . . . .                                      | 82         |
| <b>6</b> | <b>Hardware Design</b>                                    | <b>83</b>  |
| 6.1      | Subsystem Block Diagram . . . . .                         | 83         |
| 6.1.1    | Preamplification Circuitry . . . . .                      | 83         |
| 6.1.2    | LDO Circuitry . . . . .                                   | 85         |
| 6.1.3    | Buck Converters . . . . .                                 | 86         |
| 6.1.4    | Phantom Power Circuitry . . . . .                         | 91         |
| 6.1.5    | ADC Circuitry . . . . .                                   | 92         |
| 6.1.6    | DAC Circuitry . . . . .                                   | 93         |
| 6.1.7    | Headphone Amplifier . . . . .                             | 95         |
| 6.1.8    | Proper Grounding . . . . .                                | 96         |
| <b>7</b> | <b>Software Design</b>                                    | <b>98</b>  |
| 7.1      | Software Requirements and Constraints . . . . .           | 98         |
| 7.1.1    | User Stories and Acceptance Criteria . . . . .            | 98         |
| 7.1.2    | Unified Modeling Language (UML) Diagrams . . . . .        | 100        |
| 7.1.3    | Use-Case Diagram . . . . .                                | 100        |
| 7.1.4    | User Roles . . . . .                                      | 100        |
| 7.1.5    | Use-Cases Correspond to Software Modules . . . . .        | 102        |
| 7.1.6    | Constraints and Implications for Software . . . . .       | 102        |
| 7.2      | Development of Software Architecture . . . . .            | 102        |
| 7.2.1    | System Topology and Deployment . . . . .                  | 103        |
| 7.2.2    | Managing Data Flows . . . . .                             | 104        |
| 7.3      | Implementation and Concrete Design . . . . .              | 106        |
| 7.3.1    | Audio Transfer . . . . .                                  | 106        |
| <b>8</b> | <b>Administrative Content</b>                             | <b>109</b> |
| 8.1      | Budget Estimates . . . . .                                | 109        |
| 8.2      | SD1 Documentation timeline . . . . .                      | 110        |

|                                   |            |
|-----------------------------------|------------|
| 8.3 SD2 Design Timeline . . . . . | 111        |
| 8.4 Work Distributions . . . . .  | 111        |
| <b>A Citations</b>                | <b>113</b> |
| <b>B Copyright</b>                | <b>118</b> |
| <b>C LLM Citations</b>            | <b>121</b> |

# List of Tables

|                                                                                                          |     |
|----------------------------------------------------------------------------------------------------------|-----|
| 2.1 Table of Proposed Design Specifications . . . . .                                                    | 8   |
| 3.1 Audio Connector Comparison . . . . .                                                                 | 15  |
| 3.2 Standards for Phantom Power: IEC 61938:2018 . . . . .                                                | 17  |
| 3.3 Comparison of Preamp ICs . . . . .                                                                   | 18  |
| 3.4 ADC Comparison Chart . . . . .                                                                       | 21  |
| 3.5 DAC Comparison Chart . . . . .                                                                       | 23  |
| 3.6 Headphone Amplifier Comparison Table . . . . .                                                       | 24  |
| 3.7 Processor Comparison Table . . . . .                                                                 | 26  |
| 3.8 Single-Board Computer Comparison . . . . .                                                           | 28  |
| 3.9 Microcontroller Comparison . . . . .                                                                 | 32  |
| 3.10 LDO Comparison Chart . . . . .                                                                      | 36  |
| 3.11 Comparison Table of Linear vs. Switching Regulators . . . . .                                       | 37  |
| 3.12 Buck Converter Comparison Table . . . . .                                                           | 38  |
| 3.13 Table of Communication Protocols . . . . .                                                          | 48  |
| 3.14 Comparison of Flutter, Slint, and Qt . . . . .                                                      | 55  |
| 3.15 Summary of Classical Machine Learning Methods for Automixing . . . . .                              | 62  |
| 3.16 Comparison of Deep Learning and Classical Machine Learning for Automixing . . . . .                 | 64  |
| 3.17 Comparison of Automatic Mixing Methods . . . . .                                                    | 71  |
| 3.18 Comparison of Various Multitrack Audio Datasets for Automixing Courtesy of Steinmetz et al. . . . . | 72  |
| 4.5 Identified Constraints . . . . .                                                                     | 76  |
| 5.1 Comparison of ChatGPT, GitHub Copilot, and Google Gemini . . . . .                                   | 81  |
| 8.1 Estimated Cost of Production . . . . .                                                               | 110 |
| 8.2 Project Documentation Milestone . . . . .                                                            | 111 |
| 8.3 Project Design Milestone . . . . .                                                                   | 111 |
| 8.4 Duty Distribution . . . . .                                                                          | 112 |

# List of Figures

|      |                                                                                                                                                                                                            |    |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1  | Behringer Flow 8 - Compact 8-Channel Mixer.<br><b>MSRP \$299.99</b>                                                                                                                                        | 3  |
| 2.2  | Mackie MobileMix - Compact 8-Channel Mixer.<br><b>MSRP \$229.99</b>                                                                                                                                        | 3  |
| 2.3  | Allen & Heath CQ-18T - Compact 16-Channel Digital Mixer with Wi-Fi.<br><b>MSRP \$1,199.99</b>                                                                                                              | 4  |
| 2.4  | QSC Touchmix-8 - 14-Channel Digital Mixer. <b>MSRP \$1,149.99</b>                                                                                                                                          | 4  |
| 2.5  | Tascam Model 16<br>5                                                                                                                                                                                       |    |
| 2.6  | Soundcraft Signature 22 MTK<br>6                                                                                                                                                                           |    |
| 2.7  | House of Quality Diagram                                                                                                                                                                                   | 9  |
| 2.8  | Proposed High-level Hardware Block Diagram                                                                                                                                                                 | 10 |
| 2.9  | High-Level Software Architecture Block Diagram                                                                                                                                                             | 12 |
| 3.1  | TR and TRS Wiring Key                                                                                                                                                                                      | 14 |
| 3.2  | XLR to TRS Connection Diagram                                                                                                                                                                              | 15 |
| 3.3  | High-level Phantom Power Schematic, courtesy of Sweetwater                                                                                                                                                 | 17 |
| 3.4  | Example of Aliasing in Triangle Wave, created by Fraïssé and licensed under CC BY-SA 3.0 ( <a href="https://creativecommons.org/licenses/by-sa/3.0/">https://creativecommons.org/licenses/by-sa/3.0/</a> ) | 19 |
| 3.5  | Architechure of N-bit SAR ADC, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.                                                              | 20 |
| 3.6  | 4-bit SAR ADC example operation, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.                                                            | 20 |
| 3.7  | Binary Weighted DAC Example Circuit                                                                                                                                                                        | 22 |
| 3.8  | High Level Flow of Sigma-Delta DAC, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.                                                         | 22 |
| 3.9  | Basic Linear Regulator, Reproduced with Permission. © Renesas Electronics Corporation or one of its subsidiaries. All Rights Reserved.                                                                     | 35 |
| 3.10 | Example Application of TI’s Linear Regulators, Courtesy of Texas Instruments Inc                                                                                                                           | 35 |

|                                                                                                                                                               |     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.11 Example Switching Regulator Schematic, Reproduced with Permission. © Renesas Electronics Corporation or one of its subsidiaries.<br>All Rights Reserved. | 36  |
| 3.12 The Addition Operator                                                                                                                                    | 39  |
| 3.13 The Multiplication Operator                                                                                                                              | 39  |
| 3.14 The Delay Operator                                                                                                                                       | 39  |
| 3.15 Fast Fourier Transform Algorithm                                                                                                                         | 40  |
| 3.16 WebSocket Connection Steps                                                                                                                               | 44  |
| 3.17 HTTP/REST API Connection Steps                                                                                                                           | 45  |
| 3.18 MQTT Connection Steps                                                                                                                                    | 46  |
| 3.19 CoAP Connection Steps                                                                                                                                    | 47  |
| 3.20 Wi-Fi Routing Explanation                                                                                                                                | 50  |
| 3.21 BLE Connection Steps                                                                                                                                     | 51  |
| 3.22 Flow diagram of an adaptive audio effect                                                                                                                 | 58  |
| 3.23 Architecture of [1] applying Reverb. Feature Extraction and Model training Training are Decoupled.                                                       | 63  |
| 3.24 Visualization of Direct Transformation courtesy of Steinmetz et al.                                                                                      | 68  |
| 3.25 Visualization of Parameter Estimation with Audio Loss courtesy of Steinmetz et al.                                                                       | 69  |
| 5.1 Prompt and Output from ChatGPT 4o                                                                                                                         | 80  |
| 6.1 Schematic of Microphone/Line Preamplifier                                                                                                                 | 84  |
| 6.2 Schematic of the three LDOs used in power circuitry                                                                                                       | 85  |
| 6.3 Schematic of 5V Buck Converter                                                                                                                            | 87  |
| 6.4 Schematic of 3.3V Buck Converter                                                                                                                          | 88  |
| 6.5 Simulation of Startup for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.                                                                         | 88  |
| 6.6 Simulation of Load for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.                                                                            | 89  |
| 6.7 Simulation of Steady State for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.                                                                    | 90  |
| 6.8 Schematic of 1.8V Buck Converter                                                                                                                          | 91  |
| 6.9 Schematic of 0.9V Voltage Regulator                                                                                                                       | 91  |
| 6.10 36V Generator and Phantom Power enabled XLR circuit                                                                                                      | 92  |
| 6.11 Schematic of one ADC and Voltage Inverter                                                                                                                | 93  |
| 6.12 Schematic of DAC circuit and Post-DAC Low Pass Filter                                                                                                    | 95  |
| 6.13 Schematic of Headphone Amplifier and Line/Headphone Output                                                                                               | 96  |
| 6.14 Example Organization of Mixed Signal Circuit                                                                                                             | 97  |
| 7.1 Use-case diagram for mixer software                                                                                                                       | 101 |
| 7.2 Deployment diagram of computing hardware                                                                                                                  | 103 |
| 7.3 Packetization of FFT Signals                                                                                                                              | 107 |

# Chapter 1

## Executive Summary

As students who are also in bands, we are constantly searching for ways to make the set-up process more streamlined, whether this be through buying more gear or learning new workflows. However, at the end of the day, we run into problems with pricing and modularity. Normally, an entire rig of equipment will set the musicians back multiple thousands of dollars, all while taking up large amounts of space and creating a mess of cabling. To solve these practical limitations, we envisioned a senior design project with the following goal - creating a low-cost and high performance audio mixer. Additionally, with the major boom in the usage of artificial intelligence, we were inspired to implement some novel AI features to streamline the workflow of audio mixing.

While audio mixers are ubiquitous, the most common (and subsequently lowest cost) are of the analog variety. Analog mixers are a tried-and-true technology that lack any option for configuration as well as being generally bulky. From experience, it is also relatively common to have errors with the mixing console during live settings, too, which is not ideal and introduces more problems into the loop. While analog mixers have some filters (low, mid, high) and effects (reverb, delay), anything more than that is only seen in the higher echelon professional gear. If a traveling musician needs to use their analog mixer to monitor and mix themselves, they would also need to travel with a bulky rack of pedals to achieve the various audio effects they desire on their instruments. This brings us to digital mixers, which while having more configurability, cost a fortune for even the lowest level of gear. This is not to mention that these digital mixers usually do not have touchscreen functionality and rather rely on faders entirely, as well as being run by low-powered computers. The mixers are able to work due to the use of highly specialized DSP chips, which are both difficult to interface with and configure.

With this, we seek to combine the present powers of a low-cost general computer to create an all-in-solution for audio mixing that includes AI auto-mixing features, similar to those seen in Izotope Neutron 4. This device would allow for high-level mixing capabilities at a fraction of the price while also combining hitherto separate technologies. Utilizing many off-the-shelf components, we are attempting to create an approachable and modifiable device that will be able to be more robust and feature-rich than any standard analog or digital mixer. The computer itself will be programmed using a real-time operating system to maximize data processing speed, and a custom interface will be designed to ensure a smooth experience for the end user.

To conclude, we seek to create a project that advances the world of low-cost

audio mixing while including capabilities not yet on present devices through emulation of state-of-the-art techniques currently used in the audio engineering field. Through this project, we will have a deeper understanding of the fundamentals of audio engineering along with the creation of a usable and effective product for our future selves. By combining general computing audio technologies with mixers, MISTER MIXER aims to allow ourselves and other musicians to be able to produce a powerful device at a fraction of what present consumer devices cost.

# Chapter 2

## Project Description

### 2.1 Motivation and Background

As mentioned in the executive summary, music is an expensive and somewhat cumbersome hobby to partake in; there is a lot of equipment required. This extends to all facets of playing and performing. Many beginner bands have no feasible way to mix their audio, which ultimately leads to subpar, unmixed performances that affect audience enjoyment or even playing ability. Even if they possess some sort of cheap, analog mixer, they face limitations with creative effects, parameter recall, workflow intuitiveness, and bulkiness (especially if they need a pedalboard setup on top of the mixer).

These limitations were experienced by the authors themselves. As former members of a band, they would perform in miniature concerts on top of UCF garages. When multiple bands would perform, substantial intermissions would take place between performances due to the reconfiguration of instrument connections and the analog mixer board. While some overhead was unavoidable, a lot of it could have been negated with mixer presets, instant recall, and a streamlined mixing workflow. This led the authors to realize that there is a niche in the market for an all-in-one mixing solution for fledgling bands and performers.

### 2.2 Existing Product/Past Project/Prior Related Work

The closest existing product to our proposed device is the Behringer Flow 8. It features 8 analog inputs including XLR and line inputs, as well as a Bluetooth streaming input. It also features wireless control over Bluetooth with a mobile app

in addition to the analog controls. The mixer has the capability to apply several musical effects on send channels [2]. The main strength of this digital mixer is the low price for the amount of functionality provided – namely, the Bluetooth capabilities and competent control scheme. Even so, there is not much more to the device in terms of workflow features and creative effects. It lacks digital parameter recall. This is partly a design choice made due to physical limitation; it lacks expensive, digitally controlled flying faders and instead opts for analog faders which do not offer any digital state control. Additionally, the library of DSP effects is not very comprehensive. There is still a clear need for creative effects and pedalboards to be applied before any instrument enters the mixer. These limitations have spurred us to create something more useful for the small performing band. In comparison, our mixer would have digital recall, many more creative effects, and an auto-mix feature.



Figure 2.1: Behringer Flow 8 - Compact 8-Channel Mixer. **MSRP \$299.99**

Another product similar to our proposed device is the Mackie MobileMix 8-channel mixer. It is generally geared towards smaller performances in outdoor or other mobile settings, similar to our project. This miniature mixer features 8 channels of varied analog input (with 2 microphone preamps), Bluetooth input, monitoring outputs, and a small selection of effects (2-band EQ, reverb on a send channel). The control scheme is fully analog; the board is entirely controlled with potentiometers and buttons [3]. Like the previous example, this mixer once again has a very enticing cost to performance ratio, but the capabilities are little more than that of a typical analog mixer. The Bluetooth capability is once again a very nice feature to have, but for over \$200, we believe a mixer is capable of much more functionality. In comparison, our mixer would have digital recall, more creative effects, and the auto-mix features. Other mixers similar to the MobileMix also have very few effects other than panning and EQ functionality.



Figure 2.2: Mackie MobileMix - Compact 8-Channel Mixer. **MSRP \$229.99**

This project attempts to augment this type of pre-existing product and bring it more in line with a higher-end mixer, all the while being similarly priced (\$230 for the Mackie mixer as reference price). The features that will remain in common include volume balancing/control, panning, analog and bluetooth inputs, and portable form factor. The areas that would be expanded on will be explained in the paragraph below. Here are several commercially available digital mixers that offer a comparable feature set to what we would like to propose. They are the Allen & Heath CQ-18T and the QSC Touchmix 8. They are displayed in the figures below, 2.3 and 2.4 [4] [5].



Figure 2.3: Allen & Heath CQ-18T - Compact 16-Channel Digital Mixer with Wi-Fi.  
**MSRP \$1,199.99**



Figure 2.4: QSC Touchmix-8 - 14-Channel Digital Mixer. **MSRP \$1,149.99**

Several standout features of these mixers include frequency monitoring, wireless control with a tablet app, digital parameter recall, multiple auxiliary mixes, rudimentary auto-mix tools ("Anti-Feedback Wizard", "Gain Assistant"), and parametric equalizers. These mixers are very feature-rich compared to both the Mackie and Behringer mixers in computational power, workflow features, signal routing, and effects, but this comes with a significant cost increase, almost \$1000 over the others. Additionally, they are less portable (particularly the Allen & Heath CQ-18T). Ideally, our mixer would be comparable to these digital mixers with the same price point and form factor as the Mackie mixer, all with the addition of the new auto-mixing features as well as a control scheme that mimics those of Digital Audio Workstations. This would create an option in the market for a mixer that is affordable, yet extremely capable with a novel workflow that allows for greater creativity than with the typical analog mixer.

We will concede that several features found in high-end digital mixers must be omitted or pared back due to practical constraints. One of these is physical controls. Due to design time and cost constraints, almost all controls on our mixer will be through the touchscreen. There may be one rotary encoder, but we will lack the plethora of physical buttons and knobs that these competing mixers have. One other feature that would not be present is the capability to configure multiple different auxiliary and monitor mixes. We only have one line-out and headphone out, so this would not be possible to implement. Furthermore, even if we did decide to upgrade the amount of hardware outputs, we believe that would not significantly benefit the performances of small bands who do not have multiple PA systems and monitoring setups. This would simply incur extra cost with little to no benefit.

### 2.2.1 USB Audio Interfaces

A goal in our project is to utilize the mixer to have the ability to present a USB audio interface to the user to allow them a wider use of audio processing tools. USB audio interfaces are devices that act as a bridge between analog audio signals and a computer or digital audio workstation (DAW). DAWs are software platforms used for recording, editing, mixing, and mastering audio. The process is done by converting the analog signals into a digital format that can be processed by digital systems. The interface captures audio signals (e.g., from microphones, instruments, or mixers), converts them to a digital format using ADCs, and transmits them to the computer via a USB connection. Additionally, it can also convert digital audio data back into analog signals using DACs, allowing the computer to send audio to monitors, speakers, or headphones. They usually feature high quality audio converters and pre-amps, which can offer better sound quality compared to the built-in audio interface of a PC or digital device.

There are several products on the market that integrate a USB audio interface into their mixer, similar to the task that we are trying to accomplish.



Figure 2.5: Tascam Model 16

The Tascam Model 16 features a built-in USB Audio Interface to send/receive (16-in/14-out) audio data to/from DAW [6]. One of the primary advantages of adding USB audio interface capabilities to our project, as demonstrated by the Model 16, is the seamless integration it provides with DAWs. Through a USB audio interface, audio signals from the mixer can be routed into the DAW for multi-track

recording. This allows for high-fidelity recording for small bands. In addition, an attractive feature of the USB audio interface is that users can leverage the processing capabilities of the DAW to apply effects, EQ, reverb, or compression, beyond what our mixer's onboard processing may be able to provide.



Figure 2.6: Soundcraft Signature 22 MTK

The Soundcraft Signature 22 MTK offers an ultra-low-latency USB interface that flawlessly captures every channel, which can then be mixed or transferred to a DAW for further mix-down and production [7]. Another significant advantage to utilizing a USB audio interface is how it can facilitate real-time monitoring and playback of audio signals from the DAW or the computer. This allows users to playback processed audio from the DAW through the mixer for further refinement or output to speakers, allowing flexibility between digital and analog audio processing.

## 2.3 Goals and Objectives

We desire to make a digital mixer that has a compact form factor, an abundance of creative effects, and an intuitive workflow that augments the creative vision of any group of performers. This will provide an all-in-one gigging solution for small bands. The goals needed to realize this vision are outlined below.

### 2.3.1 Basic Goals

For one, the mixer will provide basic mixing capabilities. Fundamentally, it must be able to take in audio signals, sum them, and output a mixed audio signal. On top of this basic functionality, the mixer will provide a library of creative DSP effects that can be applied to the audio signals. This will allow for a compact, all-in-one performing solution that removes the burden of carrying separate audio effects like guitar pedals.

For control, the mixer will have an intuitive touchscreen interface to easily control effects and routing. Additionally, the mixer will also be able to be wirelessly controlled through the use of a mobile app. This will allow for the mixing engineer to rapidly make mix decisions from the audience's perspective. The use of fully digital interfaces allows for retention of state (e.g. saving presets) across performances and will drastically speed up the mixing process.

Adding to the convenient control scheme, the mixer will have an easy-to-use auto-mix feature to automate setup and the mixing process. The feature will ideally give the performers a competent, great-sounding mix or, at the very least, provide a solid foundation upon which small creative adjustments can be made.

Finally, the mixer will be able to act as a USB audio interface. This will enable bands to record instruments and vocals directly into a Digital Audio Workstation. Such functionality greatly increases the value proposition of the project, eliminating the need for the band to buy a separate USB audio interface to record their performances.

### **2.3.2 Stretch Goals**

The team has several stretch goals which may be explored after the conclusion of Senior Design. The stretch goals will increase the functionality of the mixer even further and make it a comprehensive solution for small bands. They are listed below.

One such goal is to support the use of external control interfaces such as foot switches and MIDI controllers. Normally, guitarists and other performers will adjust effect parameters on their pedalboards, keyboards, or other effect interfaces. This mixer aims to eliminate the need for such bulky implements, although it does introduce the downside that such typical control interfaces are taken away from the performer. This feature will reintroduce such control to the performers and even augment the control performers have due to the power of digital configuration. External control will allow for performers to make adjustments on the fly and bring a new level of dynamism to their performances.

Another goal is to allow for audio connection to the mixer by Bluetooth. This will enable backing tracks to be played with ease from a performer's phone or other mobile device. This negates cumbersome routing and cable adapter setups whereby a phone must be connected through a headphone adapter, cable, and 3.5mm to 1/4" adapter into an analog input.

### **2.3.3 Objectives**

The specific technical objectives below outline the implementation details required to achieve the desired functionality, with a focus on both essential features and additional enhancements.

There are several essential features that make up the feature set of a basic mixer. In this case, there will be 8 analog input channels with analog to digital conversion and preamplification to handle microphone, instrument, and line level inputs. For outputs, the mixer will possess both a line output to drive PA systems as well as a headphone output for personal monitoring. As for the actual mixing, the device will possess basic functions such as volume control, panning, and summing ("mixing") capabilities.

For the control scheme, as mentioned above, there will be both an onboard touchscreen LCD with a GUI as well as a wireless control scheme through a mobile

app.

In order to augment the basic capabilities of a mixer, there will be a collection of DSP effects such as reverb, delay, compression, equalization, chorusing, and distortion.

In order to expedite the mixing process, there will be a software auto-mix features that performs automatic volume balancing and equalization and applies creative DSP effects based on selected genre and detected inputs.

## 2.4 Key Engineering Specifications

Table 2.1 outlines the specifications targeted for this project. These specifications were established to ensure our digital mixer remains feature-rich compared to the consumer market options while also considering the time and money constraints of the project.

| Attribute                          | Description                                                                                                                                                          |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Dimensions</b>                  | 15" x 10" x 3"                                                                                                                                                       |
| <b>Weight</b>                      | <10 lbs                                                                                                                                                              |
| <b>SNR (Signal to Noise Ratio)</b> | >70 dB                                                                                                                                                               |
| <b>Inputs</b>                      | 8 TRS, 4 XLR (w/ combo jack), 1 USB, Bluetooth                                                                                                                       |
| <b>Outputs</b>                     | 1 Stereo Out, 1 Headphone-out, 1 USB                                                                                                                                 |
| <b>Input Gain</b>                  | >50 dB                                                                                                                                                               |
| <b>Digital Effects</b>             | Gain, Panning, Delay, Compression, Equalization, Chorus/Flanger, Distortion, Reverb                                                                                  |
| <b>Power Usage</b>                 | <75W                                                                                                                                                                 |
| <b>Auto-Mixing</b>                 | Automatically set levels, EQ, creative effects, subjectively improved over an unmixed performance. Qualitative listening tests, stereo loss to evaluate performance. |
| <b>Cost</b>                        | <\$400                                                                                                                                                               |
| <b>Overall System Latency</b>      | <20ms                                                                                                                                                                |

Table 2.1: Table of Proposed Design Specifications

## 2.5 House of Quality

House of quality is incredibly useful for comparing marketing requirements against engineering requirements. Below, the house of quality for this project is presented.

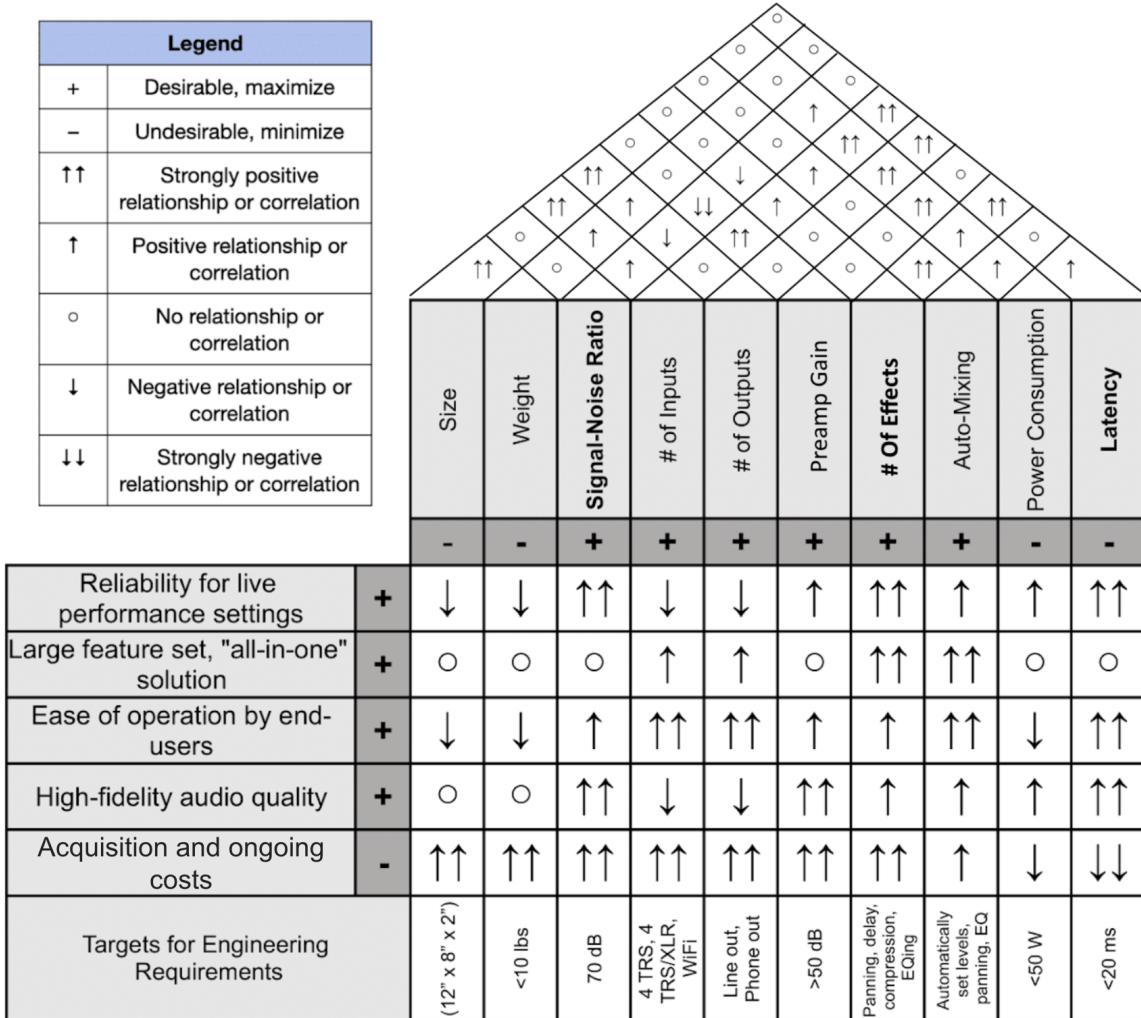


Figure 2.7: House of Quality Diagram

## 2.6 Hardware Block Diagram

Included below in Figure 2.8 is our proposed hardware diagram. The blocks are colored to indicate the main person responsible, and the color key is included on the diagram. The overall topology of our system is a dual-processor setup with one MCU to handle DSP for streams of audio and a Raspberry Pi 5 SBC to handle state management, wireless communications, and the AI auto-mix model.

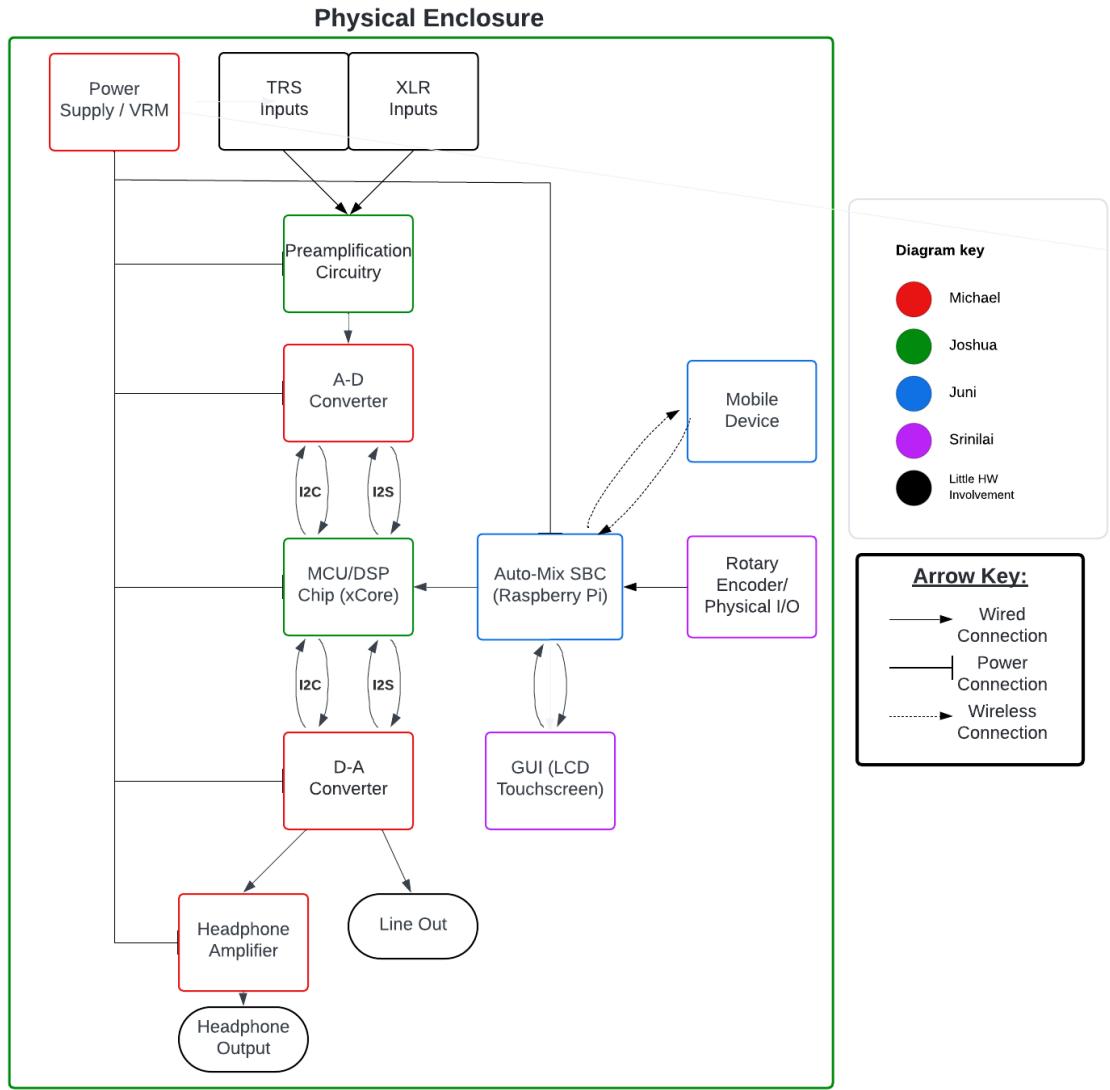


Figure 2.8: Proposed High-level Hardware Block Diagram

## 2.7 Software Architecture Diagram

Our prospective software architecture is illustrated in figure 2.9. Software components are distributed between the Raspberry Pi 5 and an XMOS xcore. The Raspberry Pi 5 is responsible for managing the user interfaces, state management, and higher-level audio processing tasks, while the xcore handles low-latency digital signal processing (DSP) operations for real-time audio effects.

The diagram is divided into two primary sections: the **Raspberry Pi 5** and the **XMOS xcore**. On the Raspberry Pi, there are several core modules:

- **State Manager** (blue) manages the mixer's central state information, ensuring all connected modules maintain consistency in settings like volume, panning, and effect parameters.
- **Audio Manager** (blue) oversees the flow of audio data between the Pi and the DSP system. It is responsible for handling audio streams, downsampling data for visualization, and passing raw audio to the auto-mixing model and the user interfaces.
- **Onboard Interface** (red) and **Mobile Interface** (red) are user interfaces connected to the manager modules via IPC (inter-process communication) and WiFi, respectively. These interfaces allow real-time control over the mixer settings and audio effects.
- **Auto-Mixing Model** (green) automates various audio processing tasks, including volume balancing and applying effects based on AI model inferencing.

The **XMOS xcore** handles real-time audio DSP through the **DSP Manager** (yellow), which manages the chain of audio effects applied to the incoming audio streams. These effects are processed and then output to either speakers or external devices via USB. The connection between the Raspberry Pi and the xcore is established via **SPI**, allowing the state and audio data to be transmitted between the two processing units in a synchronized manner.

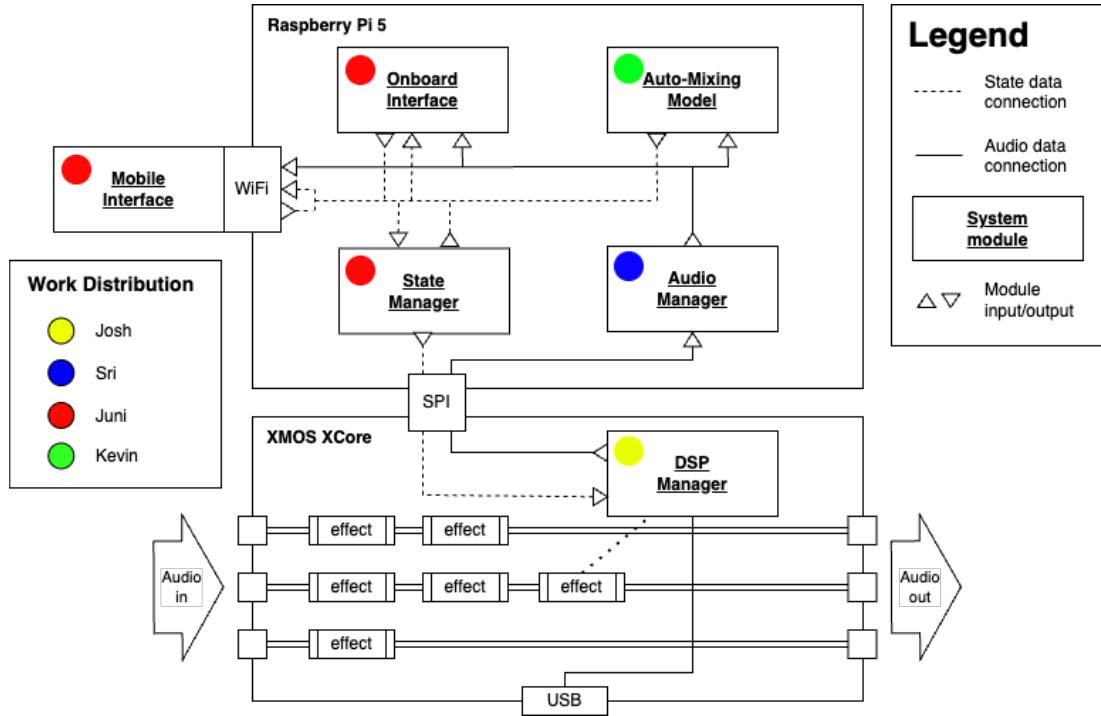


Figure 2.9: High-Level Software Architecture Block Diagram

# Chapter 3

## Research

All good engineering begins with research and it is the imperative responsibility of the team to research the components that make up the mixer. This chapter is split into two overarching sections, hardware research and software research. In each section, different technologies will be explained. Furthermore, comparisons will be made between specific parts. The selected parts will be highlighted and a justification will be given. The two sections are outlined below.

## 3.1 Hardware Research and Comparisons

### 3.1.1 Audio Signals

Before any circuitry work is to be done, identifying the types of analog audio connections that are required in a mixer is necessary for overall understanding. Differentiating between stereo and mono signals as well as balanced and unbalanced connections is important before the signal processing stage to ensure that the signal is transmitted properly. Additionally, we must name and differentiate the physical connectors that are to be used for the mixer, those being TS/TRS and XLR.

#### Signal Types: Mono vs. Stereo

There are several broad categories of audio signal types. One of these categories relates to the number of channels, that is, mono versus stereo.

Mono audio, as specified by the name, is a signal with a single channel of audio. It is by and large the standard input type for mixers and other professional audio applications. Even in applications where stereo audio is needed, stereo audio is typically transmitted through two mono cables rather than a single stereo cable. Why this is the case will be touched on in the following paragraphs.

Stereo audio is a signal with two channels of audio, typically a left and a right channel. Stereo audio is useful because it allows an audio stream to have a sense of direction and width. Without delving too deep into psychoacoustics, stereo allows for audio to be placed in space by simply adjusting the balance of audio between the two channels. For example, if you listened to a pair of headphones, you would perceive a sound as coming from the right if it only played through the right driver. Similarly, you would perceive a sound as coming from the left if it only played through the left driver. It would come from the center if played in equal volume through both drivers.

As touched on above, the signal types that our mixer will be interfacing with are predominantly mono audio signals. The only notable exception is the main output and headphone output, which can transmit stereo. This is because the process of mixing generally follows a template where mono audio signals are "placed" into a stereo sound-field to produce an immersive listening experience. That is what our mixer will give artists the ability to do.

#### Signal Types: Unbalanced vs. Balanced

Another broad category of audio signal types related to noise-rejection properties, that is, unbalanced versus balanced signals.

Unbalanced signals only have two connections, a signal connection and a ground connection. The signal is simply transmitted and measured as the difference between the signal and ground terminals. Unbalanced connections are

simple but prone to noise and interference. Thus, unbalanced signal cables must be restricted in length; the longer the cable, the more noise the signal will pick up.

Balanced signals have three connections, two signal connections and a ground connection. The key is that the two signal connections are wired so that they are 180 degrees out of phase with each other. At the end of the connection, the signals will then be phase-aligned again and summed. Any noise that is picked up by the two signal connections of the balanced cable will now be 180 degrees out of phase with its counterpart and removed when summed through destructive interference. This gives balanced cables excellent protection against noise; thus, they are preferred in professional audio environments. Balanced cable runs can be much longer.

### 3.1.2 Audio Connectors

#### Connector Types: TS/TRS

The most common audio connectors that are used are both the TS and TRS connections. TS connections consist of the Tip and Sleeve, whereas TRS connections have a wire for Tip, Ring, and Sleeve. As shown by Figure 3.1 featuring male TS and TRS connectors, both connections look very similar, but TRS can be used for unbalanced stereo or balanced mono inputs whereas TS cables can only be used for unbalanced mono sound inputs [8]. These connections can either be 1/4" or 1/8", but the normal audio standard for input from instruments is 1/4".

Of note is that while TRS connectors have the necessary quantity of pins to connect to balanced microphones, they are not frequently utilized because of the segmented design of the connector. Many microphones require phantom power. If phantom power was applied across the hot and cold sections on the TRS connector, the sections would short as they were being plugged and unplugged.

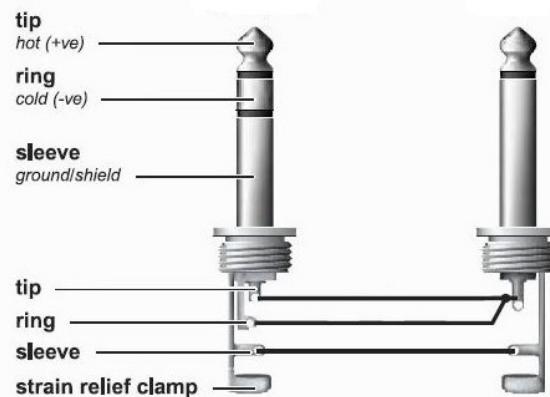


Figure 3.1: TR and TRS Wiring Key

## Connector Types: XLR

The last connection of note for this project is an XLR connection, which is predominantly used for microphone signals. Similar to the TRS, it usually features 3 (but can have up to 7) pins, with the first pin always being dedicated to ground. Figure 3.2 demonstrates that the first pin of a female XLR connection is ground, the second pin is positive, and the third (bottom) pin is negative. With its three-pin topology, XLR cables are capable of carrying both balanced mono audio and unbalanced stereo audio, however, balanced mono audio is the typical use case. In male XLR connectors, pins 1 and 2 swap sides, but the connectors still have the same pattern [9]. Pins 2 and 3 on XLR connectors are capable of carrying a Phantom Power signal in devices such as a ribbon or condenser microphone that require power to operate. The separated pin design ensures that the phantom power is delivered safely without the possibility of any shorting, unlike TRS cables.

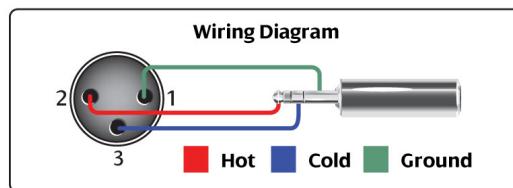


Figure 3.2: XLR to TRS Connection Diagram

## Audio Connector Comparison

Below is a tabular comparison of all the covered audio connectors. All three will be utilized in the mixer's design.

| Connector             | TS         | TRS      | XLR      |
|-----------------------|------------|----------|----------|
| # Connections         | 3          | 3        | 2        |
| Connection Capability | Unbalanced | Balanced | Balanced |
| Phantom Power         | No         | No       | Yes      |

Table 3.1: Audio Connector Comparison

### 3.1.3 Preamplifiers

#### Input Levels

The process of preamplification must begin with an explanation of the purpose of preamplification as well as different input levels found in audio. Preamplification is the process of bringing up weak analog audio signals to a standardized "line" level such that they can be processed or further amplified (power amplification)

by standard audio equipment. In the case of this project, all input audio must be brought up to line level so that they can be fed into the ADC circuitry.

What is line level? For audio, there are two general standards. One is consumer line level, which is typically measured to be -10dBv. Another is professional line level, which is measured to be around +4dBu [10]. As is described by the name, professional line level is what is typically used in professional production gear such as mixers and PA systems. As such, that is the level that our mixer preamplifiers must output to the ADCs. Professional line level measures on the order of magnitude of 1 volt, 1.3 volts RMS to be specific. One may notice the discrepancy in the measurement units of the consumer and professional line level. Consumer line level is expressed in dBv, while professional line level is expressed in dBu. dBv is measured in reference to 1 volt; 0 dBv is 1 volt. dBu is measured in reference to 0.775 volts; 0 dBu is 0.775 volts. This is an odd number, but makes sense in the right context. If you calculate the power dissipated by a 0 dBu signal through a 600 Ohm load (a common value for transmission lines), you end up with exactly 1mW of power [11].

Another common input level is instrument level. Instrument level is outputted by common instruments such as electric guitars and basses. There can be a wide range of levels, but a rough estimate of instrument level places it at -20 dBu [12], on the order of 100 mV. As such, the preamplifier will need to provide a moderate amount of amplification (20-30 dB) to produce an adequate signal for the ADC.

Beyond signal, another point of discussion is the input impedance. Instrument inputs are unique in that they are high-impedance, on the order of a million ohms. This is in contrast with the typical impedance of a microphone or line input, which is on the order of ten thousand ohms. As such, in order to prevent undesired loading and filtering side effects, the instrument input must have its own high-impedance signal path. They cannot just be plugged into a microphone or line preamplifier circuit with adjustable gain.

One last common input level is microphone level. Microphone level is extremely low, around -40 to -60 dBu. This is on the order of 1 mV to 10 mV. As such, the preamplifier will need to provide at least 40 dB of gain in order to amplify the microphone input to a usable signal level. This will be taken into account in the preamplifier design.

## Phantom Power

One important consideration in a microphone preamplifier design is phantom power. Many microphones (typically condenser microphones) require a DC voltage to be applied from the preamp in order to operate. The most common phantom power voltage specification is 48 volts, although more specifications exist. There are also standards for 24 volt phantom power and 12 volt phantom power. The voltage is brought into the circuit with a pair of resistors defined by the specification and applied to pins 2 and 3 of an XLR connection. Below is a high-level schematic of a phantom power circuit.

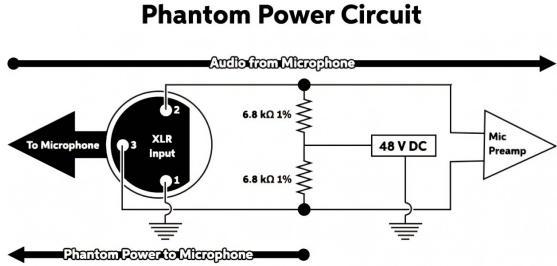


Figure 3.3: High-level Phantom Power Schematic, courtesy of Sweetwater

Most microphone designs today do not necessitate the use of the full 48 volts. In fact, most microphones step down the phantom power voltage and clamp it steady with a Zener diode at approximately 11 volts peak. However, for wider compatibility, the full 48 volts is used in most professional designs. In terms of current draw, the nominal current draw across all three phantom power specifications is quite low. For example, the 48 volt phantom power specification specifies a 10 mA nominal current draw [13]. Below is a table summarizing the three phantom power specifications.

| <b>Phantom Pwr.</b> | <b>Volt., no load</b> | $I_{max}$ ( <b>Shorted</b> ) | $R_{feed}$            | <b>Op. Current</b> |
|---------------------|-----------------------|------------------------------|-----------------------|--------------------|
| P12                 | $12V \pm 1V$          | 35mA                         | $680 \parallel 680$   | 17mA               |
| P24                 | $24V \pm 4V$          | 40mA                         | $1.2k \parallel 1.2k$ | 20mA               |
| P48                 | $48V \pm 4V$          | 14mA                         | $6.8k \parallel 6.8k$ | 10mA               |

Table 3.2: Standards for Phantom Power: IEC 61938:2018

One very interesting fact established through a design article is that the phantom power specification does not have to be obeyed at all. One can provide phantom power to microphones at non-standard voltages by simply adjusting the value of the resistors that bring in the DC phantom power voltage [14]. For example, one can provide a 15 volt phantom power supply by bringing the voltage in with 820 ohm resistors. This is an expedient design choice that can be exploited so that phantom power can be supplied from a wide range of voltages.

## Preamplifier ICs

Operational amplifiers are the backbone of any preamplification circuit. There are a plethora of popular op-amps tailored for use in audio applications. They are particularly useful due to their differential inputs. Some popular audio pre-amps are compared below.

| Preamplifier               | SSM2019               | THAT1512              | OPA2134                 |
|----------------------------|-----------------------|-----------------------|-------------------------|
| # Channels                 | 1                     | 1                     | 2                       |
| Max. Gain (dB)             | 70 dB                 | 64 dB                 | 120 dB                  |
| Supply Voltage (V)         | $\pm 5V$ to $\pm 18V$ | $\pm 5V$ to $\pm 20V$ | $\pm 2.5V$ to $\pm 18V$ |
| Noise f=1 kHz, G=1000)     | 1 nV/Hz               | 1 nV/Hz               | 8 nV/Hz                 |
| THD+N (f=1kHz, G=1000)     | 0.017%                | 0.008%                | 0.00008% (G=1)          |
| Slew Rate (V/ $\mu$ s)     | 16 V/ $\mu$ s         | 19 V/ $\mu$ s         | 20 V/ $\mu$ s           |
| CMRR (G=1000)              | 130 dB                | 120 dB                | 100 dB                  |
| Supply Current             | $\pm 4.6$ mA          | 6 mA                  | 4 mA                    |
| Output Swing ( $\pm$ Vout) | $\pm 13.9V$           | $\pm 13.3V$           | Approx. $\pm 16V$       |
| Price                      | \$5.14                | \$6.46                | \$5.24                  |

Table 3.3: Comparison of Preamp ICs

Preamplifier ICs have stellar performance across the market. They are usually not the bottleneck in any analog design. As such, the specifications are considered; however, the driving factor in preamplifier IC selection is the availability of reference designs and robust documentation. An excellent reference design has been found that utilizes a dual-IC topology with an Analog Devices SSM2019 and a Texas Instruments OPA2134. The preamplifier design will be adapted from the reference design. As such, they are the preamplifier ICs of choice.

### 3.1.4 Signal Converters (A/D & D/A)

Arguably, the most important part of any audio equipment is the Signal Conversion chain – both converting the analog audio signals into digital ones for processing as well as recreating the output audio signal from digital to analog. Thus, understanding of the entire signal chain is necessary to the proper functionality of our digital mixer.

#### Analog-to-Digital Converters

In our mixer, the analog inputs in the system will be balanced Line Out signals TRS signals (mono) and XLR signals (stereo). After these are sent through the preamplifier circuitry (to allow the dynamics of the signals to be better identified), they must be sent into an ADC for Analog-to-Digital conversion. Since most signals coming into the mixer will be stereo signals (one channel for each side), we will be using an ADC that has stereo capabilities (both a left and right input channel).

Before talking about the technologies used in ADCs, the two most important parameters involved must be addressed: Signal-to-Noise Ratio (SNR) and bandwidth. SNR, as the name suggests, represents the ratio between the input signal (non-noise bits) and latent noise in the signal pathway, which is measured in dB.

The higher the SNR, the more accurate the overall conversion process, and thus the audio signal will be less lossy. A typical SNR for a mixer will be  $\approx 70$  dB. Bandwidth is the range of frequencies that ADCs can accurately process. Due to the Nyquist-Shannon sampling theorem, which states that the maximum bandwidth cannot be more than one half of the sampling rate of the device without causing aliasing (distortion in the sampled signal) [15], as shown in Figure 3.4 with the triangle wave input. The higher the sampling rate, the higher the bandwidth, and the sampling rate is known as the Nyquist rate. For audio applications, the audible range is around 20-20000Hz, which means the sampling rate needs to be at least 40kHz to cover the entire frequency range, but the standard sampling rate is 44.1kHz to have a buffer for any extra frequencies. High fidelity audio (HiFi) nearly doubles this frequency to a total of 96KHz sampling rate, but there is not a real reason to do that with an audio mixer that is intended to be used live due to distortion from sources like speakers.

The bit number (degree of specificity for A/D conversion, given by a  $2^n$  number of combinations) of an ADC heavily affects the SNR, with a higher number of bits giving a higher SNR. Sampling rate also affects the SNR, but much less than the number of bits. This is seen where the number of bits adds  $\approx 6$  dB per bit to the SNR, whereas doubling the sample rate only adds  $\approx 3$  dB to the SNR.

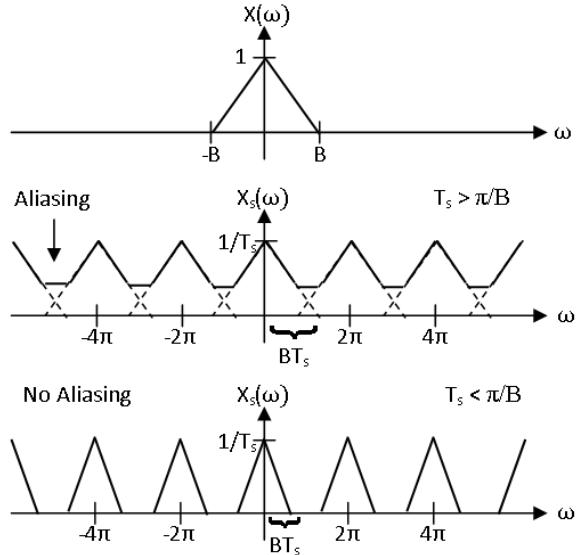


Figure 3.4: Example of Aliasing in Triangle Wave, created by Fraïssé and licensed under CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)

There are two popular methods of A-D conversion: Successive Approximation (SAR) and Sigma-Delta Conversion. SAR is an older technology that uses comparators between the input signal and internal DAC to approximate (hence the name) the voltage, resulting in an identified voltage of specified resolution depending on the number of bits [16]. This method is inherently noisier than Sigma-Delta conversion, though it can be done at low power in a small form factor.

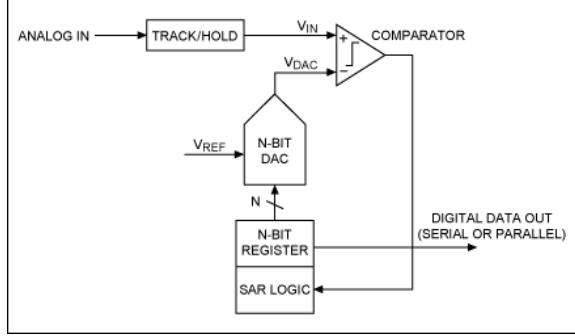


Figure 3.5: Architecture of N-bit SAR ADC, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.

As shown in Figure 3.5, the architecture inside of a SAR ADC is relatively simple, with a reference voltage being used to supply the DAC, which each bit adding another comparison operation to the ADC. Due to the intrinsic reliance on the internal DAC for conversions, it must be properly calibrated for accurate functionality. Successful operation of voltage comparisons of a 4-bit SAR can be shown below in Figure 3.6 [17]. Since these devices require successive operations, they are relatively slow compared to their sample rates (in the magnitude of megasamples per second), but this does not affect our operations as audio information is relatively slow.

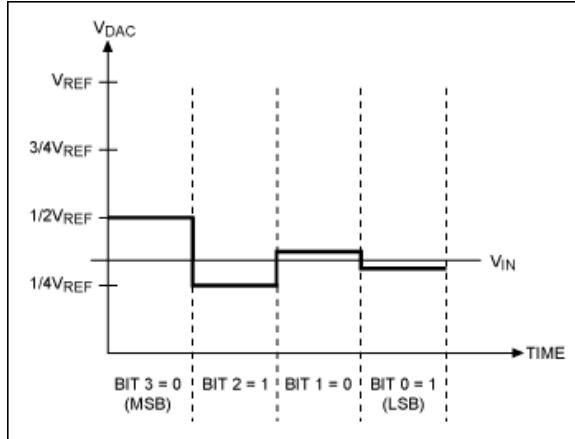


Figure 3.6: 4-bit SAR ADC example operation, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.

The internal operations of Sigma-Delta converters are relatively the same, using an input DAC with reference voltages as feedback to be summed with the input signal via a switched capacitor circuit [18]. Sigma-Delta converters offer higher resolution capabilities in addition to less noise during conversion at the cost of reduced bandwidth. Normal Sigma-Delta converters for audio applications may only use around 22KHz in bandwidth yet have sampling rates of magnitudes more, due to oversampling methods (usually 16 times the required sampling rate). Although

the total usable sample rate is lower, there is no requirement for anti-aliasing and noise does not have much of an affect on the conversion (thus higher SNR). Because of its advantages with regards to signal integrity, our device will be using Sigma-Delta converters to maximize the overall SNR.

### ADC Comparison

In choosing an ADC, we desired a resolution of at least bits and minimum sampling rate of 44.1K while having an SNR of over 90. Due to the mixing needing 8 audio inputs, finding an ADC that had maximal input channels while supporting I2S was paramount. In the commercial industry, Cirrus Logic ADCs are used most of the time. With that being said, due to the cost requirements, we chose the PCM1865 to utilize for the Analog-to-Digital conversion. This ADC also includes build in preamplification gain, which allows for more configurability of the signals post-preamp, and can operate on a single 3.3V supply. Daisy-chaining two of the PCM1865's together using I2C allows the two devices to communicate via TDM, allowing eight channels of audio to be sent to a single input pin on the DSP chip. The low cost and configurability made it an easy choice to choose for this project.

| Part                 | Resolution | Sampling Rate | SNR    | Price  |
|----------------------|------------|---------------|--------|--------|
| <b>PCM1808PWG4</b>   | 24 bits    | 96kHz         | 99 dB  | \$2.34 |
| <b>PCM1851APJT</b>   | 24 bits    | 96k           | 101 dB | \$3.21 |
| <b>PCM1822IRTER</b>  | 32 bits    | 8k - 192k     | 111 dB | \$3.67 |
| <b>CS5334-KS</b>     | 20 bits    | 2kHz - 50kHz  | 100 dB | \$3.96 |
| <b>CS5335-KS</b>     | 20 bits    | 2kHz - 50kHz  | 105 dB | \$3.96 |
| <b>CS5330A-KSR</b>   | 18 bits    | 2kHz - 50kHz  | 90 dB  | \$6.03 |
| <b>AD1877JR-REEL</b> | 16 bits    | 48k           | 90 dB  | \$9.21 |

Table 3.4: ADC Comparison Chart

### Digital-to-Analog Converters

After signal processing has taken place inside the microcontrollers, the digital audio signals must be converted back to analog signals to be able to be amplified and ultimately heard. Fortunately, the inner workings of DACs are relatively simpler than ADCs (as evidenced by the fact that many use internal DACs in comparator circuitry) and are robust for audio applications.

One of the simplest and textbook DAC solutions is using a resistor ladder e.g.  $R, 2R, 4R$  in a summing amplifier to obtain the output voltage, where the most significant bit would be at the resistor corresponding to  $R$  (and whether or not it is an open or closed circuit depends on the bit value being 1 or 0 at that point). Figure 3.7 shows this simple method of D/A conversion, but it is easy to see the quick downsides to this method. The most glaring issue is that if each subsequent bit

requires 2x the previous resistance, it exponentially increases to a point where it becomes unfeasible to process more than 4 bits of information – while this does provide a resolution of  $2^4$  or 16 possibilities, it is nowhere near sufficient for any audio application. Additionally, resistor tolerances make it so that getting exact ratios between the base and subsequent resistors is nigh impossible, let alone the fact that standardized resistor values usually do not come in multiples of 2.

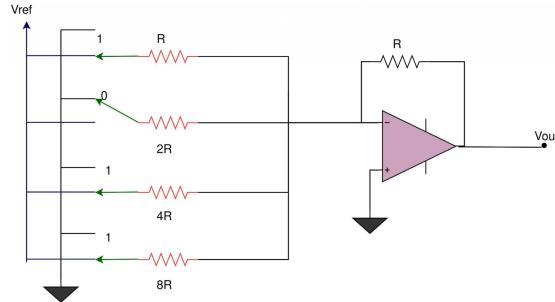


Figure 3.7: Binary Weighted DAC Example Circuit

Realistically, most DACs, like ADCs, will utilize Sigma-Delta techniques as it provides a high SNR and is not reliant on a physical comparison using a resistor ladder. As shown by Figure 3.8, Sigma-Delta DACs are heavily digitized, first going through an interpolation filter and then modulating followed by conversion. The overall process is designed to minimize noise (interpolation filter padding most of the data with zeroes, Sigma-Delta modulator acting as a low-pass filter to data, and external LPF). This is also aided by the oversampling method mentioned in the ADC section. Just as most modern ADCs utilize Sigma-Delta conversion, most DACs also utilize Sigma-Delta conversion due to the high quality functionality it provides [19].

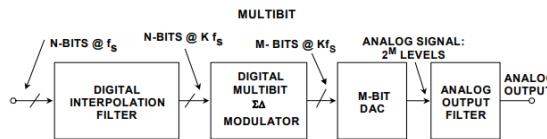


Figure 3.8: High Level Flow of Sigma-Delta DAC, Analog Devices, Inc. (“ADI”), © 2024. All Rights Reserved. These images are reproduced with permission by ADI.

### DAC Comparison

The main desires for our chosen DAC are a SNR over 90 dB, a resolution of 24 bits, and an output of 1 stereo audio channel as the signal can be split for line-out and headphone channel because we are using a headphone amplifier. Due to only needing one DAC, budgetary concerns were not particularly present and thus most of the decision went into the quality of the product as well as ease of use. Cirrus Logic DACs are also used most of the time for commercial applications, but the

PCM1780 aligns more closely with the other audio circuitry used (made by Texas Instruments) and offers ease of use coupled with low price. It is with that we chose the PCM1780 to utilize for the Digital-to-Analog conversion.

| <b>Part Number</b>  | <b>Resolution</b> | <b>Sampling Rate</b> | <b># outputs</b> | <b>SNR</b> | <b>Price</b> |
|---------------------|-------------------|----------------------|------------------|------------|--------------|
| <b>WM8521CH9GED</b> | 24 bits           | 192KHz               | 1                | 98 dB      | \$1.11       |
| <b>CS4340-KS</b>    | 24 bits           | 96KHz                | 1                | 101 dB     | \$2.00       |
| <b>PCM1780</b>      | 24 bits           | 192KHz               | 1                | 106 dB     | \$2.43       |
| <b>PCM1748KE</b>    | 24 bits           | 100KHz               | 1                | 106 dB     | \$2.69       |
| <b>CS4340A-KS</b>   | 24 bits           | 192KHz               | 1                | 101 dB     | \$2.74       |
| <b>CS4391A-KS</b>   | 24 bits           | 192KHz               | 2                | 94 dB      | \$3.08       |
| <b>PCM1742E</b>     | 24 bits           | 200KHz               | 2                | 117 dB     | \$3.18       |
| <b>CS4340-BSR</b>   | 24 bits           | 96KHz                | 1                | 91 dB      | \$3.73       |

Table 3.5: DAC Comparison Chart

### **Headphone Amplifier**

Once the audio signal is converted back to an analog signal, one must amplify it to hear through headphones. To do this, a circuit similar to a pre-amplifier is used to boost the output voltage signal into an acceptable level for headphones to be used. This circuit also helps feed current to the headphones to power them (which is how the headphones output audio). While a headphone amplifier can be as simple as a biased transistor circuit, the highest-performance and easiest to use consist of Headphone Amplifier ICs. These ICs are specifically made for audio in mind and do not require much external circuitry, all while allowing reliable operation at specified ranges.

There are a few parameters that play important roles in choosing a headphone amplifier: load impedance and maximum output power. Load impedance is the impedance of the headphones that will be used for audio playback, and the higher the impedance, the lower the output current will be (which means that it will be easier for the amplifier to drive them). Headphones with lower impedances will be able to utilize more current, which ties into the power requirements. Headphone amplifiers are capable of supplying enough power for the audio up to a point, usually less than 100mW. This means that when choosing an amplifier, one must know the general range of the headphone impedances followed by the desired gain of the output voltage signals. An important specification to look at is the Total Harmonic Distortion (THD) vs output level curves, which provide a more accurate representation of the overall performance of a specific amplifier instead of the total maximum power output. There is also the topic of headphone sensitivity (usually in dB/mV) but this ranges greatly between different types of headphones and it is better to overestimate the headphone requirements than underestimate them. As

such, choosing a headphone amplifier with a comfortable driving range is necessary [20].

### Headphone Amplifier Comparison

The most important factors when choosing a headphone amplifier are both the maximum output power and the THD + Noise ratio for the output signal. As most amplifiers above 40mW are sufficient for headphones with 16-32 ohm impedances, we chose the TPA6100A2DR for offering a very low THD + Noise factor in addition to having an acceptable maximum output power, while being very inexpensive, shown in Table 3.6.

| Part          | Max Output Power | THD + Noise (%) | Price  |
|---------------|------------------|-----------------|--------|
| LM4808MX      | 105 mW           | 0.05%           | \$0.97 |
| TPA6100A2DR   | 50 mW            | 0.04%           | \$1.05 |
| TPA6132A2RTER | 25 mW            | 0.025%          | \$1.30 |
| LM4911MM      | 145 mW           | 0.1%            | \$1.65 |

Table 3.6: Headphone Amplifier Comparison Table

### 3.1.5 Choosing The Processor Type: SBCs, MCUs, FPGAs

#### Prelude: Processor Requirements

In order to fulfill the feature requirements of the mixer, the processor must meet several key requirements. For one, it must have enough processing power to power DSP effects across at least eight channels of audio in real time. Additionally, it must have enough processing power and memory to run our AI-automix model. Most of the candidate models are quite sizeable (upwards of 2 million parameters) and require significant memory and processing capabilities to perform inference in an adequate timeframe. Additionally, the processor must feature adequate I/O to interface with at least eight channels of AD/DA conversion circuitry. Lastly, the processor must possess the ability (in and of itself or with peripherals) to communicate with another device over Bluetooth and/or WiFi protocols.

One large consideration is the use of SBCs (single-board computers), MCUs (micro-controller units) versus FPGAs (field-programmable gate arrays). Which is best suited for this project?

#### Single-Board Computers

Single-board computers are full-fledged, miniature computers. They contain one of multiple central processing units (CPUs), random-access memory (RAM), and a whole array of peripheral connectivity. They typically run full operating systems that

manage the onboard resources, allowing for parallel operation of many programs at once. They are also typically much more powerful than microcontrollers or FPGAs.

## **Microcontrollers**

Microcontrollers are essentially small computers. They contain one or multiple central processing units (CPUs), random-access memory (RAM), and input and output peripherals all in a single integrated circuit (IC). Functionally, they are designed to perform single, repeated tasks and they offer limited support for parallelism; however, this can be expanded with the use of real-time operating systems (RTOS). In terms of programming, microcontrollers typically utilize high-level system languages like C and C++ [21].

## **Field-Programmable Gate Arrays**

FPGAs are integrated circuits that offer a collection of configurable logic blocks (CLBs) and a way to configure the interconnects between these logic blocks. Fundamentally, field-programmable gate arrays offer the developer the ability to reconfigure the hardware, offering increased flexibility and performance over microcontrollers. This also allows for massive parallelism due to the configurability of the logic blocks.

However, this flexibility comes at a cost in regards to programming complexity. FPGAs are not programmed in a high-level language like C or C++ but rather a hardware design language like VHDL, Verilog, or SystemVerilog. These languages are much more difficult to program with; they fall under a completely different level of abstraction compared to high-level languages. One has to think in terms of low-level digital design, which is a completely different skill-set than procedural or object-oriented programming.

Additionally, FPGAs are often not complete systems. Peripheral interfaces are not integrated with the chip and thus must be manually developed/configured. The development cycle involves a lot more work.

## **SBCs, MCUs, and FPGAs: Conclusion**

In order to facilitate rapid prototyping and development, microcontrollers are the preferred choice for this project for audio DSP. In addition to being easier to program, microcontrollers offer integrated peripherals that are vital to our project such as serial communication interfaces for AD/DA converters and GPIO for external device control.

There are some concerns with parallelism and performance due to our multi-channel audio requirements; however, our decision is bolstered by the ubiquity of microcontrollers for DSP in the professional audio sphere.

Additionally, because of the power requirements of the auto-mix model, a single-board computer will be necessary. A single-board computer will also provide a simpler development environment with the availability of Python interpreters and

high-level development environments. The findings are summarized in a table below.

| <b>Processor</b>   | <b>Single-Board Computer</b> | <b>Microcontroller</b> | <b>FPGA</b>    |
|--------------------|------------------------------|------------------------|----------------|
| <b>Computation</b> | Very powerful                | Low Power              | Moderate Power |
| <b>Parallelism</b> | Yes                          | Limited                | Yes            |
| <b>Software</b>    | Operating System             | Bare-metal, RTOS       | Bare-metal     |
| <b>Programming</b> | Any                          | C, C++, Assembly       | HDL            |

Table 3.7: Processor Comparison Table

### 3.1.6 Choosing the Single-Board Computer

#### Raspberry Pi 5

The Raspberry Pi 5 is a single-board computer (SBC) that runs Raspberry Pi OS (a Linux distribution) in its default configuration. It is technically not a microcontroller due to the fact that it runs an operating system and features consumer-facing interfaces such as HDMI and USB. We are including it in our considerations due to the significant computational power necessary to run our auto-mix algorithm.

The Raspberry Pi 5 is the latest and most powerful model from the popular line of Raspberry Pi microcontrollers and SBCs. It features a Broadcom BCM2712 CPU with four Arm Cortex cores that run at up to 2.4 GHz. The Raspberry Pi 5 is available in up to 8 gigabytes of LPDDR4X RAM and features expandable storage through microSD and NVMe interfaces. The Raspberry Pi 5 also includes native support for Bluetooth and WiFi connections with its integrated hardware as well as its software libraries. Some of the relevant IO includes 28 GPIO (general-purpose IO) pins, 9 SPI interfaces, 6 UART interfaces, 7 I2C interfaces, and 2 I2S audio interfaces.

In terms of power, the Raspberry Pi 5 benchmarks well, scoring 72059 on the Embedded Microprocessor Benchmark Consortium's (EEMBC) CoreMark test [22]. This type of power is necessary to handle the auto-mix algorithm which may include auto-encoder and convolutional neural network models with upwards of 2 million parameters. Additionally, the extra power will be useful in developing a fluid GUI (graphical user interface). We desire to use a unified code base for the mobile and embedded GUIs, and thus the computational parity will ensure that performance on each platform will be similar.

Unfortunately, the Raspberry Pi 5 cannot handle all the computational needs of the digital audio mixer. For one, the Raspberry Pi 5 is already a complete system; there is little room to design a PCB to meet the requirements of senior design. Additionally, on a technical level, The Raspberry Pi 5 does not have the necessary amount of I2S connections to take in 8 channels of digital audio from our

ADC. Thus, it is not a viable option for audio DSP. This leaves room for another microcontroller to handle that.

Overall, the Raspberry Pi 5 fulfills enough features for it to be considered for the mixer project. Its ample power as well as its wireless communication capabilities allow it to cover both the AI auto-mix functionality and the wireless communication functionality.

### **BeagleBone AI-64**

The BeagleBone AI-64 is an extremely capable single-board computer from BeagleBoard. It features a Texas Instruments TDA4VM SOC that contains dual Arm Cortex A72 cores that run at up to 2.0 GHz as well as a plethora of accelerators. Among these are a C7x floating point vector DSP chip, a deep-learning matrix multiply accelerator, vision processing accelerators, depth and motion processing accelerators, and more. The main A72 cores benchmark at 24 million DMIPs (Dhrystone Million Instructions Per-Second). The BeagleBone AI-64 is available with 4 gigabytes of LPDDR4 RAM and comes with 16 gigabytes of flash storage which can be expanded through microSD. The BeagleBone AI-64 does not feature any wireless connectivity, however, WiFi capability can be added through the PCIe expansion slot. Some of the relevant IO includes 10 GPIO (general-purpose IO) pins, 11 SPI interfaces, 12 UART interfaces, 10 I2C interfaces, and 12 Multichannel Audio Serial Port (MCASP) interfaces supporting up to 16 channels [23].

The Beaglebone AI-64 is a commendable choice due to its processing power and comprehensive physical IO. It would be an excellent choice for audio DSP. However, similarly to the Raspberry Pi 5, it is a complete system that allows for little room to be done in the realm of PCB and hardware design. Additionally, the lack of wireless connectivity makes it a poor choice for communications. Another consideration is the price. The Beaglebone AI-64 costs approximately 230 dollars [24]. The incorporation of one would put the project well over its intended budget.

### **Nvidia Jetson Nano**

The Jetson Nano is a specialized SBC from Nvidia for use in AI inference. It features a quad-core CPU with Arm Cortex A57 cores, and, crucially, a dedicated 128 core GPU accelerator with 128 Maxwell cores. The Jetson Nano is available with 4 gigabytes of LPDDR4 RAM and comes with 16 gigabytes of flash memory. The Nvidia Jetson Nano does not come with any wireless functionality, however, it can potentially be expanded with wireless adapters through a PCIe connections. Some of the relevant IO includes 2 SPI interfaces, 3 UART interfaces, 4 I2C interfaces, and 2 I2S interfaces.

A particular draw of the Jetson Nano is its GPU accelerator. It is the only SBC in this lineup that comes with NVIDIA Cuda cores; as such, it should be by far the best in terms of AI performance. This makes it an attractive option to run the AI auto-mix models for the mixer.

The Jetson Nano is a solid option for the mixer project. However, it lacks in several key points. It does not include first-party wireless communication solutions. This makes it useless for the communications requirement of the project. Additionally, the physical audio IO is lacking. 2 I2S interfaces only allow for 4 channels of audio at best, which makes it an unviable solution for audio DSP. As such, there is not a use case for the Jetson Nano to cover within the project.

### Final Comparison and Conclusion

Below is a tabular comparison of all the relevant specifications of the compared single-board computers.

| SBC              | Raspberry Pi 5 | BeagleBone AI-64 | Nvidia Jetson Nano |
|------------------|----------------|------------------|--------------------|
| Clock Speed      | 2.4 GHz        | 2.0 GHz          | 1.43 GHz           |
| # Cores          | 4              | 2                | 4                  |
| Other Processors | N/A            | Accelerators     | CUDA Cores         |
| Core Type        | Arm Cortex A76 | Arm Cortex A57   | Arm Cortex A57     |
| RAM              | Up to 8 GB     | 4 GB             | 4 GB               |
| Flash Storage    | External       | 16 GB            | 16 GB              |
| # Audio Inputs   | 2              | 12               | 2                  |
| Price            | \$80           | \$228            | \$129              |

Table 3.8: Single-Board Computer Comparison

The Raspberry Pi 5 was chosen for the mixer project. It was chosen for its processing capabilities as well as its wireless communication capabilities.

### 3.1.7 Choosing the Microcontroller

#### ESP32-S3

The ESP32-S3 is a low-cost yet richly-featured microcontroller from Espressif with several attractive features. The ESP32-S3 includes WiFi and Bluetooth capabilities as well as a competent set of peripheral interfaces. For processing, the ESP32-S3 features a dual core Xtensa LX7 microprocessor with clock speeds of up to 240 Mhz. The ESP32-S3 also comes with 512 KB of SRAM (static random access memory) and an onboard ROM of 384 KB. Both the SRAM and ROM are expandable with external memory devices through protocols such as multi-channel SPI and QPI.

Some of the relevant IO includes 45 GPIO (General-Purpose Input/Output) pins, 4 SPI interfaces (although 2 are reserved for external memory), 3 UART interfaces, 2 I2C interfaces, and 2 I2S audio interfaces.

As for processing power, The ESP32-S3 scores 1181 on the Embedded Microprocessor Benchmark Consortium's (EEMBC) CoreMark test [25]. This is meager compared to the aforementioned Raspberry Pi 5 and also does not fare particularly well against other microcontroller platforms (which will be expounded on in following sections).

One of the attractive development features of the ESP32 series of microcontrollers is its cross-compatibility with Arduino libraries and the Arduino IDE. Arduino is one of the largest embedded platforms, and cross-compatibility with Arduino grants the ESP32 access to many high-quality DSP and networking libraries that would facilitate rapid development.

Unfortunately, the ESP32-S3 has severe deficiencies that make it unsuitable for the project. Similarly to the Raspberry Pi 5, the ESP32 does not have enough channels of I2S to handle all the incoming audio streams from the ADC. This makes it a bad choice for DSP, although given its low cost, it may be viable to use multiple ESP32-S3 units in parallel to perform DSP, with each module handling two channels. Unfortunately, this would complicate our system and increase the scope of our PCB design beyond what is reasonable.

As for graphical performance, the ESP32-S3 does not come with any graphical accelerators (like STM microcontrollers) or bespoke graphical libraries. Given our desire to develop a rather demanding GUI, the ESP32-S3 is not a great choice to power this aspect of the project.

Ultimately, the only real draw of the ESP32-S3 is its price as well as its wireless capabilities. It is not a particularly attractive choice for DSP or GUI. Given that the Raspberry Pi 5 is already covering wireless communications, there is no real place for the ESP32-S3 in our project.

## STM32H747

The STM32H747 is a powerful, full-featured microcontroller from STMicroelectronic's flagship lineup of STM32H7 MCUs. The STM32H747 features a dual-core architecture with an ARM Cortex-M7 running at 480 MHz as well as an ARM Cortex-M4 running at 240 Mhz. The STM32H747 has 1 Mbyte of RAM and supports up to 2 Mbytes of flash memory through a Quad-SPI memory interface that runs at 133 MHz.

Of note is the form factor. The STM32H747 comes in a ball-grid array package. These types of packages are impossible to solder by hand and must be assembled with an automated pick-and-place machine. This complicates the development process; if a problem is found with the board, the whole MCU and board must be thrown out; it is not feasible to remove and reattach the MCU to a different PCB.

Concerning processing power, the STM32H747 packs a real punch, scoring 3224 on CoreMark, or an equivalent 1327 DMIPs [26]. Additionally, the MCU possesses 4 DMA (direct memory access) controllers to offload memory operations from the CPU. This is crucial to real-time audio processing. For graphics, the STM32H747 possesses an integrated LCD controller, a Chrom-ART graphical hardware accelerator to reduce CPU load, and an accelerated hardware JPEG

codec. Overall, the MCU is a solid option for both DSP and GUI purposes.

Some of the relevant IO includes up to 168 GPIO (General Purpose Input/Output), 6 SPI interfaces with 3 multiplexed I2S channels, 4 UART interfaces, 4 I2C interfaces, and 4 SAI (Serial Audio Interface) interfaces. Of note is the fact that SAI interfaces are stereo, so 4 channels of SAI can theoretically handle 8 mono audio streams. This is in addition to the 3 I2S audio interfaces. As such, the STM32H747 features adequate IO for audio DSP purposes.

Unfortunately, there is still the concern of raw processing power and massive parallelism with eight-plus channels of real-time audio processing. The STM32H747 is only dual-core, and it would be a difficult task to optimize the DSP effects to operate across 8 channels in real time with minimal audio artifacts. Again, it would be possible to run several MCUs in parallel for DSP, but that is a complicated task that is beyond the scope of this project.

In conclusion, the STM32H747 is a solid choice to power the GUI portion of this project. However, the Raspberry Pi 5 is already handling the GUI, so there is no need for the STM32H747 in our project. If this project were to be expanded on in the future, it may be a good idea to combine the GUI capabilities of the STM32H747 with the wireless capabilities of the ESP32-S3 to provide a cheap yet powerful alternative to the Raspberry Pi 5.

## XMOS XU316-1024-TQ128

The XMOS XU316-1024-TQ128 is a full-featured, powerful MCU from XMOS's xcore.ai lineup of microcontrollers. The XMOS possesses a unique, multi-core architecture with 16 logical cores split into two "tiles" (groups) and a deterministic, hardware scheduler called xTIME that mimics the behavior of a real-time operating system (RTOS). The logical cores have high-speed interconnects both between the cores on the chip and with cores on other SoCs. The logical cores themselves are custom RISC cores. The XMOS possesses 1 MB of onboard SRAM. The XMOS has no onboard flash memory, but external memory can be connected through Quad-SPI and SPI protocols.

Again, the MCU package is of note. The XMOS XU316-1024-TQ128 comes in a TQFP-128 package with a 0.4mm pin pitch. The package is leadless unlike the ball-grid array of the STM32H7 microcontroller; however, the minuscule pin pitch and high pin count still make it difficult for hand-soldering. This high pin-density design may necessitate the development of a breakout board for the chip to be later connected to the main PCB. This alleviates troubleshooting woes due to removing and reattaching the MCU.

Another consideration is the power topology and overall integration complexity. The XMOS is unique in that it requires a 1v rail for core functionality, as well as 1v8 and 3v3 rails for peripherals. Additionally, the XMOS has power sequencing requirements. The peripheral rails must be brought up before the core 1v rail; then, the reset can be released the the XMOS can boot. This introduces extra complexity to our design. Fortunately, the XMOS features extensive documentation and even full schematics of their development boards intended to be used as reference

designs. The XMOS may feature extra complexity, but it compensates with robust documentation.

Concerning processing power, the MCU runs at 600 MHz and provides 2400 MIPS of total processing throughput. This is substantially greater than all the microcontrollers covered thus far and provides an adequate amount of headroom for multi-channel audio DSP. Of course, there is also the aforementioned multi-core architecture of the XMOS MCU. This multi-core architecture means that each audio stream can have a dedicated processing pipeline. This is an incredible feature and will make multi-channel audio programming much easier; there will be no need to manually implement a scheduling algorithm to process each channel of audio in sequence. This makes the XMOS MCU an outstanding option for audio DSP. The extra cores can be used to handle IO, which may include communications with the Raspberry Pi 5 as well as other hardware peripherals directly connected to the XMOS.

In addition to its raw processing power and multi-core architecture, the XMOS features a hardware vector unit for the acceleration of AI (Artificial Intelligence) tasks such as classification and inference. The XMOS was considered to run the AI auto-mix algorithm instead of the Raspberry Pi 5. Unfortunately, with some rudimentary size calculations, the model size would be in the realm of 10 Mbytes. The XMOS only comes with 1 MByte of RAM, so most of the model would have to be offloaded to external flash memory which operates at a slower pace. It remains to be seen whether or not the XMOS would process the AI auto-mix algorithm at a reasonable pace, but for now, the Raspberry Pi 5 has been chosen to run the algorithm. If the XMOS proves to be sufficient to run the auto-mix algorithm, a future MCU topology could include the XMOS MCU for DSP and AI purposes, an STM32H7 MCU for GUI and state management, and an ESP32 or dedicated wireless module for communications.

In regards to IO, the XMOS is also unique in that IO pins can be configured in software to run different serial communication protocols. The available IO is defined by the developer and not by the hardware, which makes it incredibly useful for our multi-channel audio DSP purposes. It is possible to implement eight-plus channels of digital audio input through I2S with the first-party serial communication libraries.

Overall, the XMOS XU316-1024-TQ128 is an incredibly compelling MCU for audio DSP purposes and may even be a suitable option for AI auto-mix processing. Its raw power as well as its multi-core architecture will make multi-channel processing much easier. As such, the team has decided to incorporate this MCU into the design alongside the Raspberry Pi 5.

## Final Comparison and Conclusion

Below is a tabular comparison of all the relevant specs of the compared microcontrollers.

The final processor selection is a dual-processor topology with a Raspberry Pi 5 and an XMOS XU316 MCU. The Raspberry Pi will handle the GUI, wireless

| <b>Processor</b>      | <b>ESP32-S3</b>    | <b>STM32H747</b>  | <b>XMOS XU316</b> |
|-----------------------|--------------------|-------------------|-------------------|
| <b>Clock Speed</b>    | 240 MHz            | 480 MHz, 240 MHz  | 600 MHz           |
| <b># Cores</b>        | 2                  | 2                 | 16                |
| <b>Core Type</b>      | Xtensa LX7         | ARM Cortex M7, M4 | Custom RISC       |
| <b>RAM</b>            | 512 KB             | 1 MB              | 1 MB              |
| <b>Flash Storage</b>  | 384 KB, expandable | 1 MB, expandable  | External          |
| <b># GPIOs</b>        | 45                 | 168               | 78                |
| <b># Audio Inputs</b> | 2                  | 7+                | User-defined      |
| <b>Package</b>        | Module             | BGA               | TQFP              |
| <b>Price</b>          | \$1.85             | \$16.53           | \$15.67           |

Table 3.9: Microcontroller Comparison

communications, and the AI auto-mix algorithm due to its immense power and ample memory. The XMOS XU316 will handle audio DSP due to its ample power and audio I/O as well as its easy-to-program, multicore architecture.

### 3.1.8 Clock

A clock system is necessary to every digital system. In this section, the clock requirements of the system will be established and various clock methods will be discussed.

#### Clock Requirements

The

### 3.1.9 Flash Memory

### 3.1.10 Power Supply

#### Prelude: Power Requirements and Topology

A major part of powering a mixer is providing sufficient voltage levels to the various components within the board. There are several key requirements that the power supply system must meet, dictated by the components selected for the mixer.

For one, a high-current 5 volt rail is required to power the Raspberry Pi 5. This voltage rail will also power the audio circuitry on the board in tandem with a voltage inverter that outputs -5 volts.

One may ask why a dual-ended supply is needed. Most preamplifier ICs (and operational amplifiers in general) operate off a dual-supply system. It is possible to operate them off a single supply, but it falls on the designer to create a robust virtual

ground solution which significantly increases the design complexity of the system. Virtual grounds are easy to make in theory with a resistor divider; however, any significant current draw biases the virtual ground away from the center of the single supply range such that the amplifier may clip. More robust virtual ground solutions utilize dedicated rail splitter ICs, however, they are limited in the amount of current they can provide. A dual-ended voltage rail allows for the audio signals on the board to oscillate around ground, negating the need to float the audio signals with a DC offset and simplifying the overall design.

As for other necessary voltage rails, 3.3 volt and 1.8 volt rails must be supplied to power the numerous peripherals that are necessary for the mixer's operation, things such as the AD/DA circuitry and IO control. One additional requirement imposed on the power requirements is the necessity of a 0.9 volt rail to power the XMOS xcore microcontroller. The last requirement is a high-voltage rail to provide phantom power to microphones. The high-voltage phantom rail can come in a variety of different voltages and will be expanded upon in later sections.

Such is the voltage topology of the project. Now that the voltage requirements have been established, it is necessary to discuss the implementation of the voltage supplies. Several components and technologies are involved. The following sections will expound on the power supply unit (PSU) and different types of voltage regulators, linear regulators and switching regulators.

## Power Supply Unit

The first component to consider in powering a piece of electronic equipment is the DC power supply unit. Power supply units are confusing to define because they are distinct from voltage regulators, but they may or may not contain voltage regulators. In the context of consumer electronics, DC power supply units convert the high voltage output of wall outlets into a voltage that can be utilized by the piece of equipment, usually with a transformer. This voltage may be further converted by voltage regulators within the device or simply clamped steady.

For the conversion from wall power, the 120V AC signal must be converted to DC and stepped down. This requires the use of a transformer followed by rectification— where the transformer (using different coil and winding ratios) steps the voltage up or down depending on the purpose and subsequently adjusts current output. Rectification is the method that is used to turn AC power into pseudo-DC power, most easily done with a full-bridge rectifier of four diodes.

The voltage output of the DC power supply in the case of the mixer will likely be 15 or 12 volts. These are a reasonable voltage to deliver the amount of power needed while keeping the overall current draw to a reasonable quantity. For the mixer, the power requirement is estimated to be under 75 watts. When delivering power with 15 volts, for example, the current draw is under 5 amps, which is reasonable for a consumer power supply. Once the high AC voltage wall has been stepped down to a reasonable operating voltage, the voltage must be further converted to power the different components of the system.

Upon initial research, it was discovered that there are a plethora of triple output

supplies that can supply a dual-ended voltage rail on the order of 10-15 volts and a high current 5 voltage rail. This provides a great solution to the power requirements of the mixer. A minimal amount of conversion is needed; only the 3.3 volt, 1.8 volt, and 0.9 volt rails need to be provided with further voltage regulation

Below is a comparison of some power supplies that meet the required specifications.

| Part                    | Power (W) | Rails                                  | Efficiency | Price   |
|-------------------------|-----------|----------------------------------------|------------|---------|
| <b>Mean Well RT50-B</b> | 50        | 5V @ 4A,<br>12V @ 2A,<br>-12V @ 0.5A   | 77%        | \$32.94 |
| <b>Mean Well RT50-C</b> | 50        | 5V @ 4A,<br>15V @ 1.5A,<br>-15V @ 0.5A | 78%        | \$33.91 |
| <b>Mean Well RT65-B</b> | 65        | 5V @ 5A,<br>12V @ 2.8A,<br>-12V @ 0.5A | 77%        | \$38.89 |

For the purposes of this project, the Mean Well RT65-B is the most suitable. It has the highest power output and the greatest amount of current available from its voltage rails. Its 5 volt 5 amp rail is sufficient for the Raspberry Pi and the 2.8 amps available from the 12 volt rail should be sufficient to power our 8+ preamp circuits.

It is worthwhile to mention that an alternative to this is to procure an off-the-shelf consumer power supply for existing mixers. Some of them are able to provide both 5V and  $\pm 15V$  at different amperages, such as the power supply of the aforementioned TouchMix mixer. This would negate the problem of not being able to efficiently supply the positive and negative rails of the operational amplifiers OPA2134's and cut down on the overall footprint of the PCB, making the design of the circuit significantly smaller. The problem with this approach is that it can be very costly, with most power supplies of this ilk costing upwards of \$100. With the cost comes less complexity on the overall board itself, but a detraction from the general usability of the mixer itself. The user must carry around a bulky, custom external supply versus plugging a power cord into the mixer and utilizing a relatively cheap, built-in power supply.

## Linear Regulators

Once the high-voltage wall power has been stepped down to a usable DC voltage, the voltage must be further stepped down and regulated to provide power for certain components in the system. One of the technologies with which this is accomplished is linear regulators.

Linear regulators utilize a very simple concept to step voltages down—dissipate any unwanted output voltage as heat, which can be done through a variable resistor. As such, linear regulators are only capable of lowering the output voltage rather

than also being able to raise the voltage. As expected with this type of hardware, it is very simple, requiring very little besides a few capacitors and resistors (plus transistors and Op-Amps) and thus very cost-efficient, being ideally less than \$1. Figure 3.9 exemplifies its simplicity, with very little in the way of active components.

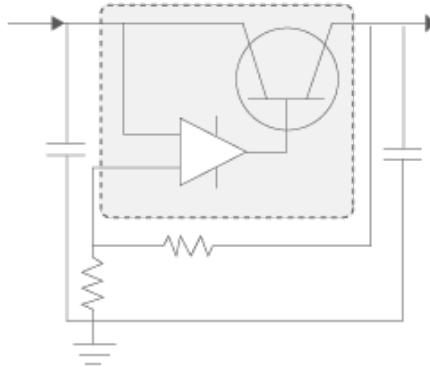


Figure 3.9: Basic Linear Regulator, Reproduced with Permission. © Renesas Electronics Corporation or one of its subsidiaries. All Rights Reserved.

In addition, linear regulators have very little noise present when being used, which is especially crucial when using analog signals.

In this case, since we would be using an IC as the regulator (e.g. TPS7A53B), it would reduce the overall complexity of the device with better performance compared to the basic transistor and Op-Amp setup. In figure 3.10, one can see that the only additional parts added to the IC are decoupling capacitors and a resistor ladder that can alter the output voltage [27]. This is effective because of the very low power required to use such an IC, but the efficiency decreases drastically the further away the desired output voltage is compared to the input voltage.

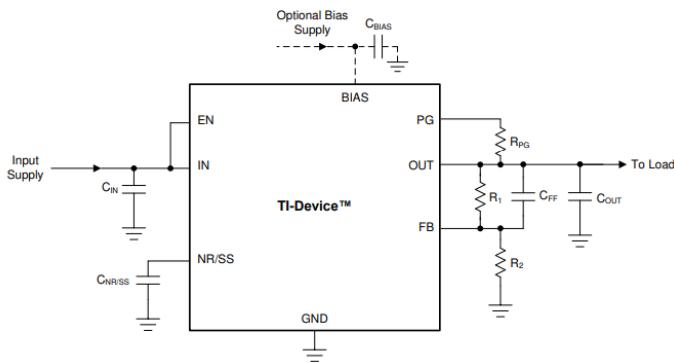


Figure 3.10: Example Application of TI's Linear Regulators, Courtesy of Texas Instruments Inc

## Linear Regulator Comparison

In choosing a linear regulator, the most desirable characteristic is the minimal number of external components required. There are numerous linear regulators that have the same external current output and voltage output, but as shown in table 3.10, we chose the NCP1117DT50RKG due to the number of external components required being only two.

| Part               | Output Voltage | Output Current | Price  | Ext. Parts |
|--------------------|----------------|----------------|--------|------------|
| NCP1117DT50RKG     | 5V             | 1A             | \$0.60 | 2          |
| SPX1117M3-L-5-0/TR | 5V             | 800mA          | \$0.36 | 4          |
| LD1117S50TR        | 5V             | 800mA          | \$0.44 | 3          |
| TLV1117-50CDCYR    | 5V             | 800mA          | \$0.46 | 4          |

Table 3.10: LDO Comparison Chart

## Switching Regulators

Another type of technology used to regulate voltages is switching regulators. As stated before, switching regulators require more complicated circuitry, but are more efficient than linear regulators. As the name suggests, switching regulators achieve the desired output voltage by switching between off and on and using an output inductor to supply the energy through the circuit when off. This can be done either synchronously or asynchronously using FETs. As shown in figure 3.11, both the internal and external circuitry is more robust than that of a linear regulator.

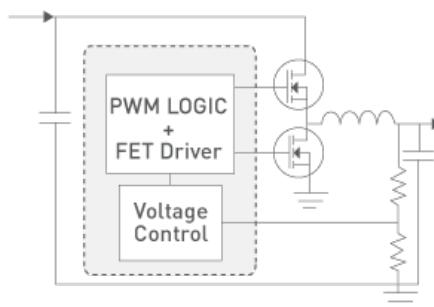


Figure 3.11: Example Switching Regulator Schematic, Reproduced with Permission. © Renesas Electronics Corporation or one of its subsidiaries. All Rights Reserved.

Unlike linear regulators, switching regulators are capable of outputting either a lower (Buck converter) or higher voltage (Boost converter), with some being able to do Buck and Boost conversion. While linear regulators become more inefficient the further away the desired output voltage is compared to the input voltage, switching

regulators do not have this problem. This subsequently allows the desired output voltage to be much lower and/or higher than the input voltage. Additionally, switching converters are able to supply much more power through the devices, such as 2A through the TPS62822 [28]. The main downsides of using these types of regulators, though, are noise due to switching (much higher than any noise created by a linear regulator) and the cost being higher than a linear regulator due to complexity [29].

| VRM Type              | Linear Regulator                      | Switching Regulator                                         |
|-----------------------|---------------------------------------|-------------------------------------------------------------|
| <b>Output Voltage</b> | Lower (Buck)                          | Lower (Buck)<br>Higher (Boost)<br>Lower/Higher (Buck/Boost) |
| <b>Cost</b>           | Low                                   | Medium                                                      |
| <b>Noise</b>          | Low                                   | Medium                                                      |
| <b>Complexity</b>     | Low                                   | Medium<br>(Depends on IC)                                   |
| <b>Efficiency</b>     | Medium<br>(↓ as $V_{in} - V_{out}$ ↑) | High                                                        |

Table 3.11: Comparison Table of Linear vs. Switching Regulators

### Switching Converter Comparison

The main factors in choosing the switching (buck) converters for our mixer came down to the overall current limit, switching frequency and voltage range. Due to needing to power multiple microcontrollers in addition to ADCs and DACs, we require that our power supplies be able to withstand around 2A of total current. The Raspberry Pi, due to its immense current drain, will need to have its own power supply, which makes the overall current limit for the other devices much lower. If there was only 1 set of power supplies, then all of the circuitry inside the power supplies would have to be able to withstand upwards of 7A, which would be a costly upgrade in addition to needing to allow for a large amount of heat dissipation. We have chosen two separate buck converters on the grounds of needing one for a low-footprint 5V supply. The TPS82140 includes an integrated inductor in the package itself, which mitigates space required to use it on the PCB. The switching frequency of the regulator is also significantly higher than others in the market, which allows for the regulator to work more effectively. The TPS62822, while not including an internal inductor, can operate in a low voltage range at a high switching frequency while remaining very low-cost. Both of these ICs are able to withstand 2A, with other versions being able to take 3A.

| Part                 | Current Limit | Switch Freq. | Voltage Range | Price  |
|----------------------|---------------|--------------|---------------|--------|
| <b>LM53602AMPWPT</b> | 2A            | 380kHz       | 3.5-36V       | \$5.64 |
| <b>BD9325FJ-E2</b>   | 2A            | 380kHz       | 4.75-18V      | \$2.49 |
| <b>TPS82140SILR</b>  | 2A            | 2MHz         | 0.9-6V        | \$4.72 |
| <b>TPS62822DLCR</b>  | 2A            | 2.2MHz       | 0.6-4V        | \$1.16 |

Table 3.12: Buck Converter Comparison Table

## 3.2 Software Research and Comparison

The software systems of the digital audio mixer will span several key areas, namely, audio DSP effects, inter-system communications, app design, and machine learning in the form of the auto-mix algorithm. The research is presented here.

### 3.2.1 Audio DSP Effects

#### Prelude: Digital Audio Representation

Digital Signal Processing involves the processing of sampled, quantized signals. Analog signals are sampled at a constant rate and quantized to a certain granularity (e.g. 10 bits of quantization → 1024 possible values).

One of the basic principles underlying digital signal processing is the Nyquist frequency. Any analog signal can be adequately sampled as long as the sampling frequency is twice that of the highest frequency present in the analog signal. In the context of audio, a typical sampling frequency is 44.1 kHz. This is due to the fact that the range of human hearing lies approximately in the range of 20 Hz to 20 kHz. In order to adequately sample the audio, the sampling frequency must be above the Nyquist frequency of  $2 * 20 \text{ kHz} = 40 \text{ kHz}$ . For this project, the minimum common sampling frequency of 44.1 kHz is selected to reduce processing requirements.

Another convention in audio DSP relates to the range of possible sample values. In most applications, audio is presented as a sequence of values ranging from -1 to 1 representing the displacement of the audio driver. The granularity of the quantization within that range is dependent on the ADC.

In summary, the overall software representation of audio in the context of this mixer project will be a sequence of displacement values ranging from -1 to 1 that is sampled at 44.1 kHz (44,100 times per second).

#### Prelude: DSP Operators

Given a sequential array of float values sampled at a constant rate, there are several key DSP operators that can be used to modify the signal and achieve desired audio effects.

One operator is addition, or summing. Different streams of audio can be added together with simple addition. It is represented with the symbol shown below.

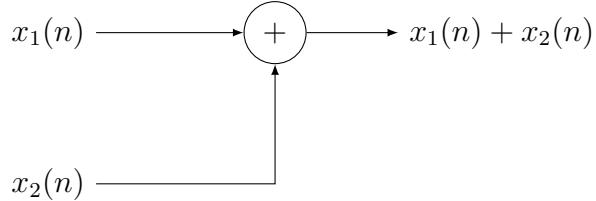


Figure 3.12: The Addition Operator

Another operator is the multiplication operator. The multiplication operator involves the multiplication of values in a stream by some constant. The operator is represented by the triangle symbol shown below. The value shown in the triangle is the multiplication constant.

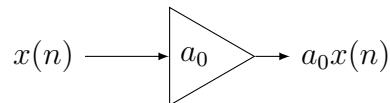


Figure 3.13: The Multiplication Operator

One last operator is the delay operator. The delay operator involves the delaying of a sample by a sampling period. The operator is represented by the symbol shown below. The exponent value of the  $z$  symbol represents the number of sampling periods of the delay. It is typically 1.

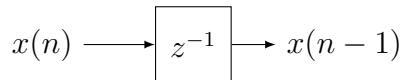


Figure 3.14: The Delay Operator

Together, these operators perform all the transformations on audio signals necessary to produce higher-level audio effects. The audio effects shown below use such operators to achieve the desired output.

## Gain

Gain (specifically, digital gain) is the simplest DSP audio effect and is used as a standalone effect and as a building block of other effects. It involves a simple scalar multiplication of an audio stream. It is important to make a distinction between digital gain and analog gain. Analog gain is the electrical gain of the preamp system applied to the analog audio signal before it is converted to a digital signal by the ADC. It is controlled by hardware rather than by software. It is also important to

note that analog gain may introduce irreversible clipping from the operational amplifiers which cannot be fixed by simply bringing down the digital gain.

Digital gain may be applied in many places. It may be applied to the signal path at the end of the signal chain or it may be applied by other effects modules (e.g. a compressor "makeup gain"). It is simple but useful.

## Pan

Panning is a DSP audio effect that adjusts the balance between left and right audio channels so that the sound is placed in a certain place within a stereo field. The effect is simple; gain is applied to the left or right channel to impact the channel balance and perception of location. However, there are several details to be considered.

One must consider how the adjustment of balance operates so that the overall perception of volume is the same. If the left and right channels linearly changed in volume as the panning swept across the sound field, there would be an increase in volume at the center position compared to a pan of left or right due to the summation of the two channels. In order to alleviate this issue, mixers and audio workstations implement something called a "pan law." A pan law dictates the attenuation of an audio signal as it sweeps across the stereo field in order to maintain a constant sound pressure level. A common pan law dictates that both audio channels will be attenuated by 3 dB at the center panning position compared to a hard pan of left or right [30].

### 3.2.2 FFT Signals: Importance and Application

While searching for a means to visualize our audio signals, we found that the industry standard is to use Fast Fourier Transform (FFT) signals. FFT signals represent a transformation of an input signal from the time domain into the frequency domain through the application of the Fast Fourier Transform algorithm. The FFT algorithm decomposes a signal, which has been sampled over a specific period of time, into its constituent frequency components.

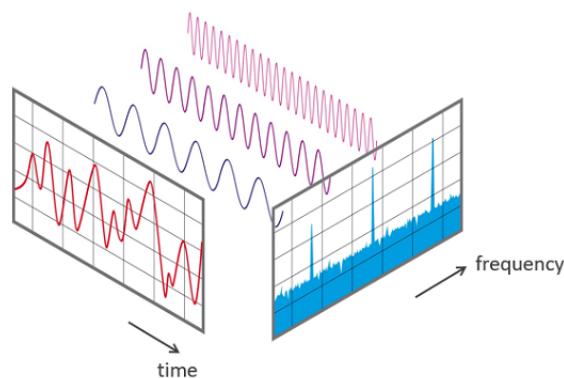


Figure 3.15: Fast Fourier Transform Algorithm

This transformation enables the visualization and analysis of the frequency content of a signal, allowing a user to see which frequencies are present as well as their associated intensities. By mapping frequency against volume in this way, we are able to observe and display the frequency spectrum of a piece of audio.

### 3.2.3 Comparison of Communication Protocols

#### Prelude: Communication Protocol Requirements

In the context of real-time data transmission between the xcore microcontroller and the Raspberry Pi, the choice of an appropriate communication protocol is paramount. A few requirements must be met. To start, real-time data synchronization is crucial. As audio gets sent from the xcore to the Raspberry Pi, it must be analyzed in real-time with little to no jitters or drop-outs. In addition, high-speed data transfer rates are necessary, as large amounts of data will be sent from the xcore to the Raspberry Pi. SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), UART (Universal Asynchronous Receiver-Transmitter), and Ethernet are the four widely used communication protocols that are examined in depth in this section. The choice of SPI for the current project is then supported by an analysis of the particulars of each protocol, including latency, data transfer speed, system complexity, and communication range.

#### Serial Peripheral Interface (SPI)

The main factors in choosing SPI as our primary communication protocol came from the fact that SPI is a synchronous, full-duplex communication protocol that can be employed for high-speed, short-distance data transfer between integrated circuits in embedded systems. Seeing as audio information will be communicated between the xcore and Raspberry Pi in real-time, it was of the utmost importance that the protocol we utilized would be able to support this data transfer with minimal to no loss. The protocol operates on a master-slave architecture, where the master (Raspberry Pi) will control the slave device (xcore). Communication is facilitated through four dedicated lines: MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCLK (Serial Clock), and CS (Chip Select). The clock line is driven by the master to synchronize data transfer, while the Chip Select is used to select the active slave device at any given time.

The full-duplex nature of SPI allows data to be transmitted and received simultaneously, which allows for efficient real-time communication. This characteristic is critical for our application of real-time Fast Fourier Transform (FFT) data transfer, where continuous communication between devices is necessary to ensure timely processing and display of data.

One of the most significant advantages of using SPI is its high communication speed. This is necessary for our application, as we will require low-latency, high-throughput communication for the real-time processing and transmission of FFT data.

## **Inter-Integrated Circuit (I2C)**

The Inter-Integrated Circuit (I2C) protocol employs a half-duplex, synchronous communication method, where data is transmitted using two bidirectional lines: SDA (Serial Data Line) and SCL (Serial Clock Line). I2C supports multiple masters and multiple slaves on the same bus.

While I2C's simplicity in wiring is advantageous in systems with limited I/O pin availability, this protocol suffers from certain drawbacks that make it less suitable for high-speed, real-time applications such as ours. First, I2C is significantly slower than SPI, with typical data rates ranging from 100 kbps in standard mode to 400 kbps in fast mode, and up to 3.4 Mbps in high-speed mode. Even at its maximum speed, I2C cannot match the high throughput and low latency provided by SPI. This limitation is particularly hindering to our needs.

Moreover, I2C introduces additional overhead due to its addressing scheme. Each I2C transaction involves the transmission of an address byte, which identifies the target slave device. This additional data increases the communication overhead and reduces the effective throughput of the protocol. Additionally, I2C is a half-duplex protocol, meaning that data can only flow in one direction at a time. For the purposes of our project, real-time, continuous data exchange between the xcore and the Raspberry Pi without delays is severely degraded.

## **Universal Asynchronous Receiver-Transmitter (UART)**

The Universal Asynchronous Receiver-Transmitter (UART) protocol was another possible communication option. UART operates in an asynchronous, full-duplex manner, meaning that it does not require a shared clock line between devices. Instead, it relies on start and stop bits to synchronize data transmission.

UART is typically used for point-to-point communication, where two devices exchange data using two communication lines: TX (Transmit) and RX (Receive). UART suffers from significant limitations in terms of speed and efficiency. Standard UART implementations typically operate at speeds of up to 1 Mbps, though some high-speed variants can achieve higher data rates. Nevertheless, even at its highest speeds, UART is considerably slower than SPI, making it less suitable for FFT data transmission.

Furthermore, UART introduces additional overhead due to the inclusion of start and stop bits for each byte of data transmitted. This overhead reduces the effective throughput of the protocol and increases latency, which is undesirable for our system. Given these factors, UART was not a viable option.

## **Ethernet**

Ethernet is a high-speed, packet-based communication protocol. Ethernet operates in a full-duplex, synchronous manner and supports data transmission rates of 10 Mbps, 100 Mbps, and 1 Gbps, which makes Ethernet an attractive option for our system which requires a large amount of data to be transferred.

However, Ethernet's complexity and reliance on packet-based transmission make it less suitable for real-time communication. Ethernet introduces variable latency's due to the nature of its packet-switched network architecture, where data is broken into packets and reassembled at the receiving end. While this architecture is highly effective for long-distance communication and networking, it can introduce unpredictable delays that are unacceptable in our use case of audio processing.

Furthermore, Ethernet requires a more complex hardware and software stack, which adds unnecessary complexity where simpler protocols like SPI can achieve the desired performance. Given the short-distance, high-speed communication requirements of our task at hand, Ethernet's advantages in long-distance communication are not applicable.

### **Justification for Selecting SPI**

After a thorough evaluation of the communication protocols, SPI was selected as the most appropriate choice for this project due to its superior speed, low-latency characteristics, and ability to handle real-time data transmission. The high-speed communication provided by SPI ensures that the large volumes of FFT data and raw audio generated in this project can be transmitted efficiently between the xcore and the Raspberry Pi. Additionally, the full-duplex nature of SPI allows for simultaneous data transmission and reception, which is essential for our real-time audio processing needs.

While other protocols, such as I2C, UART, and Ethernet, offer certain advantages in specific use cases, they each present limitations that make them less suitable for this particular application. I2C's lower data rates and half-duplex operation introduce unacceptable delays, while UART's slower speed and additional overhead make it impractical for real-time communication. Ethernet, although highly capable in terms of speed, introduces unnecessary complexity and latency due to its packet-based nature.

### **3.2.4 Analysis of Protocols for Raspberry Pi to Mobile Data Transmission**

In the design and implementation of our digital mixer, choosing the appropriate communication protocol to transmit data between the Raspberry Pi and mobile application is crucial. This decision hinges on several factors including the need for real-time data transmission, bandwidth constraints, network reliability, and ease of integration. This section provides a comprehensive comparative analysis of the most relevant communication protocols that can be utilized to transmit Fast Fourier Transform data between the Raspberry Pi and a mobile application, detailing the strengths, limitations, and ideal use cases of each protocol, and why we ultimately chose WebSockets.

## WebSockets

WebSockets are a full-duplex communication protocol that provides a persistent connection between a client and a server, enabling continuous data transfer. In the context of our project, WebSockets are particularly well-suited for real-time communication between the Raspberry Pi and the mobile application due to their ability to transmit data bidirectionally without the need to repeatedly establish new connections [31]. WebSockets establish a connection between the server (Raspberry Pi) and the client (mobile app) through an initial HTTP handshake, after which the connection is upgraded to a WebSocket protocol. Once established, this connection remains open, allowing both the server and the client to continuously send and receive data without re-establishing the connection. For our project, this enables the Raspberry Pi to stream FFT data to the mobile app as soon as new data becomes available [32].



Figure 3.16: WebSocket Connection Steps

WebSockets excel in low-latency environments where real-time data transmission is required, which makes it ideal for our purposes of transmitting FFT signals in real-time to the app. In addition, since the connection remains open, WebSockets minimize the overhead typically associated with protocols that require the establishment of new connections for each data transfer, such as HTTP/REST APIs.

## HTTP/REST APIs

HTTP/REST APIs are a commonly used communication protocol for client-server interactions. In particular, HTTP is a request-response protocol that is stateless, meaning that each request is independent of previous requests. This architecture is simple and widely supported, making it a popular choice for many applications [33]. In an HTTP/REST setup, the mobile application would send an HTTP GET request to the Raspberry Pi, which acts as a server, to retrieve the FFT data. The server processes the request and responds with the requested data in the form of a JSON. The mobile app would then display this data after processing the JSON parameters.

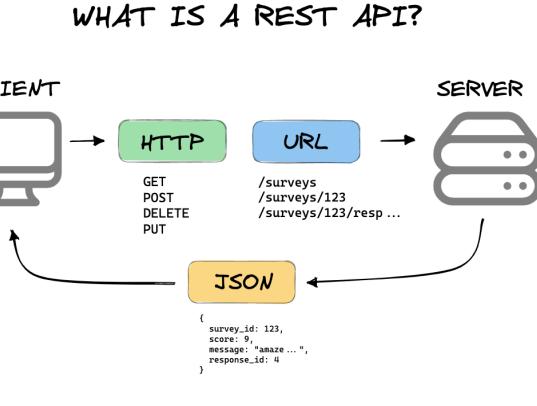


Figure 3.17: HTTP/REST API Connection Steps

HTTP is universally supported and easy to implement, which made it an attractive choice for our project initially. Since REST APIs are commonly used, they can be easily integrated with existing tools and frameworks, simplifying the development process further. However, the largest flaw we observed in HTTP/REST APIs was the fact that each data transfer requires a new connection to be established, leading to increased latency compared to protocols like WebSockets. In addition, HTTP's request-response model is less efficient for real-time applications where data needs to be streamed continuously, making it nonoptimal for our purposes, where we needed a constant, open communication stream.

### Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight, publish-subscribe protocol designed for low-bandwidth, high-latency, and unreliable networks. It is widely used in Internet of Things applications, where devices need to send data over constrained environments [34]. In an MQTT setup, the Raspberry Pi publishes the FFT data to an MQTT broker under a specific topic. The mobile app, acting as a subscriber, listens for messages published to this topic. When the Pi sends new data, the broker pushes it to all connected subscribers. The broker manages all incoming messages by receiving them, applying filters, and determining the intended recipients based on their subscriptions. It then forwards the relevant messages to the subscribed clients. Additionally, the broker maintains session data for persistent clients, which includes tracking their subscriptions and storing any messages they missed during disconnection.

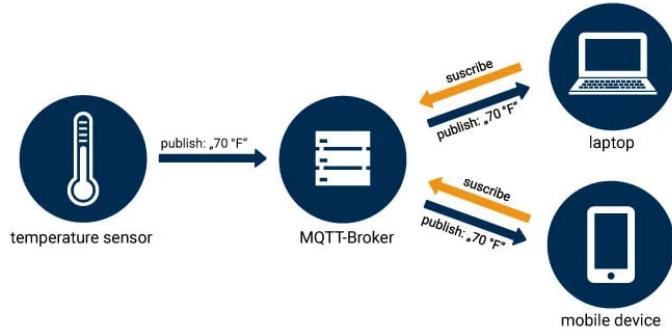


Figure 3.18: MQTT Connection Steps

MQTT is optimized for low-bandwidth environments, which makes it ideal for mobile applications such as ours that may experience intermittent connectivity. In addition, since MQTT implements a publisher-subscriber model, it enables scalable communication between multiple devices. MQTT supports different levels of delivery guarantees, ensuring that messages are delivered, which gives us a sense of security knowing that our data will be transmitted over. However, by implementing MQTT, we would need to implement the MQTT broker, which would add complexity to our project, which we are trying to minimize as much as possible. In addition, MQTT is optimized for small messages, which poses an issue for us, as FFT data is normally quite large.

### Constrained Application Protocol (CoAP)

CoAP is another lightweight protocol designed for devices, such as the Pi, similar to HTTP but optimized for Internet of Things use cases. CoAP uses UDP instead of TCP, reducing overhead and increasing speed in certain environments. In addition, CoAP employs a request-response model, similar to HTTP, except it makes modifications specifically for a constrained network such as an Internet of Things device. CoAP employs the concept of observe, which allows devices to subscribe to changes that occur, instead of continuously polling for updates and wasting resources and wasting battery [35]. CoAP operates over UDP, providing faster communication. For example, the Raspberry Pi would act as a CoAP server, and the mobile app would send CoAP requests to fetch FFT data.

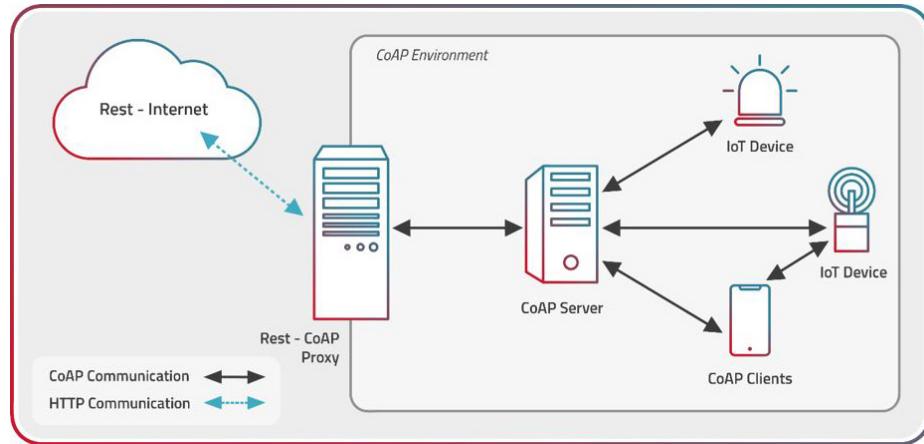


Figure 3.19: CoAP Connection Steps

CoAP's use of UDP instead of TCP makes it faster and more efficient in constrained environments. In addition, since it was designed specifically for low-power, low-bandwidth IoT devices, CoAP has minimal overhead compared to more heavyweight protocols like HTTP or WebSockets. However, since CoAP uses UDP, it does not guarantee delivery, making it less reliable than TCP-based protocols. CoAP is also less widely supported compared to WebSockets, which can make integration with existing platforms more challenging and eliminated CoAP as a viable option. Of particular importance to us is that our data is received by the client (mobile application / on-board LCD), or we may see inaccurate visualizations of the audio signals.

### Justification for choosing WebSockets

After conducting comprehensive research to come up with a decision for this project, the best option will be WebSockets for real-time, low-latency communication, especially for streaming FFT data continuously from the Raspberry Pi to the mobile app. HTTP/REST could be used in simpler cases where periodic updates are sufficient, while MQTT offers a lightweight solution for environments with constrained resources or bandwidth limitations.

| <b>Protocol</b>                     | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WebSockets                          | WebSockets provide a continuous, full-duplex communication channel between a client and a server, ideal for real-time applications. After an initial HTTP handshake, the connection is upgraded to WebSocket protocol, which remains open and supports bidirectional data flow. It ensures low-latency transmission of FFT signals, facilitating real-time data exchange between the Raspberry Pi and the mobile app.                                                                                                                                                                                                                                                                                                                                                                                         |
| HTTP / REST APIs                    | HTTP/REST APIs are popular for client-server communication due to their stateless request-response model and broad compatibility. In our setup, the mobile app would request FFT data from the Raspberry Pi via HTTP GET, receiving JSON responses. However, each request requires a new connection, adding latency and making HTTP inefficient for continuous data streaming, which is essential for our real-time needs.                                                                                                                                                                                                                                                                                                                                                                                    |
| Message Queuing Telemetry Transport | MQTT is a lightweight, publish-subscribe protocol ideal for IoT due to its efficiency in low-bandwidth and high-latency environments. In our project, the Raspberry Pi would publish FFT data to an MQTT broker, and the mobile app would receive updates as a subscriber. The broker manages message delivery and reliability options, making MQTT scalable and resilient for intermittent connections. However, the protocol's design for small messages challenges our larger FFT data needs, and setting up an MQTT broker adds complexity to the project.                                                                                                                                                                                                                                                |
| Constrained Application Protocol    | CoAP is a lightweight protocol designed for IoT applications, offering a request-response model similar to HTTP but optimized for constrained devices by using UDP instead of TCP. This reduces overhead, making it faster and more efficient for low-power, low-bandwidth environments like IoT. CoAP supports an "observe" feature, allowing clients to subscribe to updates without needing to poll continuously, conserving resources and battery life. However, since CoAP lacks TCP's delivery guarantees, it is less reliable, which can result in missed data and inaccurate visualizations. Additionally, CoAP's limited support on platforms compared to more widely adopted protocols like WebSockets reduces its viability for our project, where reliable, real-time data transfer is essential. |

Table 3.13: Table of Communication Protocols

### 3.2.5 Wireless Data Level Protocols

In our project, Wi-Fi serves as the essential means of connectivity between the Raspberry Pi and the mobile application by operating at the physical and data link layers, providing the underlying infrastructure that allows data to be transmitted over a local network. Wi-Fi enables wireless communication between devices on the same network, creating a seamless, cable-free setup. When the Raspberry Pi and the mobile app are both connected to the same Wi-Fi network, they can interact as if they were directly linked, using Wi-Fi's capabilities to handle the physical transport of data packets between them. By leveraging Wi-Fi, the devices can communicate over more considerable distances within the network's reach, as long as both devices remain connected to the same network infrastructure. While WebSockets operates at the application layer, establishing the actual data-exchange protocol for real-time, bidirectional communication, it relies on Wi-Fi to provide the physical path for data transport. WebSockets alone cannot transmit data; it requires an underlying transport layer, such as Wi-Fi, to facilitate this. Wi-Fi establishes the initial network connection and transmits packets across it, supporting WebSocket connections that, in turn, create a persistent link for sending data from the Raspberry Pi to the mobile app and vice versa.

#### Wireless Fidelity (Wi-Fi)

Wi-Fi operates as a wireless local area network (WLAN) technology that enables devices to connect over a shared network, typically within a range of approximately 30 to 50 meters indoors. Wi-Fi's underlying structure involves a physical and data link layer within the OSI model, which manages wireless packet transmission across a robust infrastructure. A primary benefit of Wi-Fi is its ability to handle high data transfer rates, which, depending on the Wi-Fi standard (such as Wi-Fi 5 or Wi-Fi 6), can range from hundreds of megabits per second to over a gigabit per second. This high data throughput is particularly advantageous when transmitting large amounts of data, as is common in applications that require real-time data visualization, such as FFT signal analysis in the mobile app.

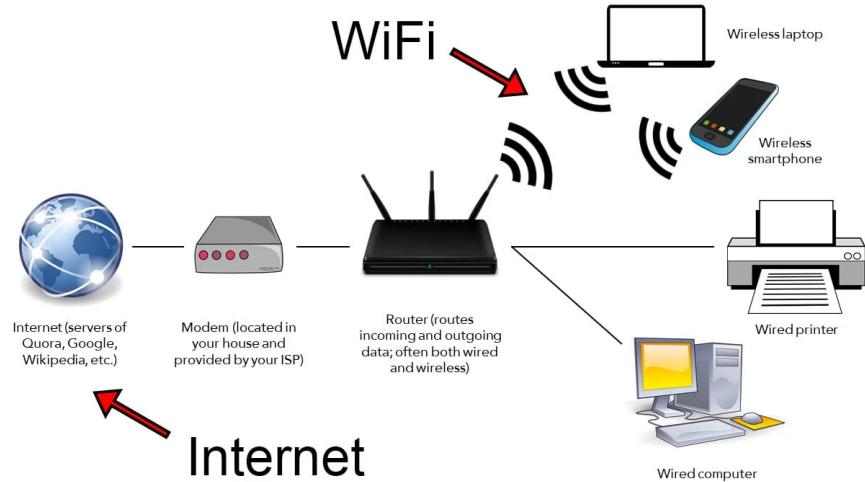


Figure 3.20: Wi-Fi Routing Explanation

Wi-Fi also benefits from its capability to support a longer transmission range compared to Bluetooth. Since Wi-Fi networks are designed to cover an entire building or large indoor space, it enables the Raspberry Pi and the mobile app to communicate across significant distances without connectivity issues, as long as both devices are within range of the same Wi-Fi network. Additionally, Wi-Fi supports network-based infrastructure, meaning multiple devices can connect to the same network simultaneously, allowing for a more scalable solution if additional devices need to be integrated. However, one notable drawback of Wi-Fi is its relatively high power consumption. The Raspberry Pi must maintain a continuous connection to the Wi-Fi network, which consumes more power compared to Bluetooth, especially in applications where long-term, battery-operated devices are used.

### Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) is a short-range wireless communication protocol that is optimized for low-power consumption. BLE is commonly used for data transfer between critical devices that are within close proximity and can only afford to use very little energy [36]. In this setup, the Raspberry Pi acts as a BLE peripheral, transmitting data to a mobile app (which acts as a BLE central device) over a Bluetooth connection.

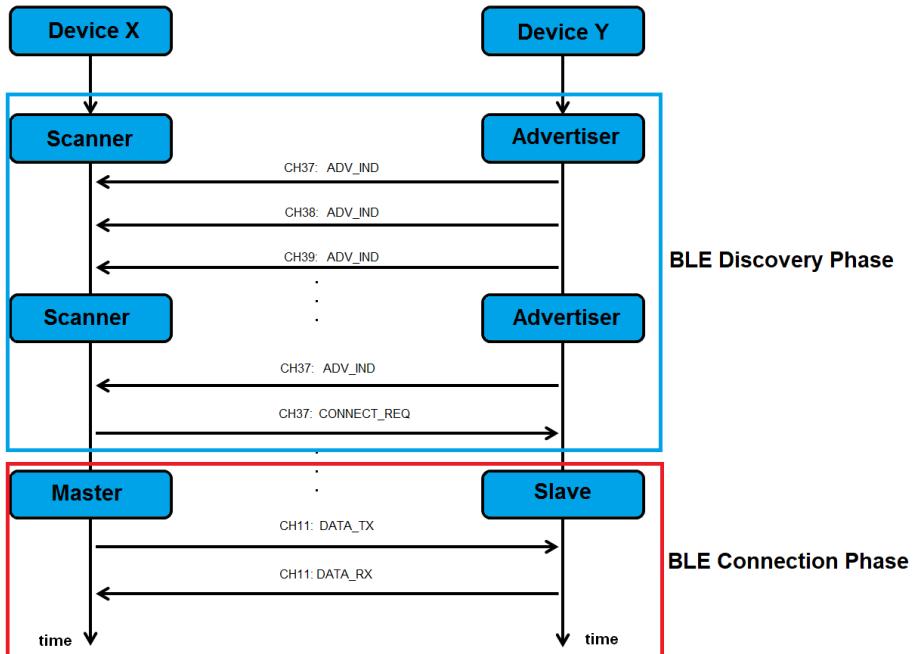


Figure 3.21: BLE Connection Steps

BLE is ideal for Internet of Things devices such as the Raspberry Pi, which has limited resources and battery life. Also, BLE excels at close-range communication, making it an efficient protocol for transmitting small amounts of data between nearby devices, such as the Raspberry Pi and mobile tablet. However, BLE is not designed for large, continuous data transfers, which would significantly hinder the ability to send FFT data from the Raspberry Pi to the mobile app.

### Justification for choosing Wi-Fi

In our project, Wi-Fi is ultimately the preferred communication protocol over Bluetooth, primarily due to its superior data transfer speed, scalability, and suitability for real-time requirements. Our project requires transmitting FFT data continuously from the Raspberry Pi to a mobile application, where high-speed data exchange is crucial to ensure smooth, low-latency visualization. Wi-Fi excels here, offering significantly higher data transfer rates than Bluetooth. While Bluetooth Low Energy (BLE) generally supports speeds around 1 Mbps, and classic Bluetooth reaches approximately 3 Mbps, Wi-Fi can handle transfer rates of hundreds of megabits per second depending on the standard used, such as Wi-Fi 5 or Wi-Fi 6. This high bandwidth allows for efficient real-time streaming, reducing lag and ensuring that the FFT data displayed in the app is timely and accurate.

Additionally, Wi-Fi provides a network-based infrastructure that supports simultaneous communication across multiple devices, which is advantageous for scalability. This setup eliminates the need for direct pairing, as required by Bluetooth, making it easier to integrate additional devices into the system. Multiple devices can connect and communicate over the same Wi-Fi network, which allows for sim-

plicity, should multiple users want to connect to the mixer. Conversely, Bluetooth's direct device-to-device connection model is less suited for scenarios where multiple devices need to interact concurrently within a shared network environment.

Real-time data transfer is also crucial in our project, as we aim to stream FFT data continuously for immediate updates on the mobile app. Wi-Fi, with its persistent high-throughput connection, allows for efficient real-time data streaming, a task that would be challenging to accomplish with Bluetooth's limited bandwidth. Leveraging WebSockets over Wi-Fi offers a stable, low-latency connection ideal for applications where timely data updates are essential, ensuring that the mobile app's visualization of FFT data is in sync with the audio being processed by the Raspberry Pi.

While Bluetooth, especially Bluetooth Low Energy, has an advantage in terms of power efficiency, our project prioritizes data transmission speed and range over power consumption. The Raspberry Pi, generally connected to a power source, can accommodate the increased power demands associated with Wi-Fi, which is a reasonable trade-off given the benefits of high-speed and reliable data transmission. Thus, while Bluetooth could work well for simple, low-bandwidth applications, Wi-Fi more effectively meets our project's requirements for real-time, high-speed data transfer.

### 3.2.6 Real-Time Processing Considerations

#### Need For Low Latency

Our application is one that requires a real-time response time; latency must be minimized at all costs. Strictly defined, "real-time" refers to the mixer's ability to respond to input within a short, predictable time window. Latency is the time between a raw audio signal being inputted, performing the requisite processing on it, and outputting it.

The live performative nature of musical performances, where rhythm and time are an inherent part of the artform, makes it very sensitive to latency in the order of magnitude of milliseconds. The upper limit of acceptable latency for musical applications is approximately 10 milliseconds – latency that exceeds this are merely conspicuous at best, and at worst being highly disruptive to performers and causing the monitor signal to be unplayable against, rendering the mixer unusable.

#### Need for Stable Processing

The processing deadlines in real-time audio processing are rigid and cannot be missed under any circumstances. Unlike traditional software, where delays merely results degraded user experience or reduced performance, the failure to meet deadlines in real-time audio applications has a direct and immediate impact on the integrity of the audio output. These deadline overruns manifest as audible artifacts introduced into the outputted audio signal.

These artifacts range in severity depending on how many samples failed to be processed before the deadline. A few missed samples produces *popping* and *clicking*, onomatopoeic terms referring to artifacts in the output audio caused by sub-millisecond durations of erroneous silence. As more samples are missed, and longer portions of the output stream become silent, the artifacts become outright *dropouts*, causing sections of audio to be skipped or delayed. These are not minor glitches that can be overlooked; even a single errant sample produces a highly conspicuous pop, and the accumulation of enough of these artifacts can reduce the audio quality to the point that the mixer is rendered unusable.

### Trade-off Between Latency and Stability

A fundamental trade-off exists between **stability** and **latency**, hinging on the size of the audio buffer. Buffering is used to mitigate the effect of variability in processing time and data availability. The size of the buffer determines how many samples, and therefore what length of audio, is processed at once in chunks, and it is this fact that causes the mixer's "buffer size" to be a crucial value in honing in the balance between low latency and high stability.

A larger buffer increases the stability of the audio stream by providing more leeway for the system to handle delays or spikes in processor load. Even if the average processing time per given length of audio is far less than that length, a long worst-case runtime can lead to deadline overruns were it not for the time headroom provided by a large buffer to absorb the delay and complete the buffer's worth of processing before the next deadline. With a smaller buffer size, every intermittent processing delay could result in audible artifacts.

However, the trade-off for the increased stability of larger buffers is an increase in latency. When a larger buffer is used, the system must accumulate larger amounts of data before it can be processed. Any amount of buffering must necessarily introduce a delay between the time the audio signal is captured and when it is heard, equal to the length of audio that the buffer's size represents.

Therefore, for the purposes of real-time processing, the optimal value for buffer size is the smallest possible setting such so that audio artifacts are non-existent, or at least very infrequent, in the output stream. This value results in the minimum possible latency without hindering stability, and is dependent upon on the optimization of the software and the power of the hardware it runs on.

Given the importance and salience of both latency and stability to users of audio software, most vendors of professional audio applications provide a setting to allow the manual adjustment of the application's buffer size, letting the user decide their own optimal balance between the two factors, depending on their hardware. Since the hardware of our mixer is a known factor, it is not a high priority for us to implement user control over buffer size; rather, we will likely hardcode it manually after empirical testing to determine its optimal value.

## Principles for Designing for Real-Time

All audio processing applications intended for live use must handle the challenge of ensuring real-time response. We can draw from precedent established by the developers of existing audio processing software, and follow their prescriptions for designing our mixer's software architecture.

In a word, the key principle for ensuring real-time response guarantees is **determinism**. Every process, algorithm, and operation involved in the audio pipeline must be predictable. The time it takes for an audio signal to be processed must be predictable and bounded, regardless of system load or external factors. It must be known that every task is guaranteed complete within an allotted time frame.

It is not enough for a task's average runtime to be short. If any part of the system has an arbitrarily long worst-case runtime, then the triggering of that worst-case can cascade and cause delays that propagate throughout the system, eventually leading to missed deadlines and audio artifacts. As such, certain operations, such as locking mechanisms, dynamic memory allocations, and complex algorithmic processes, are strictly avoided within the real-time processing loop to minimize the risk of introducing variability.

### 3.2.7 User Interface Frameworks, Comparison and Selection

In order to develop both the mobile and onboard graphical user interfaces for our mixer, we must select a suitable framework. Each UI framework is designed to cater to specific use cases and offers strengths in different areas. To narrow down our search, our selection process was guided by the following priorities:

**Cross-compilation** The chosen framework must support compilation to both mobile (iOS and Android) and desktop (Linux distro on the Pi). This ensures that we can maintain a single, unified codebase for all UIs.

**High performance with low overhead** Given the resource-constrained nature of the Pi, which must concurrently run the backend manager process for our mixer, the chosen framework should consume minimal CPU and memory resources.

**Low learning curve** The framework should be easy to adopt and efficient for our team to work with, allowing more development time to go toward working on the backend and DSP software components.

**Support for audio software interface components** Ideally, the framework should provide prebuilt audio-specific components, such as level meters, equalizer controls, and frequency spectrums.

Given these requirements, we narrowed our search to a shortlist of three frameworks: **Flutter**, **Slim**, and **Qt**. As follows in table 3.14 is a brief description and comparison of each:

| Feature                 | Flutter                                              | Slint                                            | Qt                                                               |
|-------------------------|------------------------------------------------------|--------------------------------------------------|------------------------------------------------------------------|
| Language                | Dart                                                 | Rust, C++, JavaScript                            | C++, QML                                                         |
| Cross-Platform Support  | Mobile, Web, Desktop (Linux, macOS, Windows)         | Embedded, Desktop (Linux, macOS, Windows)        | Desktop, Mobile, Embedded                                        |
| Target Platform         | Mobile, Web, Growing Desktop Support                 | Embedded Systems                                 | Desktop, Mobile, Embedded Systems                                |
| Performance             | Good (especially on mobile, not as great on desktop) | Excellent for embedded systems, very lightweight | Excellent, high performance, optimized for embedded              |
| Ecosystem and Libraries | Large, many third-party libraries                    | Small, evolving                                  | Very large and mature, rich set of modules                       |
| Learning Curve          | Moderate (Dart)                                      | Easy to moderate (depending on language)         | Steep (especially with C++ and QML)                              |
| Use Case Fit            | Best for mobile and web apps, some desktop apps      | Best for resource-constrained embedded projects  | Excellent for complex, cross-platform projects                   |
| License                 | BSD 3-Clause License                                 | GPL-3.0 or Commercial                            | GPL/LGPL for open source, Commercial license for proprietary use |

Table 3.14: Comparison of Flutter, Slint, and Qt

### Justification for Selecting Flutter

After careful consideration, we chose Flutter as the framework for developing our mobile and onboard user interfaces.

The reasons for this decision are as follows:

**Mobile-oriented** Our mobile interface is intended to be the mixer's primary UI, with the onboard interface serving in an auxiliary capacity. In addition, our primary design objective for our UI is to provide a fluid, user-friendly interface. Flutter, as a mobile-first, modern, and responsive framework, is the best framework to meet these objectives.

**Developer productivity** Out of the shortlisted frameworks, Flutter has the lowest learning curve, as well as quality-of-life features such as hot reload and

modular widget architecture. These factors will serve to significantly reduce development time, allowing for quicker iteration and refinement of our UI.

**Expansive ecosystem** Flutter has a large and active community, which offers a wealth of open-source third-party libraries and packages, making it easier to bootstrap the audio-specific UI components we require without the need to build entirely from scratch.

**Performance** Flutter's suboptimal performance is its largest pitfall; due to its bulky rendering engine, it is an order of magnitude less performant than Slint and Qt, which interface directly with the native APIs of platforms. Nonetheless, its performance is sufficient for our purposes, and the advantages Flutter offers outweighs

Ultimately, on a project with such a high degree of complexity, developer time is a highly scarce resource. Flutter strikes the best balance between ease of development, expansive built-in, and (sufficient) performance, out of the remaining frameworks, making it the most suitable choice for our project.

### **3.2.8 Automixing Methods, Comparison, and Selection**

Automatic mixing has become an essential feature in modern studio systems, especially in live performances, broadcasts and general recording environments. It falls under this broader realm of intelligent music production (IMP), a field focused on applying automation, incorporating a level of artificial intelligence (AI) and machine learning (ML) into the field of music production to streamline the music creation process. Automixing involves dynamically balancing audio channels to achieve a cohesive and professional mix without the constant need for manual oversight. This makes it particularly advantageous for speeding up workflows and offering ease of access to amateurs or those without advanced audio engineering skills.

There have been a litany of approaches to automixing, offering distinct advantages depending on the context and requirements of the application. These methods can range from traditional, rule-based or knowledge engineered DSP methods to more recent advancements in machine and now deep learning. Knowledge-based systems are generally rules and guidelines created by audio engineering professionals or specifically selected by the designer and the respective algorithm to apply the domain knowledge for the task. Whereas, machine learning methods use statistical models and algorithms in a data-driven manner by applying a sort of optimization to accomplish the general function. [37]

#### **A Historical Overview of Automatic Mixing**

It is well documented in academia and industry practice of the history and direction of music production. Here we present a more historical overview exploring the progression and highlight the major developments in the field that show approaches changing from rule-based systems to the integration of machine learning due to the improving efficiency, creativity, and flexibility aided by general commercial excitement.

Efforts in automixing began with deterministic, rule-based systems like Dan Dugan's automatic microphone mixing in 1975 [38] performing basic tasks such as gain adjustment during live events by equalizing input channel levels. This design was simple yet effective and became the foundation for subsequent designs for automixing.

Fast forward to the early 2000s, where Pachet and Delerue's constraint optimization approach for multitrack audio mixing introduced the idea that there could be an optimal solution to balancing audio, changing the landscape of IMP. Here they proposed a system for real time mixing of audio sources with spatialization capabilities with a GUI to represent sound sources and their properties. When a user moves a source, the constraint system ensures the specified properties are the same. It allows users to choose different listening viewpoints while preserving the audio engineers intentions [39, 40].

The years between 2007 and 2010 saw a new wave of innovative research through the contribution of Enrique Perez Gonzalez's work, which broadened the

definition of automatic mixing by incorporating stereo panning, equalization, gain, etc into automated systems [38, 40]. Gonzalez's work on optimizing gain levels included calculating the ratio between each channel's loudness and the overall loudness of the stereo mix. He uses a cross-adaptive algorithm to map the ratios to channel gain values with the goal of optimizing gain levels of individual input channels in a live audio mixture [41]. To clarify, as shown in Figure 3.16, an adaptive audio effect is a sound transformation that adjusts its settings based on the characteristics of the audio signal it's processing through time-varying control. A cross-adaptive audio effects are adaptive audio effects where the parameters of one effect are modulated by features of another audio signal [42]. Another example was his work on equalization through the use of accumulative spectral decomposition techniques to analyze the frequency content of each channel. This approach applies a cross-adaptive audio effects to equalize the levels [43]. This work had strong contributions to the applications of live audio mixing and improving setup time. The work also marked the true start of automixing as its known today, with systems automating a more wider range of effects and parameters.

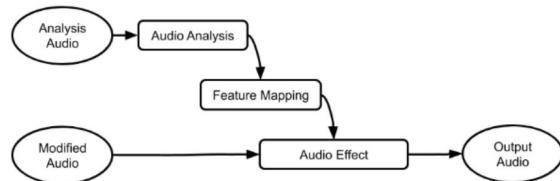


Figure 3.22: Flow diagram of an adaptive audio effect

By 2013, Brecht De Man and Joshua Reiss' introduction of a knowledge-engineered system highlighted the growing shift from simpler level and panning tasks as mentioned earlier, to more complex processing like dynamic range compression, equalization and high pass filtering, fading etc. Also, incorporating semantics like instrument and genre specific processing instead of just the low-level features of the raw audio signals [38]. By implementing these semantic mixing rules derived from audio engineering practices and textbooks, they ensured that the system made decisions that mirrored the expertise of professional audio engineers. This approach allowed automixing systems to handle more sophisticated tasks mentioned previously , which were previously difficult to automate without user input. By focusing on the semantics of the mixing process, De Man's system went beyond basic signal features and simple volume adjustments to deliver mixes that were more aligned with professional standards as well as leaving the door open to future research to include viable autonomous reverb/delay processors and reverberation rules [44].

Machine learning approaches rose to prominence between 2016 and 2017, bringing more advanced tools to the realm of automixing. The field transitioned from rule-based systems and knowledge-engineering techniques that dominated early automixing, towards more data-driven approaches. This shift was driven by the increasing complexity of tasks required in music production, such as dynamic range compression, reverb control, and audio source separation, which could no

longer be handled efficiently through deterministic, rules-based methods alone. Machine learning enabled more flexible and adaptive solutions. It also spurred the development of hybrid models and approaches that combined the strengths of both paradigms. For example, a system might use knowledge-based rules for initial tasks like instrument identification, but leverage machine learning for tasks like dynamic range compression and reverb which require finer tuning. This fusion allowed for more context-aware automation to handle technical mixing tasks but more subjective decisions based on training data [45]. The models are trained on large datasets of audio mixes to recognize the patterns and fine-grained nuances in the data that cannot be determined by predefined rules with music operating in such a complex multi-dimensional and rich parameter space. These predefined rules are now thought to be a departure from contemporary practice as its simply too restrictive. However, a common limitation of classical machine learning methods(methods as frequent late 2000s/early 2010s) that is proving to prevent further development is the lack of availability in data, with these models requiring large amounts of it to perform well. In recent years, deep learning has become widely accepted because of its ability to still generate meaningful results from smaller or out-of-domain datasets. These methods and architectures will be discussed in greater detail in the section below.

With mixing being a highly multidimensional in nature, the process requires audio engineers to make several simultaneous decisions. Is it the volume too quiet? The frequency range well-balanced? Is the panning fit withing the stereo field? Is the reverb set to appropriately to blend the instrument to the mix? Each of these elements can effect the overall mix and cannot be solved in isolation, attempting to solve one issue without considering how it interacts with other factors, can lead to new unresolved problems. Which is why mixing needs to be an all-in-one approach that is fully comprehensive and integrated [38, 46]. The transition to deep learning is what is needed to handle this interconnected and intricate task.

## **Classical Machine/Deep learning Methods and Models**

Researchers in the 2010's and later wanted to explore machine learning by leveraging the large amounts of parametric data collected from professionals to train systems to do such tasks. Below is the in depth overview of some of these prominent classical machine learning approaches for general automixing and specific complex processing units. The drawbacks for each are also mentioned to provide the necessary evidence to move towards deep learning focused models.

**Genetic Algorithms** Here an automatic mixing was based on a genetic algorithm that extracted audio features for leveling tracks. It calculate the Euclidean distance between modified spectral histograms of a mix and a target sound. This defense serves as a measure of timbral similarity, which the system uses to adjust parameters during the mixing process. Timbre is the quality of a sound that allows a listener to differentiate between two sounds that have the same pitch, duration, and intensity. The genetic algorithm optimizes

the coefficients required for best possible recreation of the mix. While the system can recreate mixes, its limitation is in dealing with complex audio signals where timbral features may not capture nuances and often impossible in providing "objective validation". The genetic algorithm also struggled due to the lack of data available [47]

**Linear Dynamical Systems** The proposed system automates the mix-down process for multitrack audio files using parameters learned through supervised machine learning within a structured audio framework. The primary model employed here is based on linear dynamical systems, which provide a way to model the relationships between different audio tracks over time. The reliance on linear models poses limitations, particularly when handling non-linear dynamics of music production, such as compression and EQ adjustments. This simplification may prevent the model from fully capturing the intricate interplay between different musical elements that are often key to achieving a professional sound. It also was only proven to work on estimated parameters, mentioning the need for a larger collection of multitrack session files and only being limited to one particular genre, likely implying its inability to properly generalize mixes for other genres as well [48].

**Hinge-Loss Markov Random Fields (HL-MRFs)** The method described here was used for multitrack reverberation via hinge-loss Markov random fields to model the spatial and temporal dependencies between audio tracks. The model enables the intelligent application of reverberation by balancing the relationships between tracks and the desired spatial characteristics. HL-MRFs provide a probabilistic framework that allows for flexible reverberation effects. However, the challenge with this approach lies in the computational complexity of the model, which can limit its scalability to larger, real-time mixing environments. Additionally, the probabilistic nature of the model may sometimes lead to reverberation artifacts that detract from the overall mix quality, especially when handling highly dynamic sound sources. It was also shown not to be comparable to professional mixes in subjective evaluations with excessive reverberation leading to poor performance. It had a lack of parametric data here and also relied on estimations. [49]

**General Statistical Feature-based Optimization Approaches** Such systems have tried to capture the automation of a DRC, compression, and reverb. The dynamic range compression system employs a statistical approach by extracting real-time features from the audio signal—such as amplitude envelope and frequency content—to optimize the attack, release, and ratio parameters automatically [50]. Feature-based optimization is used for personalized compression, where the model statistically learns user preferences through feedback and adjusts future audio compression settings accordingly, tailoring the output to individual listeners [51]. Another system, similar to HL-MRFs, attempts to improve automation in reverberation with a feature-based approach. Its done by training a statistical model to learn mappings from

datasets of reverberant and non-reverberant signals, and analyzing the signal features(frequency spectrum and temporal dynamics) to apply appropriate reverberation effects [1]. The common drawbacks across these statistical feature-based optimization approaches for audio processing include a consistent mismatch between mathematically derived or automated parameters and perceptually accurate outcomes, leading to results that may sound unnatural or suboptimal. Additionally, the automation of key parameters limits user flexibility, reducing the ability to fine-tune settings for specific, complex audio scenarios. These methods also rely on predefined or static mappings between features and parameters, which fail to adapt dynamically to unexpected or rapidly changing audio inputs, further restricting their effectiveness in diverse real-world applications.

Overall, the classical methods over rely on predefined relationships between limited sets of audio features and parameters. It also only had limited and often manually curated datasets that included only specific audio features without comprehensive coverage of all relevant parameters (e.g., detailed processor settings, dry and processed audio comparisons) that often had to be estimated or excluded, impacting qualitative and at times quantitative performance. Such scarcity of these datasets prevented further pursuit and development of this field.

### **Deep Learning vs. Classical ML**

Despite the availability of larger, more comprehensive datasets today, classical machine learning techniques for audio processing, and more specifically automixing aren't revisited or as accepted due to several reasons:

**Superior Performance** Deep learning architectures are made up of neural networks with many layers, optimized via loss functions enabled by gradient descent and backpropagation [52] to provide powerful generalization and representation learning capabilities. For example, models such as convolutional neural networks(CNNs), Long Short Term Memory (LSTMs), autoencoders, etc, have consistently outperformed classical machine learning techniques. These model architectures are capable of learning complex, non-linear relationships between high-dimensional audio features and output directly from the data, without requiring manually crafted features or predefined mappings.

**End-to-End Learning** Deep learning has the ability to perform what is known as end-to-end learning. Instead of separating feature extraction and model training as classical methods do (shown in Figure 3.17), deep learning models simultaneously learn both processes, optimizing them together. This integrated approach can capture richer, more nuanced relationships in the data and leads to better generalization, which classical ML models, even with new datasets, struggle to replicate effectively.

**Transfer Learning and Pretrained Models** In deep learning the concept of transfer learning is a very flexible and powerful tool. Transfer learning is when a

| Method                                           | Description                                                                                                                                                                                                                        |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Genetic Algorithms</b>                        | Uses genetic algorithms to optimize audio features, adjusting parameters to achieve timbral similarity between a mix and target sound, but struggles with complex signals and lacks objective validation due to limited data [47]. |
| <b>Linear Dynamical Systems</b>                  | Automates multitrack mixing using learned parameters, but linear models are limited in capturing non-linear music production elements, resulting in poor generalization across genres [48].                                        |
| <b>Hinge-Loss Markov Random Fields (HL-MRFs)</b> | Applies probabilistic modeling for reverberation, balancing spatial dependencies, but computationally intensive, and subjective evaluations show limitations in achieving professional mix quality [49].                           |
| <b>Statistical Feature-based Optimization</b>    | Optimizes dynamic range compression and reverb using statistical features, but often lacks perceptual accuracy and flexibility, limiting adaptability to dynamic audio environments [1, 50, 51].                                   |

Table 3.15: Summary of Classical Machine Learning Methods for Automixing

model is trained on one task and is used as the starting point for a more domain-specific task. These pretrained models are originally trained on large datasets that can then be fine-tuned to specific tasks. This gives more adaptability to deep learning models for various inter and intra domain specific tasks related to audio processing. For example, a large network could be pretrained on large dataset of commercial music tracks to automate mixing parameters for new audio input across different genres or instruments. Once this model is pretrained, the model can be fine-tuned to a smaller, more limited dataset specific to a new task like mixing live concert audio or podcasts. The pretrained model uses its learned knowledge to quickly adjust and automate the mixing parameters, leading to faster, potentially more accurate

decisions for the new tasks even with less data and manual intervention.

**Feature Engineering Limitations** Classical ML methods heavily depend on manual feature engineering, which is labor-intensive and often limited in scope. While more parametric data is available now, classical methods would still require considerable effort to determine the right features and how to use them for specific audio tasks. On the other hand, deep learning automatically learns relevant features from raw data, significantly reducing the need for manual feature selection.

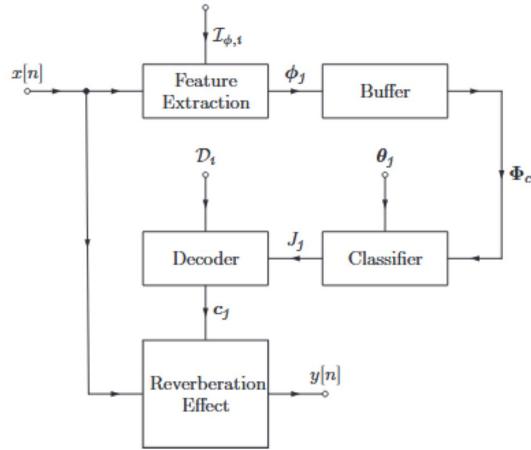


Figure 3.23: Architecture of [1] applying Reverb. Feature Extraction and Model training Training are Decoupled.

**Computation Power and Infrastructure** With modern computing infrastructure, can efficiently handle large datasets and complex models, whereas revisiting classical ML techniques might not leverage computational resources as effectively. Classical models generally don't scale as well with larger datasets or benefit as much from parallelization on modern hardware like GPUs.

**Research & Development Support/Focus** Many areas of machine learning, has shifted focus toward neural networks and deep learning. As a result, there is more funding, tools, and community support for advancing deep learning approaches. This focus means less incentive to revisit classical methods, even with the availability of better datasets.

### Challenges for Deep Learning Models in Automixing

The promise of deep learning is that it has the potential to learn mixes directly from tracks and mixes without any knowledge of the parameters used, alleviating a major constraint in data procurement and availability. However, in order to get there, it has to account for a number of challenges. 1) There is *still* limited training data,

| Characteristics                             | Deep Learning (DL)                                                                                                                                            | Classical ML                                                                                                        |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>Performance</b>                          | Utilizes deep neural networks with multiple layers to capture complex, non-linear relationships and achieves superior generalization and representation [52]. | Depends on handcrafted features and predefined mappings, which can limit model flexibility and performance.         |
| <b>End-to-End Learning</b>                  | Integrates feature extraction and model training, optimizing both simultaneously for improved generalization.                                                 | Requires separate feature extraction and model training, limiting capture of nuanced data relationships.            |
| <b>Transfer Learning</b>                    | Leverages pretrained models on large datasets, enabling quick adaptation and fine-tuning for domain-specific tasks with less data.                            | Lacks transfer learning capability, often requiring models to be built from scratch for new tasks.                  |
| <b>Feature Eng.</b>                         | Automatically learns features from raw data, reducing need for manual feature engineering.                                                                    | Relies heavily on manual feature selection, which is labor-intensive and often requires extensive domain knowledge. |
| <b>Computation Power and Infrastructure</b> | Efficiently scales with modern hardware (e.g., GPUs), making it feasible to handle large datasets and complex models.                                         | Less scalable on modern hardware, with limited benefits from parallel processing.                                   |
| <b>R&amp;D Support</b>                      | Receives significant funding, tools, and community support, with more resources focused on advancing DL.                                                      | Has limited support and fewer new developments, with less incentive for research focus.                             |

Table 3.16: Comparison of Deep Learning and Classical Machine Learning for Automixing

needing paired inputs of original tracks and the good mixes. The demands on generalization make this task extremely data hungry, even as more datasets become available. **2)** Evaluating a mix is not necessarily as straightforward as it seems due to the subjective nature of music. There are not universal rules or guidelines to what makes a good mix. **3)** Mixing often has highly variable inputs with no consistent size and structure to inputs. Audio recordings can vary significantly, with different mixes varying the number of input tracks (vocal, guitar, bass, drums, etc). This could prove to be an obstacle when it comes to generalizing across different mixes. The structure of the input could have different spectral, temporal and dynamic properties. Some tracks may be more high-frequency and some may be dominated by low frequencies. Mixes come in different genre's and styles with

some being more vocal and others more instrumental. These subjective, artistic decisions can prove to be difficult to model. **4)** Ensuring a high-fidelity output is not easy. Such audio requires high sampling rates, often at 44.1 kHz, 48 kHz, or in professional contexts 96 kHz. The high sampling rate ensures the full spectrum of audio is captured in order to preserve all audio information. But doing this results in large volumes of data which can be very computationally demanding on both training and inferencing times. In real time automixing environments, the models need to process audio with minimal latency and the large inputs can generate artifacts when the system cannot keep up with the audio stream. Artifacts are unwanted alterations that distort the sound, like clicks, pops or unnatural frequency modulations. When deep learning applies various audio effects, it must account for this and reduces the margin of error in parameter estimation or direct transformations to prevent such distortions. The model has to have temporal and spectral consistency. There cannot be abrupt changes in amplitude over short time windows, this can create noticeable clicks. Thus, smooth transitions are necessary, which puts more stress on loss functions to minimize this. **5)** User interaction is something that should be incorporated into the deep learning process. For audio engineers, they often may need to tweak and adjust the mix output. While a model can fully automate the process, the creative and subjective nature of audio mixing requires human intervention to fine-tune the results. [53, 54].

### Considerations for Automixing

The challenges outlined can then be distilled into several important considerations. The first being **interpretability**. It is important that the model can show what kind of manipulation is being applied to the input tracks in mix production as deep learning models tend to be *black-box* in nature. In the same vein, the **controllability** of the system can help understand not just what types of signal effects are being applied but allow user interaction so that adjustments to these transformations can be made. Extending from this is the idea of **context-aware systems**. Taking into the context of the genre, instrument, so that the final mix is more tailored and musically relevant.

The system's input recordings are unordered. Therefore, the order in which the recordings are given to the system should not change the results. There are two main ways to go about doing this. One being a **fixed input taxonomy**, which trains the model using a dataset where the kind of and number of input sources are fixed in both training and inferences. E.g - In a vocal recording, the input could follow a set structure of lead vocal, backing vocal 1, backing vocal 2, and ambient mic. This allows for simplification in processing but can restrict flexibility and handicapping generalization capabilities where input tracks can vary unpredictably. The other more permutation invariant approach is to share weights. The same set of weights are applied to all input tracks, independent of its order, thus focusing on the track's intrinsic features rather than the position of the track. Similarly is how the model handles the **number of inputs** given to it. A max number could be set, with silence being added when there is fewer than the maximum. If its greater

than the maximum number of recordings, the shared weight implementation can handle this since it treats the tracks as set instead of a fixed vector. Another way is through possible aggregation or concatenation techniques and context sharing of the inputs [55]. Finally, the **expressivity** of the model should not be looked over. It defines how well the model can replicate to a wide range of mixing scenarios present in the dataset [53].

## Deep Learning Model Applications and Overview

Recently, deep learning models have served a variety of nuanced techniques: data generation via source separation, creating differentiable audio effects, automatic upmixing, audio source manipulation and style transfer. But in general the goal is to reverse engineering an existing mix or create a new mix given a set of stems.

Early iterations applied deep neural and feed forward networks for specific processing of audio effects such as with dynamic range compression and for parameter prediction [56–58]. Implementations for specific effects also saw the use of convolutional neural networks, such as here with end-to-end equalization [59]. Models have shown how to deconstruct a multitrack mix and obtain the specific parameters from it given the original stems and the final mix [60]. A popular model known as Wave-U-Net was proposed for source separation directly in the time domain, eliminating the need for spectral transformations [61]. This model was then repurposed for specifically producing mixes for drum sets, with this variant being more data driven, swapping the input and output [55].

Furthering the development of automixing was the **Differentiable Mixing Console** developed by Steinmetz et al. They pushed boundaries by creating a differentiable pipeline for emulating the series connection of audio effects of EQ, compression, and reverb. The model enabled the full automation of multitrack mixing by predicting parameters directly from raw audio waveforms, introducing neural proxies for traditional effects processors, and incorporating a stereo-invariant loss function [62]. Despite these advances, the challenge of generalization remained but was significant step toward learning mixing conventions and paving the way for more sophisticated neural audio effects systems.

To address the limitations of datasets and generalization capabilities in models, Martínez-Ramírez et al. proposed an approach to reuse out-of-domain datasets like wet stems for training deep learning models to mix audio. The method involved normalizing wet stems based on audio features related to common effects, allowing the model to approximate professional-level mixes. They also designed a novel stereo-invariant, perceptually motivated loss functions to improve mix quality. The use of large datasets in conjunction with the new loss function showed promising results and reduced the gap between auto and human-made mixes [63].

Working around genre limitations brings about the idea of style transfer. Style transfer involves applying the characteristics of one mix or genre to another. In the context of music production, this could mean adapting the mixing decisions or audio effects typical of one genre to another. By leveraging deep learning models trained on large datasets, a system could automatically identify and adapt the ap-

propriate mix based on the target style, making it more intention-driven to what a potential client wants [53].

Addressing the need for style transfer, the latest work of Vanka et al. (2024) introduced **Diff-MST**, a framework that automates the generation of a multitrack mix by inferring production attributes from a reference song. Their system builds of the differentiable mixing console with a transformer-based controller to predict control parameters for audio effects to generate mixes with high fidelity and can accommodate any number of input tracks. Diff-MST offers user control and allowing post-hoc adjustments, offering the practicality that previous models may not have had for real-world applications and tasks. Additionally, the audio production style loss function captures the global sound, dynamics and spatial attributes of the reference song, improve the transfers accuracy [64].

## Model Architecture Comparison and Selection

Before delving into the specific architectures under consideration, its important to look at the frameworks or paradigm approach these models will fall under, why its relevant to our task, and a justification of one over the other. The two dominant schema for automixing model architecture is either direct transformation or parameter estimation.

### Direct Transformation

Direct Transformation approaches aim to simplify the process by directly mapping input recordings to a final audio mix, bypassing the need for manual parameter adjustments required by traditional audio engineering methods. The model tries to generate a mix that is as close as possible to the reference mix, without requiring information about specific audio effects or the internal workings of digital audio workstations (DAWs). The training is guided by a loss function that minimizes the difference between the predicted and ground truth mixes in either the time, frequency, or combination of both.

Steinmetz et al. characterizes Direct Transformations as follows: A dataset consists of multiple examples, with each example containing a set of input recordings  $\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \dots, \mathbf{x}_N^{(i)}$ , along with a corresponding mixed output  $\mathbf{Y}^{(i)}$  produced by an audio engineer. Formally, the dataset is represented  $\mathcal{D}$  as:

$$\mathcal{D} = \left\{ \left( \mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \dots, \mathbf{x}_N^{(i)}, \mathbf{Y}^{(i)} \right) \right\}_{i=1}^E$$

where  $E$  is the number of examples. The model  $f_\theta$  is trained to take the recordings as input and produce an estimate  $\hat{\mathbf{Y}}^{(i)}$ , aiming to minimize the difference between  $\hat{\mathbf{Y}}^{(i)}$  and the ground truth mix  $\mathbf{Y}^{(i)}$ . This is done by optimizing a loss function  $\mathcal{L}_a(\hat{\mathbf{Y}}^{(i)}, \mathbf{Y}^{(i)})$  [53].

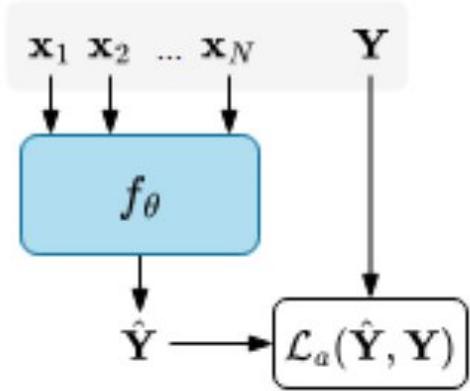


Figure 3.24: Visualization of Direct Transformation courtesy of Steinmetz et al.

**Advantage** Due to its minimal assumptions. It allows the model to implicitly learn and apply audio effects without needing explicit knowledge of the signal processing chain.

**Disadvantage(s)** A large dataset is required for a reasonable training performance and generalization since the model makes few assumptions. It tends to produce degrading signal transformations, aka artifacts. **Interpretability and controllability** is limited since the user cannot see the mixing parameters but only the final audio transformation.

### Parameter Estimation and Justification

The limitations of direct transformations motivate this schema to focus on automatically adjusting audio processing parameters (gain, pan, EQ, etc) to optimize the overall mix. In deep learning, the process begins with the network observing the multi-track input audio, and learning how to predict the best set of parameters for different effects. The predictions are most often framed as regression problems where the network is trained to put specific values that would result in the desired sound [65].

Steinmetz. et al models it as follows: A mixing console modeled by a function  $h$ , which processes a set of  $N$  input recordings along with parameters for each recording, resulting in a final stereo mix [53]:

$$\mathbf{Y} = h(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N, p_1, p_2, \dots, p_N)$$

**Advantages** Modularity because the signal processing change assumes each input is processed through the same structures - a series connection of audio effects. **More Interpretability/Control** as this manipulation is handled by predefined signal processing devices outlined in  $h$ . This in turn reduces the work that has to be done by the model.

**Disadvantages** Reduced expressivity limited by  $h$ . A loss computed in the parameter space measuring the distance between estimated parameters and ground truth parameters does not have many datasets that support this. It would require audio engineers to follow a fixed mixing chain to generate training data which would hurt practical use cases and data representativeness. Additionally, there is the idea of overparameterization, where different combinations of parameters could result in very stark differences in the audio domain. Sets of parameters could generate similar mixes but be far apart in parameter loss and be penalized [53].

However, there is a work-around to the computing parameter loss where the audio loss is simply used. This would require a differentiable signal processing chain of audio effects so that gradients could be automatically calculated and so the parameters could be optimized in the audio domain, thus circumventing the limitations of the parameter space, avoiding challenges of overparameterization and data scarcity. With preexisting methods for automatic differentiation in PyTorch, TensorFlow available, a parameter estimation approach can be leveraged within the deep learning framework. And given the nature of our project as a whole, where real-time performance and user control are cornerstones, it is the best approach. The only concern with automatic differentiation being computation as signal processing has recursive operations that, in the time domain, can be problematic for gradient computation because the gradient will need to be calculated sequentially at each time step during back-propagation [53]. This will be taken into account in the design phase.

With the problem formulated, the architectures can now be outlined, compared and one method ultimately selected.

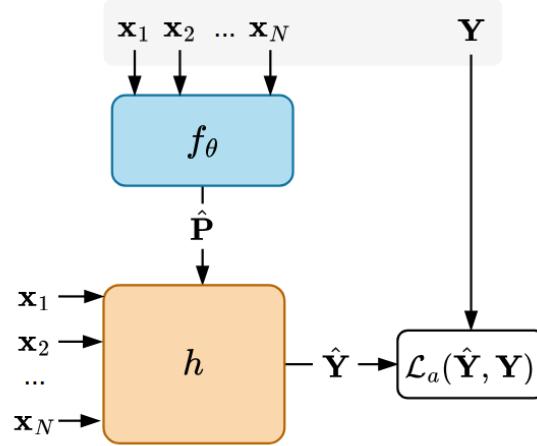


Figure 3.25: Visualization of Parameter Estimation with Audio Loss courtesy of Steinmetz et al.

**Architecture Breakdown and Justification** By breaking down the various models in Table 3.17, we can see that while Mix-Wave-U-Net offer promising capabil-

ties due to its simplicity and generalization in the context of wet stems, it leaves much to be desired due to the output being a final stereo mix as a result of direct transformations. The Diff-MST overall structure and need for a reference track does not align with our necessary application where in our setting we would only be giving the raw audio supplied by the performers using the mixer. Although, there are aspects of its architecture that are desirable such as how it implemented the differentiable effects fairly straightforward without proxies via the **dasp-pytorch** library. This leaves the Differentiable Mixing Console as the ideal candidate for the automixing algorithm to be implemented because of its flexibility and control over parameters. Also the input being in mel-Spectograms allows for faster processing compared to raw audio recordings. The complex gradient computation will be something to be weary of as its put into design, noting potential drawbacks for real-time inferencing.

**Time aligned pairs:** Stems and the final mix should be time-aligned so that temporal relationships and transformations can be learned.

**Diverse Instruments and Genres:** Better generalizing capabilities during inferencing, exposed to more diverse data

**Dry multitrack stems:** Wet stems have effects already applied, limiting the model's ability to learn how to apply mixing techniques independently and its hard to remove these effects during training. Dry stems are unprocessed and are "ready to be learned".

It is clear from Table 3.18 that the best dataset to use for training is **MedleyDB** given the sheer size, and variety compared to all other datasets, and what seems to be no drawbacks at face value. It is also the dataset that the originators of the DMC recommend and use. Cambridge Multitrack is second choice and perhaps could be trained alongside MedleyDB if preprocessing is implemented to account for the recordings not being time-aligned or as a comparative dataset to evaluate performance.

| <b>Method</b>                    | <b>Input</b>                                                                                         | <b>Output</b>                                                     | <b>Architecture</b>                                                                                                                                                               | <b>Adv.</b>                                                                                                           | <b>Disadv.</b>                                                                                                                         |
|----------------------------------|------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Mix-Wave-U-Net</b>            | Multitrack audio recordings (typically 8 channels) in the time domain                                | Final stereo mix (direct transformation)                          | Encoder-decoder with 1-D convolutions, skip connections inspired by U-Net architecture                                                                                            | Efficient at mimicking professional mixes (EQ, compression, reverb) without requiring knowledge of individual effects | No user control, artifacts with out-of-distribution inputs, fixed number of input channels                                             |
| <b>DMC</b>                       | Mel-Spectrograms of multitrack audio recordings with pre-processed context embeddings for each track | Predicted mixing parameters (EQ, compression, gain, panning, etc) | CNN-based VGGish encoder for feature extraction, shared-weight networks, context embedding, MLP for parameter prediction via Post-processor, Transformation Netork for DSP Chain, | Flexibility in input track numbers, user control over mixing parameters, adapts to different scenarios                | Complex gradient computation, requires differentiable effects, computationally expensive without proxie networks pretrained beforehand |
| <b>Fx-Normalization (CRAFx2)</b> | Wet multitrack recordings (processed tracks with applied effects)                                    | Predicted mix with adjusted loudness, panning, EQ, compression    | Convolutional and recurrent neural networks (CRAFx) with stacked TCNs and BLSTMs for normalization and mix prediction                                                             | Utilizes real-world wet data, adaptable to real scenarios, produces mixes comparable to professional mixes            | Requires extensive preprocessing (Fx-normalization), complex architecture and large architecture, requires large datasets for training |
| <b>Diff-MST</b>                  | Multitrack audio recordings and a reference track for style transfer                                 | Predicted mixing parameters (EQ, compression, gain, panning)      | Transformer-based controller, two encoders (one for input tracks, one for reference), outputs parameters to DMC for final mix                                                     | Effective style transfer, adaptable to variable input tracks, allows further control and fine-tuning by the user      | Relies on reference mixes, potential for artifacts with mismatched inputs, requires large and high-quality reference datasets          |

Table 3.17: Comparison of Automatic Mixing Methods

| Dataset                     | Size (Hrs) | no. of Songs | no. of Instrument Category | no. of Tracks | Type             | Usage Per-mis-sions | Other info                      | Remarks                                                    |
|-----------------------------|------------|--------------|----------------------------|---------------|------------------|---------------------|---------------------------------|------------------------------------------------------------|
| <b>MedleyDB</b>             | 7.2        | 122          | 82                         | 1-26          | Multitrack, Wav  | Open                | 44.1KHz, 16 bit, stereo         | -                                                          |
| <b>ENST Drums</b>           | 1.25       | -            | 1                          | 8             | Drums, Wav/AVI   | Limited             | 44.1KHz, 16 bit, stereo         | Drums only dataset                                         |
| <b>Cambridge Multitrack</b> | >3         | >50          | >5                         | 5-70          | Multitrack, Wav  | Open                | 44.1KHz, 16/24 bit, stereo      | Not time-aligned, recordings for all songs are not uniform |
| <b>MUSDB</b>                | 10         | 150          | 4                          | 4             | Multitrack, Wav  | Open                | 44.1KHz, Stereo                 | Used mainly for source separation, wet stems               |
| <b>Slakh</b>                | 145        | 2100         | 34                         | 4-48          | Synthesised Flac | Open                | 44.1KHz, 16 bit, stereo         | Used mainly for source separation; sometimes wet stems     |
| <b>Shaking Through</b>      | 4.5        | 68           | >30                        | >40           | Multitrack, Wav  | User only           | 44.1/88.2KHz, 16/24 bit, stereo | -                                                          |
| <b>BitMIDI</b>              | -          | >1M          | >5                         | >5            | Multitrack, MIDI | Open                | MIDI data                       | MIDI data submitted by users across world                  |

Table 3.18: Comparison of Various Multitrack Audio Datasets for Automixing Courtesy of Steinmetz et al.

## Chapter 4

# Standards and Design Constraints

This chapter is split into two overarching sections, standards and design constraints. It is imperative to have a grasp on both. Standards are critical to engineering projects in general; they provide consistency across the product and are particularly important in interfacing systems. They are particularly pertinent to the plethora of inter-device communications present in this project as well as the outward-facing connectors for audio input and output.

Design constraints, economic constraints, and other practical considerations will guide the project and limit its scope to that which is feasible within the time-frame of Senior Design 1 and Senior Design 2. Without detailing these constraints, certain aspects of the project may never reach completion.

The two sections are outlined below.

## 4.1 Standards

### Phantom Power

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Title</b>        | Standards for Phantom Power: IEC 61938:2018                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Authors</b>      | International Electrotechnical Committee                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Year Created</b> | 2018                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b>  | IEC 61938:2018 gives guidance on the best practices for multimedia analog interfaces to achieve interoperability. Among these is a specification for microphones and the use of phantom power. Three formal standards are specified, P48, P24, and P12. These specifications include the nominal voltages, operating currents, coupling resistor values, and more.                                                                            |
| <b>Application</b>  | For the mixer, phantom power must be available to power the use of certain microphones. The actual phantom power design will actually not adhere to the phantom power specifications, but it will extrapolate from the specifications to create a design that provides the same amount of power as the formal specifications. This is due to the difficulty of implementing a voltage regulator that matches the phantom power specification. |

### Universal Serial Bus (USB)

|                     |                                                                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Title</b>        | USB Implementers Forum                                                                                                                                                                                                   |
| <b>Authors</b>      | Apple - Dave Conroy, HP Inc. - Alan Berkema, Intel Corporation - Brad Saunders, Microsoft Corporation - Toby Nixon, Renesas Electronics - Philip Leung, STMicroelectronics - Gerard Mas, Texas Instruments - Anwar Sadat |
| <b>Year Created</b> | 1996                                                                                                                                                                                                                     |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | This standard outlines the guidelines and technical specifications for cables, connectors, and communication protocols used to enable seamless interaction between devices such as computers, peripherals, and other hardware. The USB standards, which are developed and overseen by the USB Implementers Forum (USB-IF), were established to create universal compatibility and simplify the connection process across devices. The USB-C connector, our chosen type for this application offers both speed and versatility. Its reversible oval design allows for insertion in either orientation. In addition to offering faster data transfer rates and enhanced power delivery, USB-C supports a broad range of functionalities, including audio streaming through USB audio interfaces. Furthermore, USB-C maintains backward compatibility with earlier USB versions, including USB 1.0 through USB 3.2. |
| <b>Application</b> | For the mixer, a USB interface will be implemented to facilitate the recording of audio into a host computer. The USB interface will conform to the physical standards of USB-C and the operability of USB 2.0 to ensure maximum compatibility. USB drivers and software are handled by first party libraries for the microcontroller platform, ensuring compliance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

### Serial Peripheral Interface (SPI)

|                     |                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Title</b>        | KeyStone Architecture Serial Peripheral Interface (SPI)                                                                                                                                                                                                                                                                                                 |
| <b>Authors</b>      | Texas Instruments                                                                                                                                                                                                                                                                                                                                       |
| <b>Year Created</b> | March 2012                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b>  | The Serial Peripheral Interface (SPI) is synchronous, meaning it uses a clock signal shared between the master and slave devices to ensure data integrity during transmission. SPI uses separate lines for sending and receiving data (MOSI and MISO), allowing full-duplex communication, which enables simultaneous data transfer in both directions. |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Application</b> | The mixer will utilize the Serial Peripheral Interface protocol to communicate in between devices on the PCB as well as between the xcore microcontroller and Raspberry Pi 5. Care will be taken to implement all the data lines necessary to utilize SPI for full-duplex communication. In terms of the software implementation, both the xcore and Raspberry Pi 5 support SPI with robust, well-documented libraries. This ensures compliance with the standard and should allow for smooth communications between the connected systems. |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Rust (programming language)

|                     |                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Title</b>        | Rust Programming Language                                                                                                                                                                                                                                                                                                                                                           |
| <b>Authors</b>      | Graydon Hoare, Mozilla, The Rust Foundation                                                                                                                                                                                                                                                                                                                                         |
| <b>Year Created</b> | May 2015                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>  | Rust is a multi-paradigm, systems-level language with a focus on safety, concurrency, and performance. It offers strong memory safety guarantees via its unique ownership model, which statically eliminates data races and memory leaks without requiring garbage collection. Its ecosystem includes extensive libraries for safe systems programming, audio processing, and more. |
| <b>Application</b>  | Rust will be used to implement the state manager module on the Raspberry Pi 5. It is the natural choice due to the need for low-level control over hardware interfaces between the Pi and other devices in conjunction with high-level constructs for abstracting our custom data types.                                                                                            |

## 4.2 Design Constraints

Below is a summary table of the main constraints upon the mixer project. Their details and impact on the project are detailed in the sections below.

| Constraint           | Description                                                                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cost                 | The ideal budget for this project should be under \$400, though \$500 may be required. There is no definitive maximum, but the intent is for it to be competitive with mixers on the market.                                                                                                                                                       |
| Time                 | The current semester of SD1 gives roughly 3 months (or 12 weeks) of time to properly plan and develop the software and hardware prototype for this device. Once in SD2, we will have another 3 months to finalize our project before the final deadline. Using our time wisely during SD1 is the most crucial part for the success of the project. |
| Hardware Constraints | Certain unavoidable hardware details place constraints on the design of the mixer. Part of the design process is finding suitable workarounds/solutions to such constraints.                                                                                                                                                                       |
| Social               | The mixer lies in the realm of music and the creative arts, and as such, the developers bear a level of responsibility to ensure the mixer is not used for offensive content or other negative purposes.                                                                                                                                           |

Table 4.5: Identified Constraints

#### 4.2.1 Economic & Time Constraints

In regards to economic constraints, the mixer project is self-funded, and as such, significant budgetary constraints are placed upon the project due to the limited resources of the developers. The most straightforward way in which the budget impacts the project is in the price of the selected components. Where possible, the cheaper option is used.

An example is in the operational amplifier selection. The NE5532 operational amplifier is quite old and does not perform like the best operational amplifiers on the market. However, audio op-amps are already quite good across the board. In order to reduce the cost of the hardware, the NE5532 was chosen due to its tremendous value.

Another example is in the selection of the single-board computer. In many ways, the Beaglebone AI-64 was the best option. It boasted vastly better audio IO compared to the Raspberry Pi 5 as well as a plethora of accelerator hardware components on the SOC which would have befitting our auto-mix algorithm. However, the cost was almost triple that of the Raspberry Pi 5. The extra features could not make up for the huge difference in cost.

There are several less obvious ways in which the economic constraints of the

project impact the design process. Among these is the way in which budget impacts the iterative nature of the prototyping process. While it may be ideal to test a variety of parts for each type of component, the budget constraints make it so that for most components, it is most viable to simply construct and test a single part for any given hardware design. Additionally, it is imperative that great care be taken care of to minimize the amount of hardware iterations necessary to create a functioning prototype. Beyond the repeated additions to the overall cost from the price of parts and board re-fabrication, another consideration is the shipping cost. Most electronic parts are unavailable from online shopping platforms that offer free shipping like Amazon. Instead, the predominant marketplaces like Digikey and Mouser charge money for shipping. As such, it is imperative to ensure the inclusion of all necessary parts in each order as well as the inclusion of enough spare parts to minimize the number of repeat orders.

The restriction on repeated parts reordering and board re-fabrication also factor into the constraints on time. For one, parts shipping is often a lengthy process. Any lack of parts will set back prototyping by several days. Additionally, board manufacturing and shipping is a very hefty time bottleneck. It can take upwards of several weeks to receive an order. Board orders must be thoroughly inspected and re-checked before they are sent in order to expedite the design process.

Time is also a particular constraint for the mixer project due to several hardware details. For one, the fine pitch of the XMOS xcore microcontroller pins means that board assembly cannot be done by hand. This introduces extra delays in the production of a board and also prevents quick rework. The necessity of a board that is easy to iterate on and rework has directed the PCB level design choices to emphasize modularity and reworkability through the use of expansion headers and breakout boards.

## 4.2.2 Hardware Constraints

As touched on above, the actual details of the hardware used for the mixer project impose certain constraints on the design process as well as the design itself. While it may be argued that the entire point of engineering is to work around and figure out the hardware that is available, there are some ways in which these hardware constraints are so unworkable such that they influence the part choice and other critical design decisions.

One example is the way in which the Raspberry Pi 5's IO situation affected the design of the mixer. While the Raspberry Pi 5 was indeed the best choice for the project due to its ability to cover the AI auto-mix functionality and wireless communications functionality, its limited audio IO complicated the design of the system. Because the Raspberry Pi 5 only has 2 channels of I2S, the full 8 channels of audio input cannot be streamed in real-time to the Pi. This restriction is particularly challenging to the design of the reactive audio GUI and the auto-mix feature, both of which require the input audio to operate. This hardware restriction necessitated some design compromises to be made, such as the onboard transformation of the audio on the microcontroller to be sent in small packets over SPI instead of the live

audio being transmitted.

### 4.2.3 Datasets

For production use, MedleyDB, MUSDB, Slakh, Cambridge Multitrack, and BitMIDI are generally suitable due to their open licenses, though some may require attribution or have minor restrictions on commercial use—reviewing specific license terms. MedleyDB, MUSDB, and Slakh are especially favorable, as they offer open access with minimal constraints. In contrast, ENST Drums and Shaking Through have restrictive or limited permissions, making them unsuitable for commercial production without additional licensing agreements or explicit permission but can still be used for educational purposes.

### 4.2.4 Social

The social aspect of the mixer is largely constrained and governed by its existence in the sphere of music and art. In as much as the mixer can be a helpful tool for artists who want to spread their legitimate artistic messages, bad actors can also use the mixer for purposes contrary to what is helpful or ethical. While the burden does not directly lie on the developers to mitigate such issues, it may be worthwhile to consider how the design of the mixer can be thought out so that it is unlikely to be used for purposes outside of what is intended.

## Chapter 5

# Comparison of ChatGPT with Other Similar Platforms

For better or for worse, ChatGPT and other LLMs have established their presence in our academic and professional lives. In this chapter, the ramifications of such a technology will be discussed. The topic is split into several overarching themes. One is a brief comparison between ChatGPT and other LLMs available to the students as well as a personal assessment of the usefulness and limitations of LLMs in application to our project. The next section will present examples of how ChatGPT has impacted the learning and research experience of the students.

## 5.1 Limitations, Pros, and Cons

A key advantage that ChatGPT has over its competing platforms is the immense financial endowment that it has compared to other LLMs. Research and development, training, and inferencing are all incredibly expensive activities, computationally and financially. The increased funding that OpenAI possesses translates to faster generation, more accurate and helpful inferences, more advanced models – all around, an enhanced user experience.

However, it is possible for LLMs to specialize for performing certain types of tasks. Microsoft's *Github Copilot* is a fine-tuned GPT-4o model, developed to assist with programming. It is unique in that it has been trained on snippets of code, can directly draw from public GitHub repositories, and operates within a text editor/IDE, offering real-time code suggestions to the programmer. Github Copilot is an excellent example of an alternative LLM implementation that offers increased utility to a developer compared to the chatbot interaction format of ChatGPT.

Another example of an LLM with specialized utility is Google's Gemini, specifically the Gemini instance embedded within Google search functionality. Gemini provides helpful summaries and explanations that directly answer the question you have typed into Google search. This is in contrast with the normal Google search functionality, which simply gives you hyperlinks to websites; you still have to look on the website yourself and infer the correct information. This gives Gemini a similar functionality to ChatGPT. However, it is superior in terms of answer reliability. Gemini, in addition to directly performing inference off of the search data, offers reference links showing where it got its answers from. That way, you can verify the legitimacy of its answers. This makes it particularly useful for research, where LLMs may tend to hallucinate and give you untrue or inaccurate answers to questions. Such is another example of a specialized LLM which extends the utility of LLMs beyond that which is offered by ChatGPT alone.

In terms of its overall utility, ChatGPT and other LLMs have proven quite useful to the students in their research for the mixer project as well as their studies in other fields. It functions as an enhanced web browser search which presents information in an organized and readable way. If accuracy is paramount, results can simply be verified through a browser search after a ChatGPT prompt.

However, there are certain limitations to ChatGPT and other LLM platforms. For one, LLMs offer drastically varying amounts of utility depending on the field of study. One may note that ChatGPT, in popular culture, has particularly affected the plight of computer science majors and software developers. LLMs are particularly useful in coding because of their text-based chat-bot format. One can simply ask some sort of prompt and they will receive a batch of ready-made code. Of course, the code may still need refactoring to work as intended. For one, LLMs are not omniscient in their coding abilities. Another factor is that LLMs typically cannot see the context of larger coding projects. This is even true of IDE-integrated LLMs like GitHub Copilot, although certain "software engineering" LLMs have emerged (e.g. Devin) with limited success. As such, LLMs have limited utility in large codebases.

Code snippets still necessitate the developer to understand how to integrate them into the code base.

However, with that said, LLMs offer drastically more utility to software engineers than they do to electrical engineers or other disciplines. In the case of electrical engineering, LLMs cannot produce an output such that the output can immediately be utilized or incorporated into a larger work. LLMs cannot output things useful to electrical engineers such as schematics, footprints, and PCB layouts. At best, they can describe what needs to be done in a verbal manner; however, such output still requires a knowledgeable engineer in order to incorporate such verbal suggestions into a real engineering design. One may suggest that there are generative image AI platforms incorporated into LLM platforms which can theoretically generate schematics and designs. However, such generative AI technologies are so laughably inaccurate such that they are entirely useless for engineering work. Below is an example of an attempt to generate a schematic. The prompt used was "Can you generate an image of a schematic of a preamp circuit with the SSM2019 IC?" The prompt was fed into ChatGPT, specifically the 4o model.

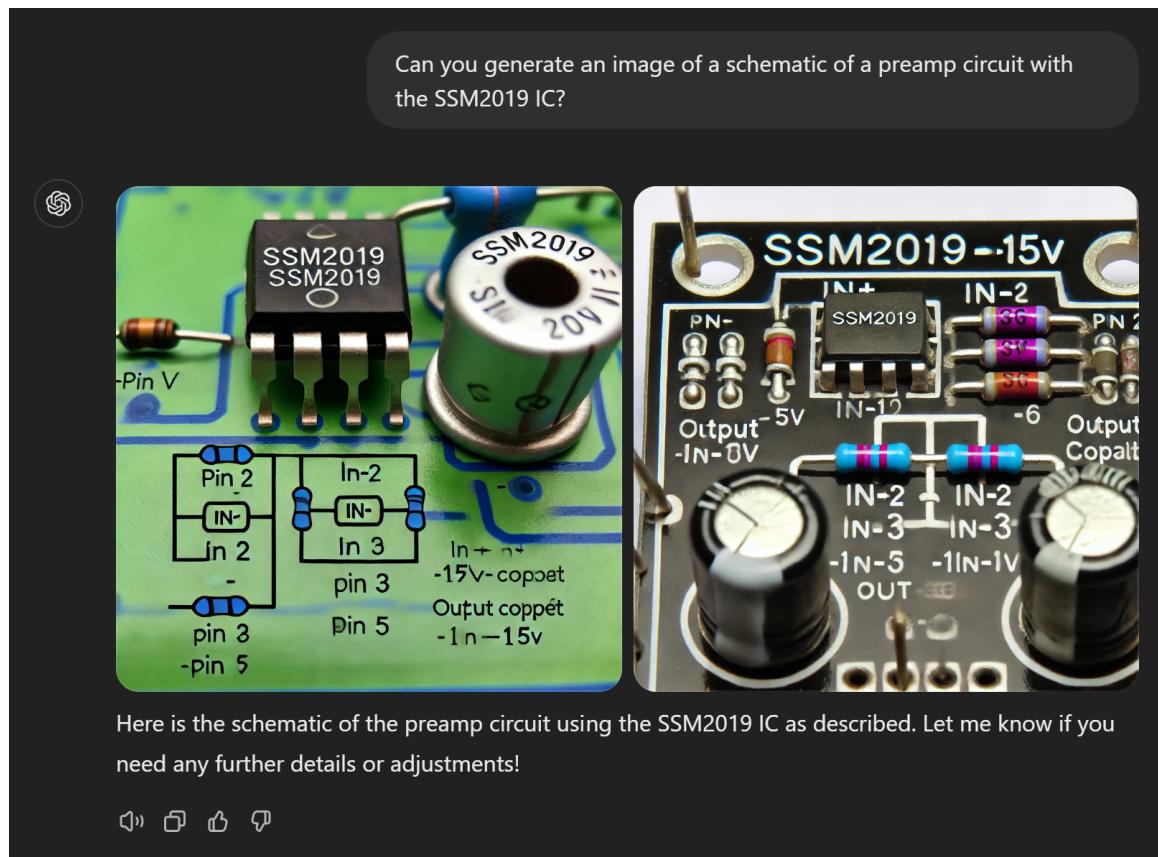


Figure 5.1: Prompt and Output from ChatGPT 4o

One may clearly see that the output is totally incoherent. Not only is the output itself not a real schematic, but the connections and text presented in the image output are complete gibberish. Generative image AI is particularly deficient when

it comes to text, and the results are clear.

| Model                 | Specialization                                 | Key Features and Limitations                                                                                                                                                                                                                      |
|-----------------------|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ChatGPT</b>        | General-purpose LLM                            | Provides natural language responses and assists with a wide range of topics. Useful in research and as an enhanced search tool, but prone to inaccuracies (hallucinations) and less specialized for technical fields like electrical engineering. |
| <b>GitHub Copilot</b> | Coding and IDE integration                     | Fine-tuned on code, operates within IDEs for real-time coding suggestions. Useful for software engineers, but limited in context-awareness for large projects and requires user knowledge for integration.                                        |
| <b>Google Gemini</b>  | Search integration and information reliability | Offers direct answers with reference links for verification, making it more reliable for factual queries and research tasks. Limited to search-related applications and less interactive than a chatbot.                                          |

Table 5.1: Comparison of ChatGPT, GitHub Copilot, and Google Gemini

## 5.2 Examples

There are many ways in which LLMs have affected the learning experience of the students. Below are several examples. Most of them are positive, albeit some are negative.

### 5.2.1 Automation of Boilerplate Tasks

One incredibly useful manner in which LLMs have been useful in the process of this project is the automatic generation of LaTeX tables. This document is typeset in LaTeX. For the uninitiated, LaTeX is a typesetting system for technical papers that offers incredible flexibility and aesthetically pleasing documents. However, there is some increased complexity when it comes to the creation and formatting of tables. Tables are not visual elements that can be modified with mouse clicks

or simple dragging actions. They are typeset with code, and as such, it can be cumbersome to create them. However, LLMs have created an incredibly simple workflow booster. With this new workflow, I can create a table in Microsoft Word or some other document editor, take a screenshot, and paste it into an LLM. Upon prompting, the LLM will immediately generate the necessary code to create the table in LaTeX, reducing overhead with LaTeX typesetting significantly.

### 5.2.2 Information Retrieval

LLMs can be seen in the context of the evolution of information discovery and retrieval mechanisms. Every advancement in this process involves the automation of tedious, repetitive tasks involved in procuring knowledge relevant to users. For instance, consider how search engines automate the process of finding relevant information given a query; the alternative would have been manual trawling of indexes and consciously identifying relevant entries, diverting time and attention away from the ultimate purpose of research – the synthesizing of information to solve a novel problem. In this light, LLMs can be viewed as a supercharged search engine, capable of taking complex, natural-language queries and producing an overview of relevant information for a highly specific problem, delivered in a narrative format.

In this fashion, LLMs have proved invaluable in their ability to illuminate "unknown unknowns" while researching complex, expansive topics. These are concepts and sources that one would not even know to solicit more information about, as opposed to "known unknowns". The uncovering of "unknown unknowns" is a task that search engines are particularly weak at – their strength of returning highly relevant information given a specific query is also a weakness, as search engines are of little help in navigating concept networks laterally, necessitating, again, the manual trawling of search results and conscious identification of gaps in one's existing knowledge.

## 5.3 Conclusion

For better or for worse, LLM-based AI is a technological innovation that is here to stay. The titanic investments being made by the US government [66] and private investors [67] into AI companies and research go to show the important role they will play in tomorrow's economy. As such, all professionals employed in knowledge work must learn to adapt to their presence, and understand the implications that it has for their work. For such professionals, the most realistic and beneficial view they can take regarding LLMs is to conceive of them as *tools*. Like any other tool, LLMs are best used as a means of performing work more efficiently, by making certain problems require less manual labor to overcome.

Inherent to the nature of tools is that a given tool will be helpful with performing some types of tasks, and useless with others. They are *not* a panacea; LLMs are *not* AGI, and the path to AGI will most likely *not* consist of an exponential runaway

of the intelligence of LLMs [68]. The overestimation of the abilities of LLMs are a mistake made by many, especially by the professional-managerial class, consisting almost entirely of knowledge workers who generally feel as if their livelihoods are existentially endangered by advances in AI [69]. The fear of AI, and automation in general, eliminating one's highly specialized position in society is a valid one to harbor. However, LLMs specifically are unlikely to be the harbinger of great upheaval. Like any other technological innovation, LLMs will in fact cause transformation in the economy, contributing to the loss, change, and addition of jobs. But ultimately, the maximum capabilities of LLMs are bottlenecked by their fundamental nature – they are simply memorization machines, no more, no less.

# Chapter 6

## Hardware Design

The following chapter details the hardware design of the digital mixer.

### 6.1 Subsystem Block Diagram

#### 6.1.1 Preamplification Circuitry

The input topology of the mixer includes 4 microphone/line combo inputs and 4 dedicated high-impedance (Hi-Z) instrument inputs. As such, this represents eight discrete preamplifier circuits split into two designs. The designs are shown below.

##### **Microphone and Line Preamp**

Below is the schematic for the microphone and line combo preamplifier circuit. As mentioned in the research section on input levels, the total input signal chain must offer at least 40 dB of gain for microphone inputs and a small gain ( $<10$  dB) for line inputs. Additionally, the preamplifier must be able to supply phantom power to the XLR input for microphones. The requirements are satisfied through the design presented.

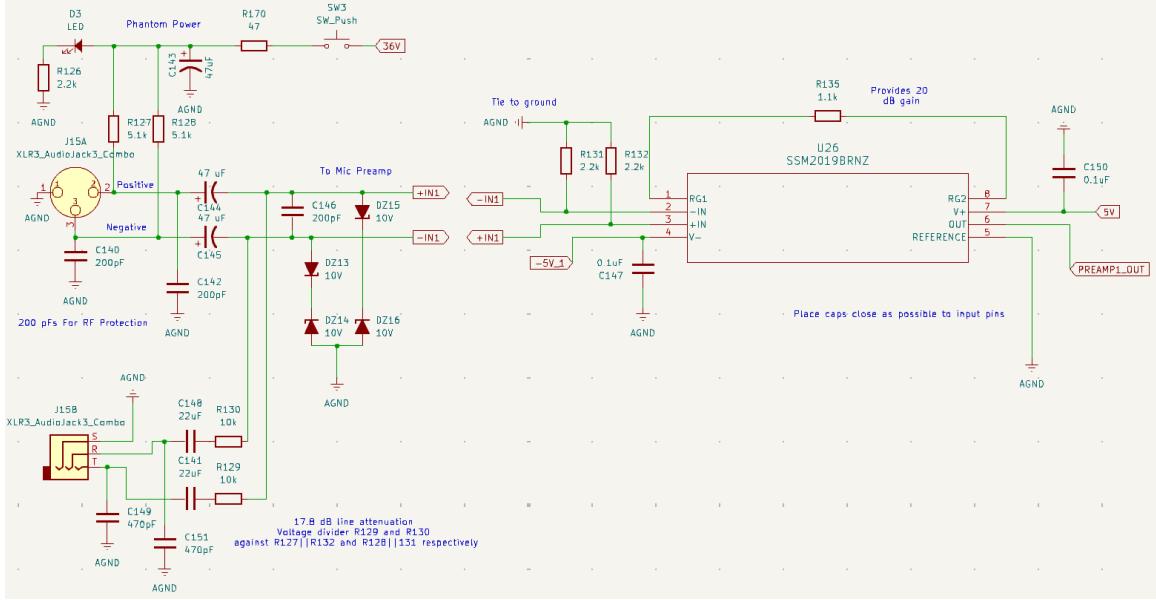


Figure 6.1: Schematic of Microphone/Line Preamplifier

There are quite a few design details of note. First are the various EMI protection capacitors strewn throughout the circuit. There are capacitors attached to the XLR and TRS inputs with values chosen based on THAT Corporation's design note [70]. Additionally, there is a 200 pF capacitor attached between the differential inputs into the preamplifier, with a value chosen based on the Analog Devices SSM2019 datasheet [71]. Similarly, there are 0.1 uF decoupling capacitors placed as close as possible to the supply pins of the SSM2019 IC.

Also of note is the phantom power implementation. The phantom power implementation uses 36 volts. This design is not adherent to the IEEE phantom power specification and is brought into the circuit with custom resistor values of 5.1k Ohms. This design is based on a reference design where the author utilizes custom phantom power voltages [14].

For the line input, the voltage is attenuated by 17.5 dB. This attenuation will allow for extra headroom for the line-input to operate without clipping the operational amplifier or the ADC. The attenuation is achieved by a voltage divider. The T and R line signals are attenuated by a voltage divider between R129 and R127—R132, and R130 and R128—R131, respectively. This yields a voltage attenuation of

$$\text{Attenuation}_V = \left( \frac{5.1k \parallel 2.2k}{10k + 5.1k \parallel 2.2k} \right) = 0.133$$

which translates to a decibel attenuation of

$$\text{Attenuation}_{dB} = 20 \times \log(0.133) = 17.5.$$

Once the microphone or line signal is brought into the preamplifier IC, a gain of 20 dB is applied. This gain is set by wiring a 1.1kOhm resistor between pins

1 and 8 of the SSM2019 IC as seen in the diagram. This is straight from the datasheet [71].

Because of the capability of the ADC to amplify the signal by over 30 dB, the preamplifier design above meets the gain requirements.

### 6.1.2 LDO Circuitry

As mentioned in previous sections, LDOs offer a near-DC output voltage that is able to be done with very minimal circuitry. In addition, LDOs have significantly less noise than Buck converters at the cost of being more energy-inefficient. Because we are working with analog devices that need noise-free power supplies, using an LDO is a must for those analog power pins. Because we are only using the LDO voltage output for specific analog power pins, we do not need high current output, allowing the low-cost NCP1117 to be used. Similarly, we can also use a TPS73601 for lower voltage outputs. Shown below is the very simple circuit needed to operate the LDO, with only a few capacitors/resistors required.

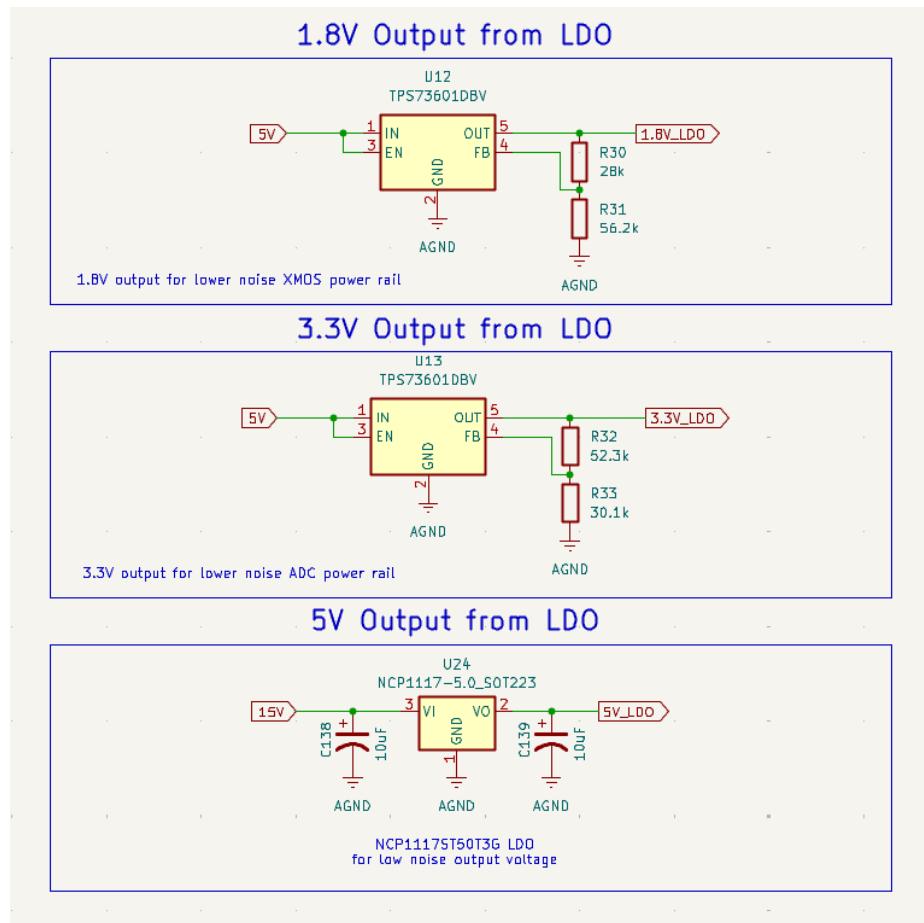


Figure 6.2: Schematic of the three LDOs used in power circuitry

### 6.1.3 Buck Converters

The circuitry required in the device requires various different operation voltages. As such, the converted 15V from beforehand will once again need to be converted into lower voltages. This is required for both analog and digital capabilities, with 5V and 3.3V being the norm. Additionally, using the xcore chip requires additional downregulated voltages of 1.8V and 0.9V.

Buck Converters utilize switching at very high frequencies (in the orders of MHz) to allow for synchronous DC-DC conversion with low quiescent current and high output current [28]. These devices are more complicated than an LDO and require a feedback pin to ensure the proper output voltage, though they are relatively simple to integrate into existing circuits. The successful operation of the converter also relies on the current load, with higher loads allowing for more efficient conversions due to the resultant frequency of the switching being higher. Because this device operates through constant switching, there is a transient response, and there can also be a subsequent voltage ripple. One of the main features of the leading Buck converters is ensuring these transients and ripples are as small as possible.

#### 5V Downregulator

The first buck converter that is required must convert the 15V input from the wall AC adapter into 5V for the various circuitry power supplies. Using the TPS82140, the schematic shown in Figure 6.3 is as specified by the datasheet [72]. This specific converter is meant for higher voltage inputs (up to 17V), which is necessary for the original AC-DC converted voltage. As per Eq. 1, we needed to set the values of R5 and R6 to obtain an output voltage of 5V. Using existing resistor values, it was found that 523K and 100K were sufficient resistor values for an output of 5V. The rest of the capacitor values are the recommended values as per the datasheet, and R<sub>PG</sub> (Power Good resistor) was also given in the datasheet to be 100k. One advantage of the TPS82140 is that the inductor required for the buck conversion is built into the IC, and thus allows for a final circuit of lower complexity and size.

$$V_{\text{out}} = 0.8 \times \left(1 + \frac{R_5}{R_6}\right) = 0.8 \times \left(1 + \frac{523\text{k}}{100\text{k}}\right) = 4.98V \quad (\text{Eq. 1})$$

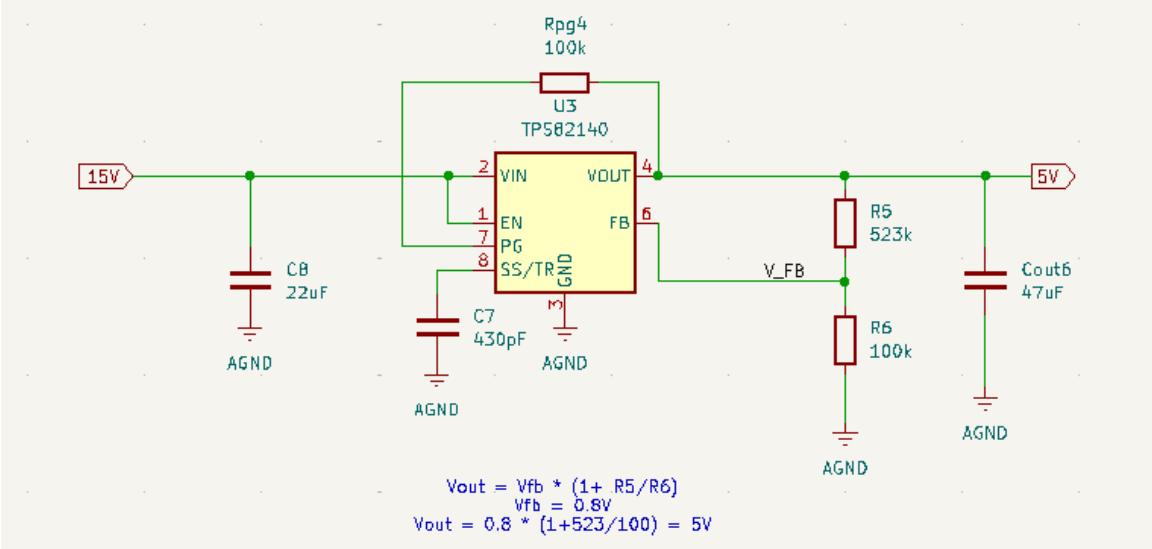


Figure 6.3: Schematic of 5V Buck Converter

The one thing of note in this buck converter is that because it needs to be powering further DC-DC converters (as well as power ICs), the current through the circuit is rated up to 3A, allowing for all power requirements to be met without damaging any circuitry. Decoupling capacitors are also utilized to mitigate any switching noise in the power lines.

### 3.3V Downregulator

An important factor to keep in mind is the time it takes to turn the converter on, which is given by Eq. 1. One should notice that the time decreases the farther away the output voltage is in reference to the input:

$$T_{ON} = \frac{V_{OUT}}{V_{IN}} \cdot 450\text{ns} \quad (\text{Eq. 1})$$

As per the TPS62822 manual [28], the normal switching frequency is 2.2MHz, which is sufficient for our application.

The most important parameters that need to be set are the output resistor values, which control the final desired voltage. Depending on the ratio of the upper to lower resistor, the output voltage can change according to Eq. 2:

$$R1 = R2 \times \left( \frac{V_{OUT}}{V_{FB}} - 1 \right) = R2 \times \left( \frac{V_{OUT}}{0.6V} - 1 \right) \quad (\text{Eq. 2})$$

In the case of the TPS62822, the feedback voltage is 0.6V.

With that, we created the required Buck converter circuitry below. The input capacitors are specifically used in order to filter any input voltage ripples and reduce any voltage drops. The output inductor was specified to be 470nH or higher,

while the output capacitors were specified to be at least 10uF and act to smooth the output.

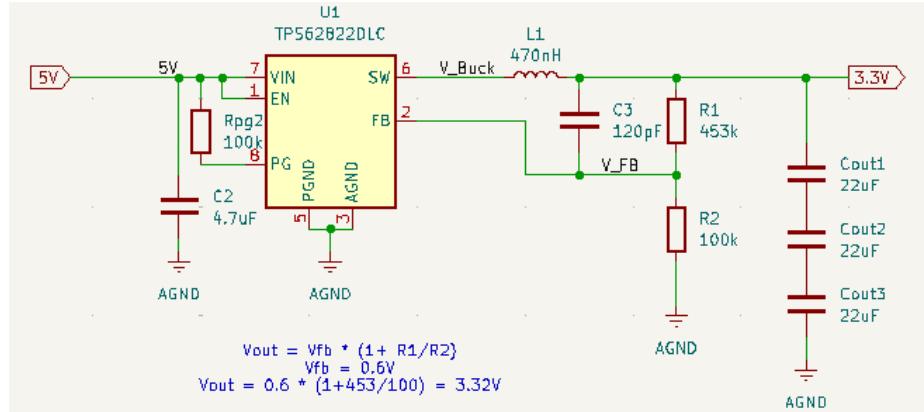


Figure 6.4: Schematic of 3.3V Buck Converter

### Simulation of 3.3V Buck Converter Performance

To demonstrate the real-world behavior of a sample buck converter circuit, we utilized TI Webbench Power Designer Tool to see the startup, steady state and load behavior using the circuit from 6.4. As shown by Figure 6.5, it can be seen that it takes  $\approx 500\mu s$  to reach the required 3.3V output with a 5V input [73]. This is notable as order of power supply startup is essential in the use of the xcore DSP chip (where the 3.3V rail needs to be fully powered before the 1V rail). Ensuring that our power sequencing is consistently proper is paramount to the successful operation of our circuitry.

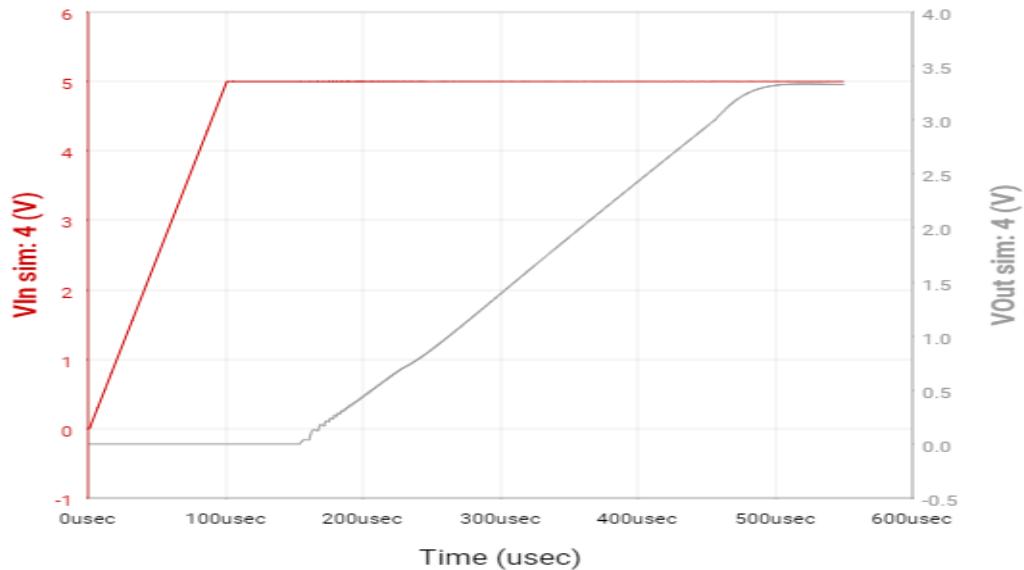


Figure 6.5: Simulation of Startup for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.

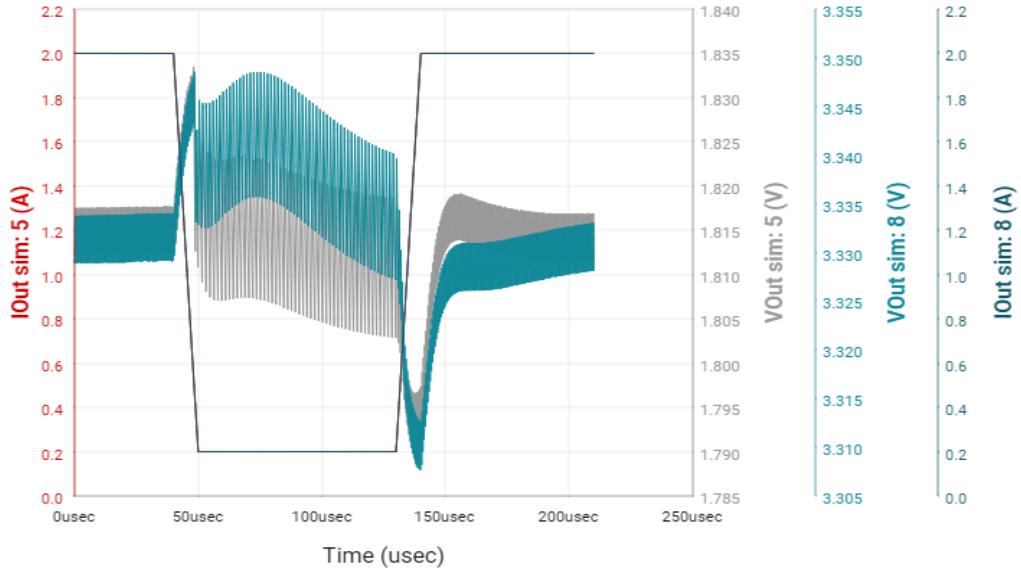


Figure 6.6: Simulation of Load for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.

In Figure 6.6, it can also be shown that under load, the output current for Buck Converter will be much higher. This allows the overall efficiency of the circuit to be higher, whereas if the circuit is not under load then the efficiency of the overall Buck Converter will be lower.

As shown by Figure 6.7, it is evident that, like the theory notes, the voltage output for a Buck Converter is AC in nature, but at extremely high frequencies (in the order of MHz). This output, while sinusoidal in nature, is very controlled and deviates little from the set value (e.g. 3.3V or 1.8V), usually 0.005V or so. Because the oscillating frequency is so high, the AC voltage can still be seen as DC-like to the devices using the power rails, but one must take special care in the case of powering analog devices due to the AC switching noise present. Because the voltage regulation is extremely consistent, Buck Converters are an acceptable use in our circuit due to the high efficiency provided by the device itself.

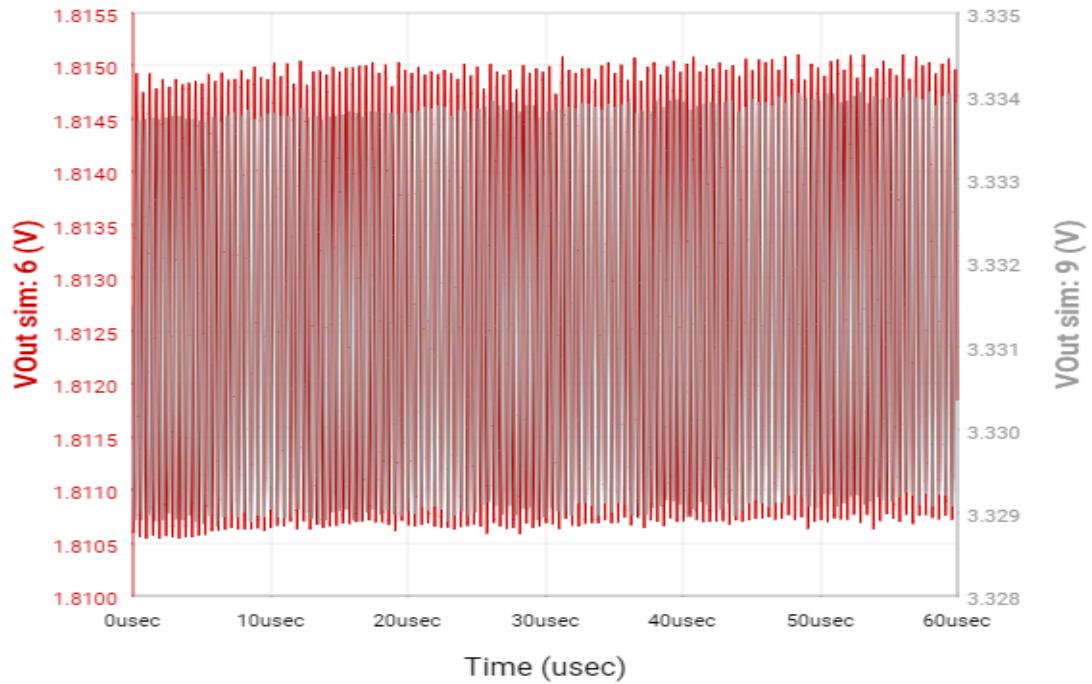


Figure 6.7: Simulation of Steady State for 3.3V Buck Converter, Courtesy of Texas Instruments Inc.

### **1.8V Downregulator (Required for xcore chip)**

Utilizing the same circuitry as the 3.3V regulator, our 1.8V regular is required for xcore chip to operate successfully. Due to the ease of configurability using the TPS62822, this only required R3 and R4 to be altered to have a ratio of 2. The resistor values are not extremely small to limit the current going through  $V_{FB}$ , otherwise the usable current would be limited and the resistors utilized could be burnt out due to insufficient power requirements.

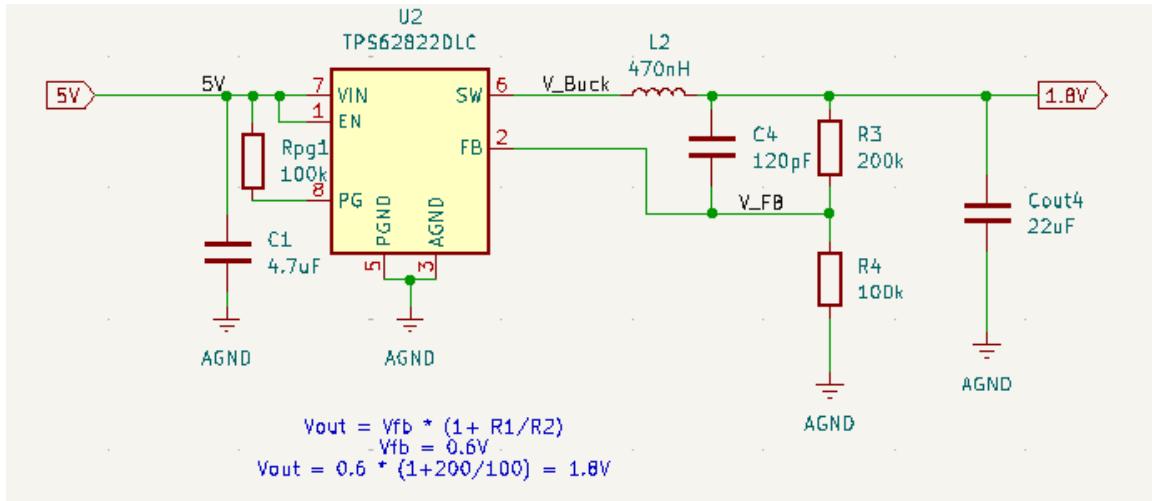


Figure 6.8: Schematic of 1.8V Buck Converter

### 0.9V Downregulator (Required for xcore chip)

Similar to the schematic for the 1.8V buck converter, the main difference in the 0.9V regulator is that it is only operated after the successful operation of the 5V downregulator in addition to a 0.9V enable signal. This is required due to the xcore MCU needing an order in which the power supplies activate.

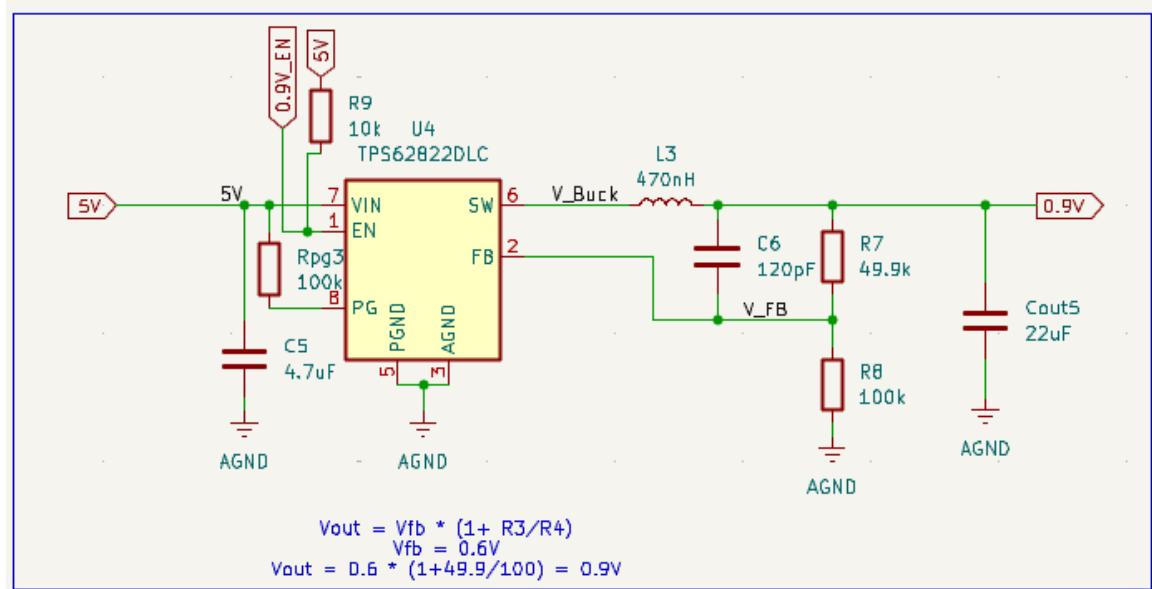


Figure 6.9: Schematic of 0.9V Voltage Regulator

#### 6.1.4 Phantom Power Circuitry

While the standard for phantom power is officially 48V, the vast majority of microphones (besides extremely expensive condenser mics) do not require the full 48V,

rather stepping the 48V down to a lower voltage. There is an excellent reference design online that utilizes 15V for phantom power [14] but a better design for the mixer's design is 36V, which would mitigate the overall current required. To do this, we are using the LT8338 as a Boost Converter to boost our original 15VDC input into 36VDC to power all 4 of our phantom power enabled inputs. As the voltage is relatively high for phantom power, the overall current used would be much less than the maximum 100mA that the LT8338 can output.

Shown in Figure 6.10, we are also using the basic phantom power circuitry of having two equivalent resistors on the positive and negative pins (6.8K for 48V) in addition to electrolytic capacitors for smoothing and zener diodes for voltage clipping. In this case, we used 10V zener diodes to ensure that the maximum voltage that the microphones get is around 10V, which is more than sufficient for phantom power operation and protects circuitry. There is also a simple enable switch with LED to indicate the use of phantom power for the mixer. The one switch pictured would indicate that phantom power is enabled for all four channels of the TRS/XLR combo-jack. After the XLR channels receive phantom power and are smoothed and clipped, the signals are then sent to the preamplification circuitry to then be processed.

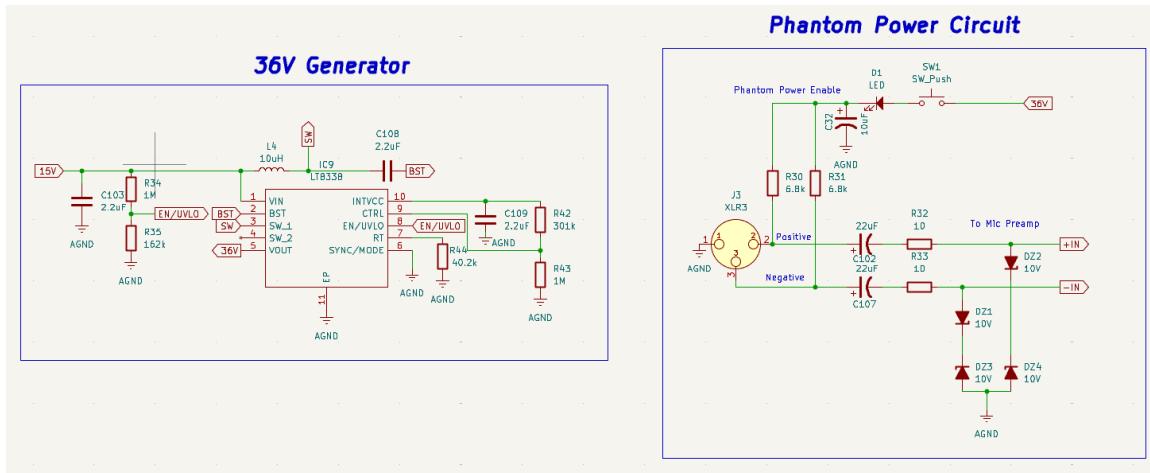


Figure 6.10: 36V Generator and Phantom Power enabled XLR circuit

### 6.1.5 ADC Circuitry

After the input signals are amplified, they must be converted to digital representations to be processed by the microcontroller and DSP chip. To do this, one must use an ADC, and since we are working with audio, we need to ensure that the signal quality is not degraded through conversion. To do this, we require a high SNR (ideally above 80) coupled with a sampling size of over 16 bits. While our mixer will require decently high quality conversion, due to budgetary and practical limitations, a 44.1KHz sample rate is sufficient as we are not working with specifically Hi-Fi audio technology. This limitation ultimately allows the processing of the data

to be significantly easier on the microprocessor itself, lowering the required RAM by multiple factors.

Due to the fact that we are going to be mixing many audio inputs, we chose an ADC that allowed for I2S communication while having the highest number of inputs possible. Using the PCM1865, we are able to have four channels be output per ADC and thus two will be required. This adds a slight level of complication to the routing of signals, as proper separation between the analog and digital pins is very important to mitigate noise.

Additionally included is the schematic for the voltage used to create a -5V rail for the OPA2134 circuitry used as a buffer.

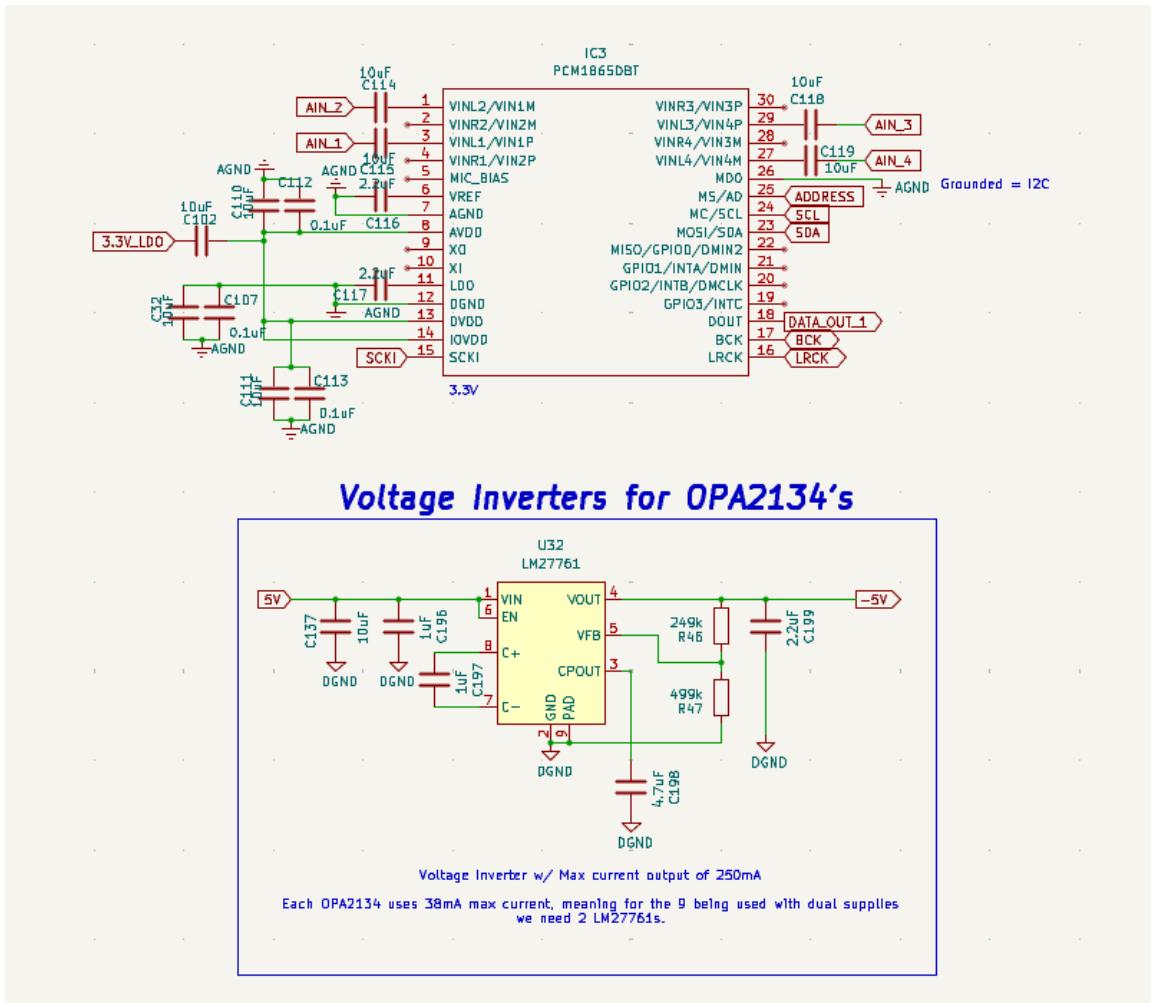


Figure 6.11: Schematic of one ADC and Voltage Inverter

### 6.1.6 DAC Circuitry

After processing the signals, they must be converted back to analog to be heard by the musicians and audience. Just as bitrate mattered for the input conversion, it also matters for the output conversion with how accurate the final signal will be

in comparison to the digitized version. Inside the mixer, there only needs to be a singular DAC as the two desired outputs (line-out and headphone-out) can be obtained from the same signal after splitting it. If there were more desired audio buses, more DACs would need to be utilized. The DAC will be communicated to via the xcore DSP chip, similar to the ADC, and will be clocked in unison with the other microcontroller chips. There are very few external parts required in this specific DAC circuit, with only some resistors and electrolytic capacitors being sufficient for proper regulation. Although most ADCs and DACs can be powered by either 3.3V or 5V, choosing to operate with 5V allows the SNR to be higher and reduces sampling errors due to a larger reference voltage.

The chosen DAC (PCM1780) requires filtering after conversion to ensure audio integrity, which requires a second-order low-pass filter to prevent any high-frequency noise (e.g., switching) from being introduced. After going through the low-pass filter, the two audio signals going out would be the left and right audio channels. These signals are going to split off: one copy goes to the Headphone Amplifier for proper amplification and another goes to the line-out TRS connection for use as an input in an amplifier or PA system.

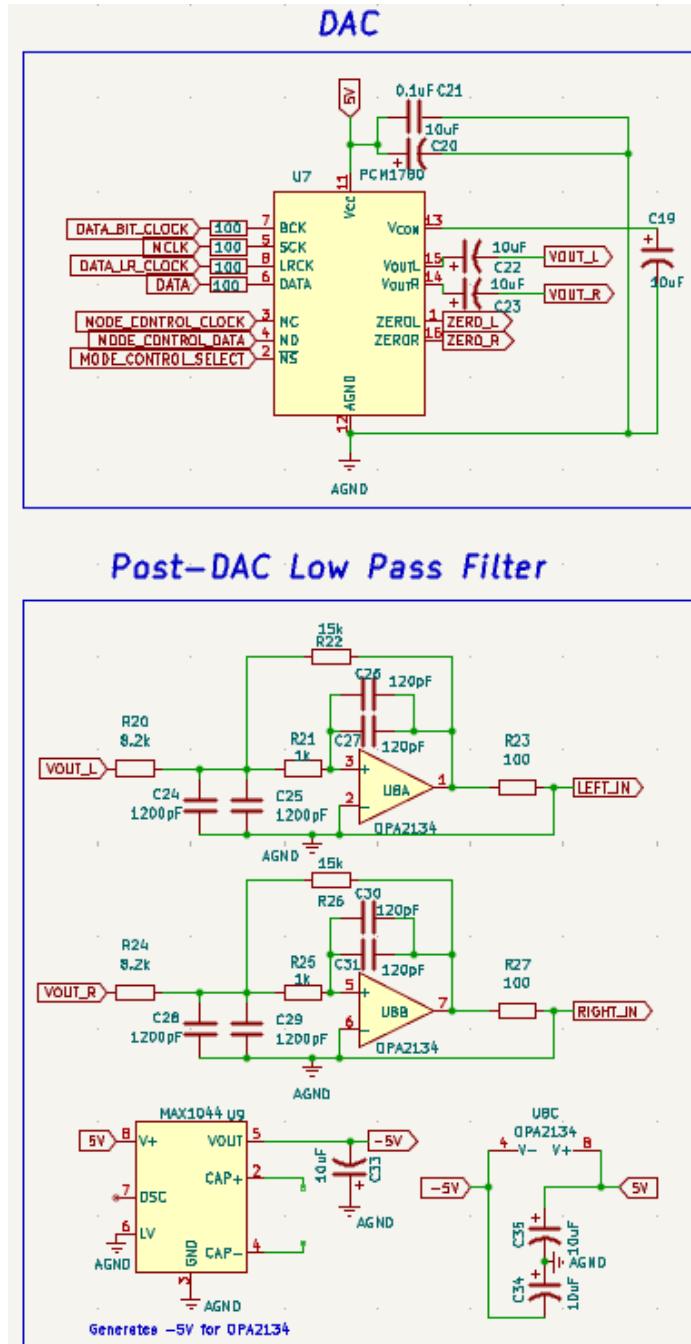


Figure 6.12: Schematic of DAC circuit and Post-DAC Low Pass Filter

### 6.1.7 Headphone Amplifier

After being converted back to an analog signal, the end user must be able to hear it. With a headphone output being standard for an audio mixer, it is only natural that our mixer also include one for monitoring purposes. To properly transmit a signal to headphones, the line-out signal (which is the signal obtained from the DAC and then sent through a buffer) must be amplified to a level that headphones are able to

drive. This technology is similar to a preamplifier (by way of being a post-amplifier) and there are many ICs on the market that are able to do this without introducing noise into the system.

In our case, we are using the TPA6110A2DGN, which offers a high SNR and requires minimal external circuitry (capacitors and resistors) to properly operate. Once output, the lines are then sent to a TRS connector for output, where the T connection corresponds to the left audio channel and R corresponds to the right audio channel (and the S connector always being the reference ground connection).

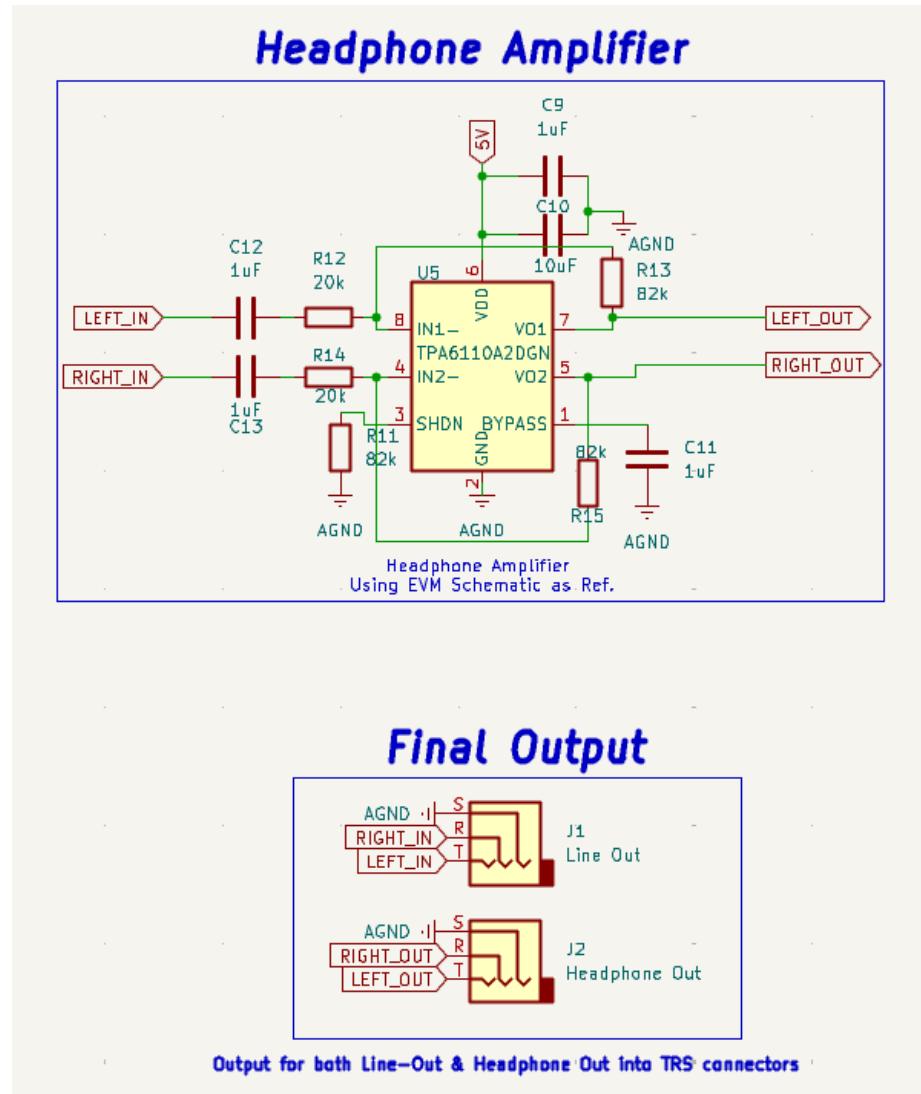


Figure 6.13: Schematic of Headphone Amplifier and Line/Headphone Output

### 6.1.8 Proper Grounding

The circuitry within the mixing device incorporates multiple grounding systems, specifically for analog and digital components. The distinction between these

grounds is critical due to the varying levels of noise present in the signal path. Analog circuits, which typically have low current requirements, are highly sensitive to noise, making proper grounding essential for maintaining audio quality [74]. Therefore, the analog ground must be isolated from the digital ground on the PCB to prevent interference from devices with high switching frequencies, which can introduce noise into the signal path.

In contrast, digital grounds are designed to handle noisy signals, as digital devices are less affected by noise. This is particularly important in power regulation components like buck converters, which operate at high switching frequencies. Ensuring that analog and digital signals are routed to separate ground planes is crucial for the reliable operation of the circuits.

Although analog and digital grounds must be separated to function correctly, they still need to be connected at a single point, known as the "star ground." This narrow junction allows for a shared reference voltage between the two planes while minimizing the switching noise that could affect the analog ground. Proper placement of the star ground is essential for maintaining signal integrity. As illustrated in Figure 6.14, keeping analog and digital components as isolated as possible is key to ensuring a properly functioning circuit.

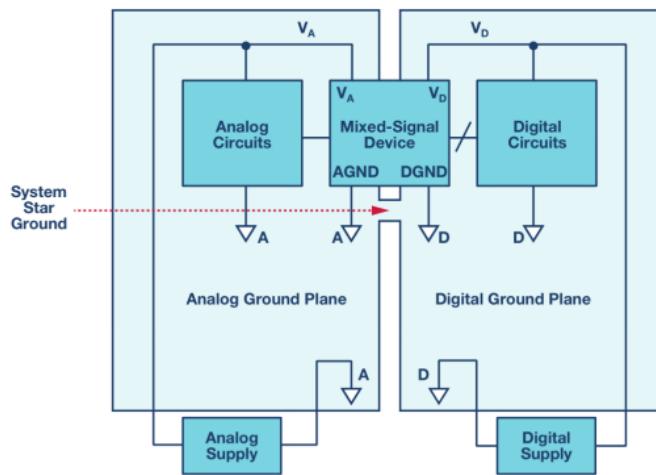


Figure 6.14: Example Organization of Mixed Signal Circuit

# Chapter 7

## Software Design

### 7.1 Software Requirements and Constraints

Our project aims to be a fully-fledged digital mixer, and we aim to deliver a product that will meet professional constraints for audio quality and allowing for diverse functionality. The software that is required to meet our project's objectives requires a substantial amount of engineering effort.

To this end, we will adopt modern software development methodologies to guide the development of our software architecture – namely, the *Agile* methodology. The starting point for project development under Agile is to formally and exhaustively define the requirements of our software, from the perspective of the end users.

While we have outlined the overall goals of our project earlier in Section 2.3 of our report, here we will specifically focus on the requirements of our mixer's *software*, as well as other relevant information that have implications for our software design, and discuss how it will inform our architecture's design.

#### 7.1.1 User Stories and Acceptance Criteria

A foundational component of the Agile methodology is the use of *user stories* – natural language descriptions of desired functionality from the first-person perspective of users – to guide development and ensure that engineering effort is focused on directly implementing and servicing the requirements of end-users.

We will take these user stories into consideration when designing our software architecture, by ensuring that our design is as parsimonious as possible while still meeting the acceptance criteria for every following story point.

##### 1. BASIC AUDIO MIXER CAPABILITIES

- (a) **As a user**, I want to connect up to 8 analog input channels to the mixer so that I can route my instruments through the system.
  - **Requirement:** Support for 8 analog inputs with ADC.
  - **Acceptance Criteria:** The system can recognize and process up to 8 analog input channels without signal degradation.
- (b) **As a user**, I want to be able to control volume and panning for each input channel so that I can balance the audio mix.
  - **Requirement:** Implement volume and panning control for all input channels.

- **Acceptance Criteria:** Each input channel has adjustable volume and panning via the touchscreen interface.
- (c) **As a user**, I want the mixer to sum audio from multiple channels into a stereo output, so that I can send a combined signal to the speakers.
- **Requirement:** Implement summing functionality for stereo output.
  - **Acceptance Criteria:** The mixer sums multiple input channels and outputs a stereo signal to line out and headphone out.

## 2. CREATIVE AUDIO EFFECTS

- (a) **As a user**, I want to apply creative audio effects (reverb, delay, compression, EQ, chorus/flanger/phaser) to my audio inputs so that I can enhance the performance.
- **Requirement:** Implement DSP effects: reverb, delay, compression, EQ, chorus/flanger/phaser.
  - **Acceptance Criteria:** Each input can have multiple effects applied, with adjustable parameters, processed in real-time.
- (b) **As a user**, I want to adjust effect parameters through the touchscreen interface so that I can fine-tune the sound during performances.
- **Requirement:** Intuitive effect parameter control via touchscreen.
  - **Acceptance Criteria:** Each effect's parameters can be modified through the touchscreen, with visual feedback.

## 3. INTUITIVE TOUCHSCREEN INTERFACE

- (a) **As a user**, I want to use an intuitive touchscreen interface to control the mixer so that I can easily navigate between effects and input routing.
- **Requirement:** Develop a digital touchscreen interface with user-friendly navigation.
  - **Acceptance Criteria:** The touchscreen interface is responsive, easy to navigate, and allows quick access to mixer functions such as routing and effect control.

## 4. WIRELESS CONTROL VIA MOBILE APP

- (a) **As a user**, I want to control the mixer wirelessly from my mobile device so that I can adjust settings remotely during performances.
- **Requirement:** Implement wireless control via a mobile app (WiFi/Bluetooth).
  - **Acceptance Criteria:** Users can control key mixer functions (volume, effects, routing) via a mobile app on both Android and iOS platforms.

## 5. AUTO-MIX FEATURE

- (a) **As a user**, I want the system to automatically adjust the volume levels based on the input dynamics so that the mix is balanced without manual intervention.
- **Requirement:** Implement automatic volume balancing based on input levels.
  - **Acceptance Criteria:** The auto-mix algorithm adjusts volume in real-time, maintaining a balanced mix.
- (b) **As a user**, I want the system to automatically apply effects based on the

genre and input type (e.g., vocals, guitar) so that I can quickly set up the mixer.

- **Requirement:** Implement automatic effects selection based on genre and input type.

- **Acceptance Criteria:** The system selects and applies appropriate effects without manual configuration based on predefined genre profiles.

- (c) **As a user**, I want the system to automatically apply EQ to unmask instruments in the mix, so that all instruments can be heard clearly.

- **Requirement:** Implement automatic EQ adjustments to reduce masking between instruments.

- **Acceptance Criteria:** The auto-mix feature applies EQ automatically, ensuring all instruments remain distinct in the mix.

## 6. USB COMPUTER CONNECTIVITY

- (a) **As a user**, I want to connect the mixer to my computer via USB so that I can record and transfer audio data directly.

- **Requirement:** Implement USB connectivity for recording and data transfer.

- **Acceptance Criteria:** The mixer can be connected to a computer, allowing audio to be recorded and data to be transferred via USB.

### 7.1.2 United Modeling Language (UML) Diagrams

Throughout our software documentation, we will utilize diagrams from the United Modeling Language (UML), a technical standard for the visual modeling of the design, implementation, and behavior of software systems. Due to the clarity and conciseness offered by UML diagrams in the communication of the design choices we make regarding our architecture, as well as their ubiquity in industry, we will prefer to use UML standards when producing visual models of our software design. We will provide expository detail about diagram types as they are used.

### 7.1.3 Use-Case Diagram

The use-case UML diagram depicts a high-level overview of the possible interactions that users can have with a system. It is a type of *behavioral* diagram, which are a class of diagrams that display the dynamic behavior of systems (as opposed to *structural* diagrams, which display the static structure of systems). The use-case diagram represents the overall functional requirements of the system, displaying the system's inputs produced by users and outputs consumed by users, as well as the high-level actions that the system must take in response to inputs.

### 7.1.4 User Roles

As shown in figure 7.1, there are four distinct roles that a user can have in relation to the mixer, as determined by their primary form of interaction with it:

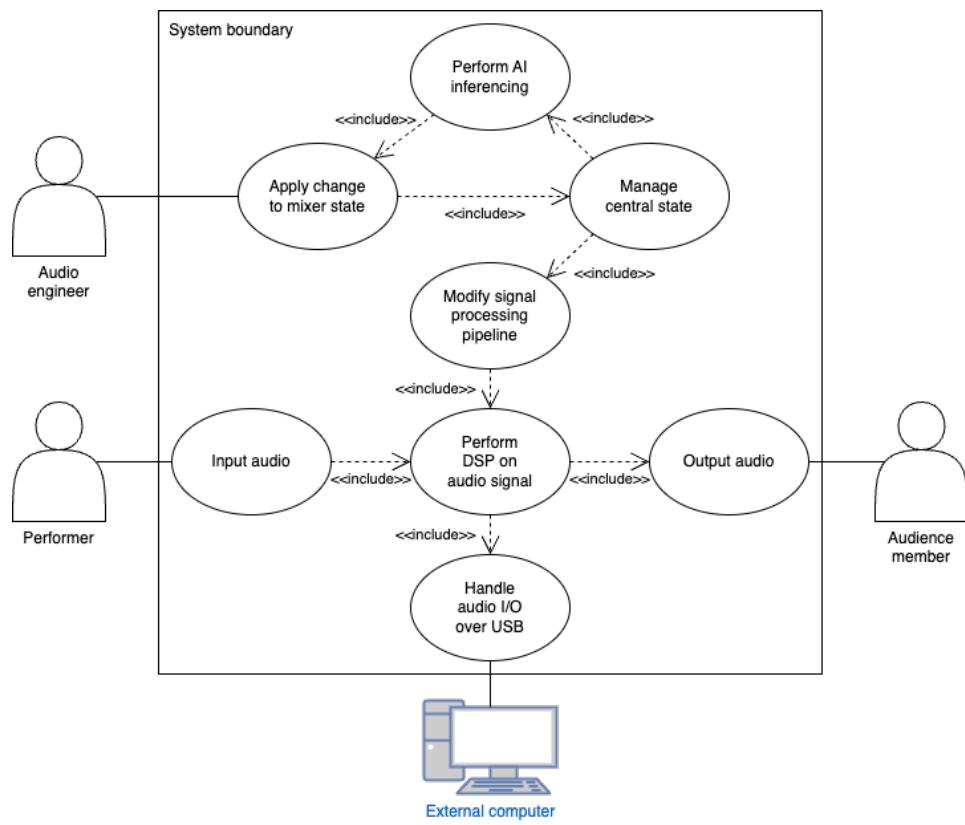


Figure 7.1: Use-case diagram for mixer software

**Performers** generate input signals, most of the time using instruments or microphones, to be converted to digital signals, processed, and mixed.

**Audio engineers** determine what digital processing to perform on the input signals, in order to achieve the best overall mix, both on a technical and aesthetic level. It is this action that is often simply referred to as "mixing".

**Audience members** consume the output signal, being the summation of the processed input signals, and converted back to analog to be fed to a transducer, such as a loudspeaker or PA system.

**External computers** utilize the mixer as a USB audio interface, treating the mixer's inputs and outputs as an extension of the computer's audio recording and routing capabilities.

### 7.1.5 Use-Cases Correspond to Software Modules

An essential task in designing complex software systems is to decompose the system into self-contained modules, connected by simple, stable interfaces. These modules should be constructed such so that they encapsulate groups of closely linked data and actions, and have clearly defined purposes.

The individual use-cases in the diagram are natural-language definitions of major actions that the system should take in response to actions performed by users. This lends the use-case diagram as being a convenient blueprint for determining the scope and interactions of formal modules, as will be expanded upon in the upcoming section 7.2.

### 7.1.6 Constraints and Implications for Software

## 7.2 Development of Software Architecture

In this section, we aim to provide readers with a detailed overview of the considerations that went into designing software architecture, building up to it from first principles. We will not only describe the "what" – the actual structure and behavior of our system, but also the "why" – details regarding design decisions, alternatives considered, and trade-offs involved in our chosen approach. As for the "how" – implementation and development details, we discuss this the next section 7.3.

Given the diverse set of functionality that we intend on implementing, a certain degree of complexity in our design is inevitable. Complexity in any software project is undesirable, but it is especially hazardous in our project, with the extent of the work that is required to achieve our minimal viable product goals; unmanaged complexity may result in failure to deliver a functional product all together. This makes good software engineering practices all the more important; special attention will be directed toward the decision of how to **modularize** our system.

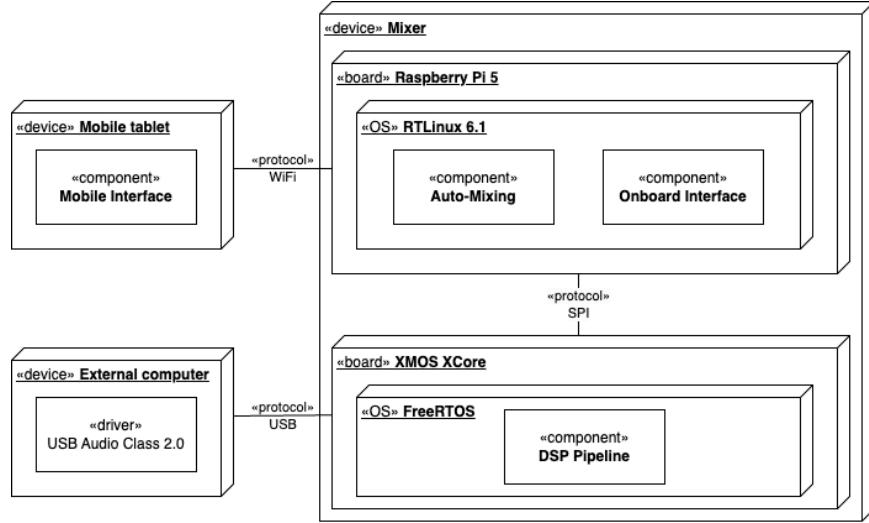


Figure 7.2: Deployment diagram of computing hardware

### 7.2.1 System Topology and Deployment

First, we must address a major constraint on our software design – the physical topology of the system’s computing hardware. The organization of our software must conform to the nature of the hardware it runs on. We will display the major hardware components, as well as their primary software responsibilities with a UML deployment diagram.

#### Onboard SoCs

To reiterate, the onboard computing hardware on our mixer is comprised of two system-on-a-chips: the **Raspberry Pi 5** and the **XMOS xcore**. They are connected to each other via the **SPI** communication protocol, the purpose of each will be discussed later in section 7.3.1. As for the SoCs, their primary responsibilities are as follows:

**Raspberry Pi 5** General purpose computer, to be used for all non-DSP tasks – AI inferencing, onboard touchscreen UI, and connection with mobile device.

**XMOS xcore** Specialized crossover platform for handling real-time DSP, including effects processing and handling audio I/O over USB.

#### External Devices

The strict boundary of our mixer’s hardware is indicated by the *Mixer* device node in figure 7.2. However, the functionality of our system extends to two external connected computing devices:

**Mobile tablet** Runs the mobile remote-control user interface, connected to the mixer over a WiFi connection.

**External computer** Utilizes the mixer as an audio interface over USB, as explained in section 2.2.1. All major operating systems provide built-in drivers for USB audio interface support; these generic drivers are called *class drivers*, and should enable plug-and-play functionality for our mixer's interface functionality.

When modules communicate with each other. If the communication occurs between hardware components, it will take place over one of the four communication protocols displayed in the deployment diagram, and we will explicitly specify which one. Otherwise,

## 7.2.2 Managing Data Flows

The primary engineering challenge that will inform the design of our architecture is the management of the flow of data between various locations in the system. Especially considering the complex topology of our system, with data consumers and producers scattered between disparate locations in hardware, a solid understanding of the connections that each module requires is crucial to designing a robust architecture.

### Data Needs of System-Level Components

We will begin by taking into consideration the data needs of the four preliminary software components from the deployment diagram (figure 7.2). The way these components were obtained was by grouping related user stories (from section 7.1.1) under umbrella terms, for the purpose of indicating the primary responsibilities of each hardware component in our system.

The components obtained by this loose categorization of desired functionality lends itself as a blueprint for formal modularization. These components, as they are, make for effective system-level subdivisions; they encapsulate tightly linked behavior and state, and are loosely coupled with each other. Here, we will describe what coupling is entailed between these components:

**User interfaces** Produces commands in response to user actions, such as adjusting mixer settings. The effect of these commands should be to mutate the current state of the mixer. Since both interfaces (mobile and onboard) must remain synchronized to each other, they must also consume this state information to update their local state. In addition, audio data must be consumed by the interfaces for the visual monitoring of audio levels and frequency spectra.

**AI auto-mixer** Consumes audio data in the form of spectrograms; this is the input data to the model, which then produces parameters to be applied to the mixer's state.

**DSP pipeline** Consumes state commands, which direct it to modify the processing pipeline in some fashion. Produces audio data, which is extracted from

the processing stream and forwarded to other modules that require real-time audio data.

## Types of Data

As evidenced by the data needs of our modules, there are two primary types of data that are used for inter-module communication in our system: **state data** and **audio data**. These distinct types of data are diametrically opposite of each other in nature, and require different handling and mechanisms to ensure correctness in communication.

**State data** refers to *discrete*, trigger-based changes in system state, such as adjusting a fader, enabling/disabling effects, or changing a parameter like panning. These events are instantaneous and represent individual updates to the system's state.

State data must be synchronized between subscribing (consuming) modules, including the DSP, AI auto-mixer, and both user interfaces, via a central mechanism. Each module will react to state changes as needed. For example, if the onboard interface triggers a volume change, the DSP will apply this change to the audio, while the mobile interface will update its UI to reflect the new state. We discuss the centralized mechanism for this in the upcoming section 7.2.2.

**Audio data** refers to the *continuous* stream of audio samples that flows between the DSP, the AI auto-mixer, and the visual monitoring components of the system. In contrast to the discrete nature of event-based data, audio data is continuous and must be handled in real-time for effective audio processing and visualization.

Beyond actual mixing in and of itself, audio data is central to the auxiliary functions of the mixer. The AI auto-mixer requires it to generate inferences, and the user interfaces require it for visual monitoring (e.g., displaying audio levels in the meters of channel strips, frequency spectrum visualizer for equalizer effect, etc.). Without mechanisms to ensure reliable and continuous audio data streaming from the DSP module into the modules on the Pi, these functionalities would be compromised.

## Centralized Manager Modules

Due to the need for synchronization in our state data as well as real-time distribution of our audio data, we will employ centralized managers to be intermediate hubs for inter-module communication. These managers will run on the Raspberry Pi 5 and be responsible for the routing, management, and transformation of data between system modules.

**State manager** Handles all event-based, discrete state data. It will follow a **publisher-subscriber design pattern** to ensure that all relevant modules are synchronized in real-time. When a state change occurs (e.g., volume adjustment, effect toggling), the State Manager will publish this update, and all subscribing modules—such as the DSP, AI auto-mixer, and both user interfaces—will receive and apply the change accordingly. The State Manager ensures consistency across all modules, maintaining a single source of truth for mixer parameters, thus avoiding conflicts or discrepancies in system state.

**Audio manager** Handles the continuous flow of audio data within the system. Its primary role is to manage the streaming audio data coming from the DSP and ensure it is routed appropriately. It will forward high-fidelity audio streams to the **AI auto-mixer** for real-time inferencing and processing. Simultaneously, it will downsample the audio for the **user interfaces** (both onboard and mobile) to provide real-time visual feedback, such as audio levels and frequency spectrums. The Audio Manager ensures efficient handling of high-throughput audio data while maintaining the low-latency requirements necessary for real-time processing and visualization.

By separating state management and audio data handling, and centralizing the intermediary logic between the various processing and interface modules, we create a modular system architecture that is easier to maintain and extend.

## 7.3 Implementation and Concrete Design

### 7.3.1 Audio Transfer

#### Data transfer from xcore to Raspberry Pi

As discussed in 3.2.2, our choice of method for data visualization will be Fast Fourier Transform signals. Due to their popularity in industry, as well as their ease of use, they were the obvious choice as a means to visualize the audio information the xcore sends to the Raspberry Pi. This section provides a detailed and structured explanation of the process involved in transferring FFT signals from the xcore to the Raspberry Pi using SPI.

**FFT Data Preparation** The first step in this process involves the transformation of raw audio into FFT data on the xcore. This transformation is performed using the Fast Fourier Transform algorithm, allowing us to extract key frequency components and their magnitudes. Upon performing the FFT, the xcore generates complex numbers for each frequency bin, consisting of both real and imaginary components. These numbers represent the amplitude and phase of the signal at each frequency, essential for audio analysis and visualization. In our system, it is important to structure this data efficiently for transmission via the SPI interface. Each FFT bin will be represented as a ??? (either 16 bit

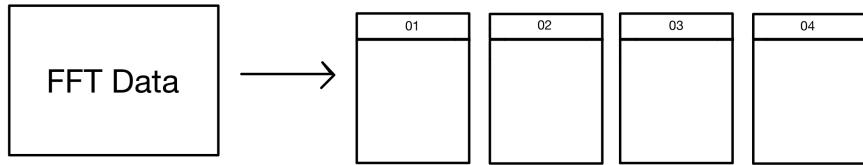


Figure 7.3: Packetization of FFT Signals

integer or 32 bit floating point), due to ??. The complex numbers are stored in a buffer on the xcore in preparation for transmission.

**Packetization of FFT Data** Given the nature of SPI as a serial communication protocol, it is essential to packetize the FFT data before transmission. The goal here is to divide the large FFT data into smaller, manageable packets for efficient transfer. For our project, a ???-point FFT will be performed. This results in ???/2 real values and ???/2 imaginary values, which will be split into smaller packets of ??? (32 or 64 data points). Packetization ensures that the data can be transmitted in chunks, reducing the risk of data overflow and enhancing error handling. Each packet will consist of a header that identifies its sequence in the data stream, followed by the actual FFT data. The header will also include error-detection mechanisms to ensure the integrity of the data. This structure allows the Raspberry Pi to verify the correctness of the received data and request retransmission if necessary.

**SPI Configuration** The xcore acts as the SPI master, controlling the data transfer to the Raspberry Pi. On the xcore, the clock speed, mode configuration, and data word size must be adjusted appropriately. The SPI clock speed will be ??? MHz, as this is the required data rate of the Raspberry Pi as the SPI slave. The SPI mode will be configured to ensure that the clock polarity and phase are synchronized between the xcore and the Raspberry Pi by operating in mode ??. This mode came from the fact that ??. For the data word size, we will be using ???-bit words. This word size was selected to optimize data transmission. Once the SPI interface is configured, the xcore sends out the FFT data packet-by-packet. Each bit is shifted out on the MOSI (Master Out Slave In) line, synchronized with the clock signal. The CS (Chip Select) line then is used to select the Raspberry Pi as the active recipient.

**Processing and Visualization** After the complete FFT frame is successfully received by the Raspberry Pi, it proceeds to process the data for visualization purposes. The FFT signals are used to generate a visual representation

of the audio spectrum, which is then displayed on the on-board LCD of the mixer via UNIX domain socket. Furthermore, the processed FFT data is then transmitted to the mobile tablet via WebSockets for remote visualization. This process is detailed in ?? and 7.3.1, respectively.

### Data transfer from Raspberry Pi to mobile tablet

Once the Fast Fourier Transform signals are successfully transferred from the xcore to the Raspberry Pi, the subsequent phase of the project involves transmitting these signals to a mobile application for real-time visualization and processing. The primary challenge here is to ensure efficient, low-latency communication between the Raspberry Pi and the mobile application while maintaining the integrity of the FFT data during the transfer. This section details the process for transferring FFT data from the Raspberry Pi to a mobile application, exploring the communication protocols involved and the steps required to facilitate real-time interaction with the audio data.

**FFT Data Preparation** After receiving the FFT data from the xcore, the Raspberry Pi stores this data in memory. Each FFT frame consists of a series of complex numbers, representing the real and imaginary components of the frequency spectrum of the audio signal. These FFT values, which reveal frequency and phase information, are crucial for visualizing the audio's spectral content in real time. The first step in preparing the FFT data for transmission is serialization. Serialization refers to converting the in-memory data structure (the FFT array) into a format that can be transmitted over a network. The Raspberry Pi can serialize the FFT data into commonly used formats. Although JSON would be the easiest to work with due to previous experience as well as human-readability, we will be using a binary format as FFT signals generate a lot of data. For these large datasets, binary formats are more efficient and reduce overhead during transmission, especially in real-time applications. (Protobuf or CBOR)

**Establishing Communication** In order to send the FFT data from the Raspberry Pi to the mobile application, a communication protocol must be utilized. For our project, we will be using WebSockets, as it allows for real-time data transfer, as opposed to traditional HTTP/REST APIs which are not real-time and must be used by periodically sending requests to the server. By using WebSockets, we simplify our data communication pipeline with one active connection that allows for full-duplex connection, as opposed to HTTP/REST APIs, where we would need to continuously establish connections to transfer data.

**Processing of FFT Data on Mobile Tablet** Once the FFT data has been transferred to the mobile application via WebSockets, the next task is to process and visualize the data. The mobile application will display the FFT data as a frequency spectrum, with the x-axis representing frequency bins and the y-axis representing the amplitude of each frequency. This provides the user

with a real-time view of the frequency components of the audio signal. As a result of using WebSocket-based communication, the mobile app continuously updates the visualization as new FFT data is received, offering us a dynamic representation of the audio spectrum.

# Chapter 8

## Administrative Content

The following chapter covers the administrative content related to the mixer project. Topics include budget estimates, timelines, and division of labor. Numbers and figures are still tentative at this stage of the project.

### 8.1 Budget Estimates

The budget estimate for our project is shown in Table 8.1 below. The parts are nearly finalized, however, some slight modifications may still be made based on further research.

| Item                       | Quantity | Price Per Unit | Adjusted Price |
|----------------------------|----------|----------------|----------------|
| Raspberry Pi 5             | 1        | \$80           | \$80           |
| Raspberry Pi Touchscreen   | 1        | \$60           | \$60           |
| Raspberry Pi Display Cable | 1        | \$1            | \$1            |
| PCM1865 ADC                | 2        | \$3            | \$6            |
| NCP1117 LDO                | 1        | \$1            | \$1            |
| LM27761 Voltage Inverter   | 1        | \$1            | \$1            |
| TPS82140 Buck Converter    | 1        | \$5            | \$5            |
| TPS62822 Buck Converter    | 3        | \$1            | \$3            |
| OPA2134 Op Amp             | 8        | \$5            | \$40           |
| AD SSM2019 Op Amp          | 8        | \$5            | \$40           |
| PCM1780 DAC                | 1        | \$2            | \$2            |
| TPA6110A2DGN Headphone Amp | 1        | \$2            | \$2            |

|                             |     |       |              |
|-----------------------------|-----|-------|--------------|
| PCB Estimated costs         | 1   | \$50  | \$50         |
| Metal Casing                | 1   | \$20  | \$20         |
| XU316-1024-TQ128 (DSP Chip) | 1   | \$15  | \$15         |
| XLR Combo Jack Connector    | 4   | \$4   | \$16         |
| 1/4" TRS Jack               | 4   | \$1   | \$4          |
| Caps/Inductors/Resistors    | 100 | \$0.1 | \$10         |
| <b>Estimated Total:</b>     |     |       | <b>\$385</b> |

Table 8.1: Estimated Cost of Production

The estimated cost of the digital mixer is roughly competitive with the sale price of several competitor products such as the Behringer Flow 8 (\$299) and the Mackie MobileMix 8 (\$229). Of course, the price of the components is not an entirely fair comparison to make with the off-the-shelf price of a developed product. However, it is encouraging to the team that the cost of the project itself does not significantly exceed the price of a typical mixer in this category.

## 8.2 SD1 Documentation timeline

Below is the timeline for tasks within Senior Design 1. Senior Design 1 is strictly related to research and theoretical design, and as such, the tasks are related to the SD1 document. Timelines are relatively loose and are more strictly dictated by the actual submission timelines posted online in Webcourses.

| Task                               | Start Week      | Est. Duration (Weeks) |
|------------------------------------|-----------------|-----------------------|
| Brainstorming/Preliminary Research | Start of Summer | 12                    |
| Individual Research                | 8/19/2024       | 1                     |
| Finalize project scope             | 8/23/2024       | 1                     |
| Divide and Conquer                 | 8/26/2024       | 2                     |
| 30 page milestone                  | 8/19/2024       | 3                     |
| 60 page milestone                  | 9/9/2024        | 3                     |
| 90 page milestone                  | 9/30/2024       | 3                     |
| 120 page milestone                 | 10/21/2024      | 3                     |
| Final draft milestone              | 11/11/2024      | 3                     |
| Review                             | 11/21/2024      | 0.5                   |

---

Table 8.2: Project Documentation Milestone

## 8.3 SD2 Design Timeline

Below is the timeline for tasks within Senior Design 2. Senior Design 2 contains the development and prototyping phase of the project, and the tasks reflect that. The tasks and timeline may be updated with further research. It may be wise to move the timelines up considerably to account for unforeseen issues in hardware design and development.

| Task                          | Description                                                                                       | Start Week | Est. Duration (Weeks) |
|-------------------------------|---------------------------------------------------------------------------------------------------|------------|-----------------------|
| Interfaces between work units | Establish contracts between individual modules                                                    | 1/6/2025   | 1                     |
| Power supply                  | Test proper power delivery to hardware components                                                 | 1/13/2025  | 2                     |
| Mixer state protocol          | A standardized protocol for inter-module state communication is drafted                           | 1/20/2025  | 1                     |
| Audio work loop               | Audio successfully passes through xcore unit                                                      | 1/27/2025  | 3                     |
| Basic onboard GUI             | Minimal GUI implemented for testing purposes                                                      | 2/3/2025   | 2                     |
| Communications co-processing  | Developed successful communication schematic for WiFi and Bluetooth capabilities                  | 2/24/2025  | 3                     |
| Machine learning model        | Successfully predicts optimal mixing parameters for audio inputs to enable real-time audio mixing | 2/10/2025  | 4                     |
| Control & DSP integrated      | Raspberry Pi can send control signals to xcore                                                    | 3/3/2025   | 3                     |
| Compute Integration           | xcore and Raspberry Pi properly communicate with each other through appropriate protocols.        | 3/31/2025  | 4                     |

Table 8.3: Project Design Milestone

## 8.4 Work Distributions

Work distributions among the five team members are delineated in this section. Firstly, Michael Meyers is the hardware lead in the digital mixer project. He is

responsible for schematic design in areas such as voltage regulation, AD/DA conversion, and power amplification. He is also responsible for the design of the PCB.

Joshua Yu is the system architect and integration lead as well as the DSP effects lead. He is responsible for the overall vision of the project as well as the integration of hardware and software systems. He acts as a liaison between the hardware and software members of the team. In terms of his specific technical responsibilities, he is responsible for schematic design for both the MCU itself (GPIO, clock source, flash, etc.) as well as the preamplification circuitry. He is also assisting Michael Meyers in PCB level design. Furthermore, he is writing the DSP effects library.

Juni Yeom is the lead software developer, responsible for the design and implementation of the system architecture. He is overseeing the integration of the disparate software work performed by team members into a cohesive system. He is tasked with implementing the central data management process to handle the mixer's control and audio data, in collaboration with Sri Kankipati. Additionally, he leads the development of the user interface, ensuring intuitive control over mixer functions and real-time audio visualization.

Srinilai Kankipati is the device communications lead. He is responsible for the standardization of data transfer between both inter-device modules and external sources. He is also responsible for developing and configuring the interconnects between said devices. In terms of specific technical responsibilities, he is responsible for inter-device serial communication, the data transfer formats, and the development of the USB audio interface capabilities. Additionally, he is support for Juni Yeom, particularly in the realm of central state management and real-time performance. Finally, he is support for Joshua Yu in writing the DSP library.

Kevin Kurian is the auto-mix algorithm lead. He is responsible for researching state-of-the-art auto-mix machine learning models, creating models, and integrating them with the system. In terms of specific responsibilities, he is tasked with reading literature on the topic, researching and curating datasets, recreating and pruning model architectures for integration on embedded platforms, and evaluating model performance in the context of the mixer project.

A short summary table of responsibilities is listed below.

| <u>Task</u>                        | <b>Primary Person</b> | <b>Secondary Person</b> |
|------------------------------------|-----------------------|-------------------------|
| <b>Hardware Design</b>             | Michael Meyers        | Joshua Yu               |
| <b>DSP Design</b>                  | Joshua Yu             | Michael Meyers          |
| <b>Auto-mix Algorithm</b>          | Kevin Kurian          | Srinilai Kankipati      |
| <b>Software Design</b>             | Juni Yeom             | Srinilai Kankipati      |
| <b>Inter-device Communications</b> | Srinilai Kankipati    | Juni Yeom               |

Table 8.4: Duty Distribution

# Chapter A

## Citations

## Bibliography

- [1] E. T. Chourdakis and J. D. Reiss, “A machine-learning approach to application of intelligent artificial reverberation,” *Journal of the Audio Engineering Society*, 2017.
- [2] Behringer, “Behringer flow 8 8-channel digital mixer,” <https://www.gear4music.com/us/en/PA-DJ-and-Lighting/Behringer-FLOW-8-8-Channel-Digital-Mixer/3N67>, 2024.
- [3] Mackie, “Mackie mobile mixer,” <https://mackie.com/en/products/battery-powered/live-sound/Mobile-mix.html>, 2024.
- [4] Sweetwater, “Qsc touchmix-8 14-channel touchscreen digital mixer,” <https://www.sweetwater.com/store/detail/TouchMix8-qsc-touchmix-8-touchscreen-digital-mixer>, 2024.
- [5] Allen and Heath, “Cq-18t,” <https://www.allen-heath.com/hardware/cq/cq-18t/>, 2024.
- [6] Tascam, “Tascam model 16,” [https://tascam.com/us/product/model\\_16](https://tascam.com/us/product/model_16), 2019.
- [7] Soundcraft, “Signature 22 mtk,” <https://www.soundcraft.com/en-US/products/signature-22-mtk>, 2015.
- [8] Audiostance, “The ultimate guide to audio connectors and cables,” <https://www.audiostance.com/audio-connectors-and-cables/>, 2020.
- [9] Schematron, “Xlr balanced female to male diagram,” <https://schematron.org/xlr-balanced-female-to-13-stereo-male-wiring-diagram.html>, 2018.
- [10] L. Fuston, “Understanding signal levels in audio gear,” <https://www.sweetwater.com/insync/understanding-signal-levels-audio-gear/>, 2022.

- [11] J. Vautin, “dbv, dbu, and dbm,” <https://jeffvautin.com/2009/06/dbv-dbu-and-dbm/>, 2009.
- [12] C. J. Heiduska, “What does line level mean?”, <http://www.ovnilab.com/articles/linelevel.shtml>, 2006.
- [13] R. Elliott, “Phantom power - what is it and how it works,” <https://sound-au.com/articles/p-48.htm>, 2021.
- [14] DJJules, “Build the four-channel ssm2019 phantom powered mic preamp,” <https://www.instructables.com/Build-the-Four-Channel-SSM2019-Phantom-Powered-Mic/>, 2018.
- [15] R. B. et al., *Signals and Systems*. Rice University, 2021.
- [16] Arrow, “Analog to digital converter types and basic functions,” <https://www.arrow.com/en/research-and-events/articles/analog-to-digital-adc-converter-types-and-basic-functions>, 2024.
- [17] A. Devices, “Understanding sar adcs: Their architecture and comparison with other adcs,” <https://www.analog.com/en/resources/technical-articles/successive-approximation-registers-sar-and-flash-adcs.html>, 2001.
- [18] A. Devices, “Sigma-delta adc tutorial,” <https://www.analog.com/en/resources/interactive-design-tools/sigma-delta-adc-tutorial.html>, 2024.
- [19] A. Devices, “Basic linear design,” <https://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter6.pdf>, 2005.
- [20] J. Siau, “Interpreting thd measurements - think db not percent!,” <https://benchmarkmedia.com/blogs/applicationnotes/interpreting-thd-measurements-think-db-not-percent>, 2017.
- [21] M. C. Limited, “Fpga vs. microcontroller - what’s the difference?”, <https://www.mclpcb.com/blog/fpga-vs-microcontroller/>, 2023.
- [22] E. M. B. Consortium, “Eembc benchmark score viewer,” *Embedded Microprocessor Benchmark Consortium*, 2024.
- [23] beagleboard.org, “Design and specifications,” <https://docs.beagleboard.org/boards/beaglebone/ai-64/03-design-and-specifications.htmlbbai64-design>, 2024.
- [24] Digikey, “Beagl-bone-ai-64,” <https://www.digikey.com/en/products/detail/texas-instruments/BEAGL-BONE-AI-64/21283880>, 2024.
- [25] Espressif, “Esp32-s3 series datasheet,” *Placeholder Journal*, 2024.
- [26] STMicroelectronics, “Stm32h747/757,” <https://www.st.com/en/microcontrollers-microprocessors/stm32h747-757.html>, 2024.

- [27] T. Instruments, “Tps7a53b 3a, low-input voltage, low-noise, high-accuracy, low-dropout (ldo) voltage regulator,” <https://www.ti.com/lit/ds/symlink/tps7a53b.pdf>, 2024.
- [28] T. Instruments, “Tps6282x 5.5-v, 1-a, 2-a, 3-a step-down converter family with 1
- [29] Renesas, “Linear vs. switching regulators,” <https://www.renesas.com/us/en/products/power-power-management/linear-vs-switching-regulators>, 2022.
- [30] S. Craze, “The pan law within your daw explained in detail,” <https://samplecraze.com/tutorials/the-pan-law/>, 2024.
- [31] GCore, “What is websocket? — how does it work?,” <https://gcore.com/learning/what-is-websocket/>, 2023.
- [32] M. contributors, “The websocket api (websockets),” [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API), 2024.
- [33] S. J. Bigelow, “What is rest api (restful api)?,” <https://www.techtarget.com/searchapparchitecture/definition/RESTful-API>, 2024.
- [34] O. Router, “What is mqtt? a practical introduction,” <https://www.opcrouter.com/what-is-mqtt/>, 2024.
- [35] Radware, “What is coap? understanding the constrained application protocol,” <https://www.radware.com/security/ddos-knowledge-center/ddospedia/coap/>, 2024.
- [36] M. Afaneh, “Bluetooth low energy (ble): A complete guide,” <https://novelbits.io/bluetooth-low-energy-ble-complete-guide/>, 2022.
- [37] C. Steinmetz, “Automaticmixingpapers: A collection of papers related to automatic mixing.” <https://github.com/csteinmetz1/AutomaticMixingPapers>, 2023.
- [38] B. De Man, J. Reiss, and R. Stables, “Ten years of automatic mixing,” *Placeholder Journal*, 2017.
- [39] F. Pachet and O. Delerue, “On-the-fly multi track mixing,” *Advances in Engineering Software*, 2000.
- [40] D. Moffat and M. B. Sandler, “Approaches in intelligent music production,” in *Arts*, vol. 8, p. 125, MDPI, 2019.
- [41] E. Perez-Gonzalez and J. Reiss, “Automatic gain and fader control for live mixing,” in *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 1–4, IEEE, 2009.

- [42] U. Rave, “Automixing and its applications in music production,” 2021.
- [43] E. Perez-Gonzalez and J. Reiss, “Automatic equalization of multichannel audio using cross-adaptive methods,” in *Audio Engineering Society Convention 127*, Audio Engineering Society, 2009.
- [44] B. De Man and J. D. Reiss, “A knowledge-engineered autonomous mixing system,” in *Audio Engineering Society Convention 135*, Audio Engineering Society, 2013.
- [45] M. N. Lefford, G. Bromham, G. Fazekas, and D. Moffat, “Context aware intelligent mixing systems,” in *Placeholder Book Title*, Audio Engineering Society, 2021.
- [46] D. Koszewski, T. Görne, G. Korvel, and B. Kostek, “Automatic music signal mixing system based on one-dimensional wave-u-net autoencoders,” *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2023, no. 1, p. 1, 2023.
- [47] B. Kolasinski, “A framework for automatic mixing using timbral similarity measures and genetic optimization,” in *Audio Engineering Society Convention 124*, Audio Engineering Society, 2008.
- [48] E. M. Schmidt and Y. E. Kim, “Automatic multi-track mixing using linear dynamical systems,” *Placeholder Journal*, 2011.
- [49] A. L. Benito and J. D. Reiss, “Intelligent multitrack reverberation based on hinge-loss markov random fields,” in *Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio*, Audio Engineering Society, 2017.
- [50] D. Giannoulis, M. Massberg, and J. D. Reiss, “Parameter automation in a dynamic range compressor,” *Journal of the Audio Engineering Society*, vol. 61, no. 10, pp. 716–726, 2013.
- [51] A. Mason, N. Jillings, Z. Ma, J. D. Reiss, and F. Melchior, “Adaptive audio reproduction using personalized compression,” in *Audio Engineering Society Conference: 57th International Conference: The Future of Audio Entertainment Technology—Cinema, Television and the Internet*, Audio Engineering Society, 2015.
- [52] I. Goodfellow, “Deep learning,” 2016.
- [53] C. J. Steinmetz, S. S. Vanka, M. A. Martínez Ramírez, and G. Bromham, *Deep Learning for Automatic Mixing*. ISMIR, Dec. 2022.
- [54] C. J. Steinmetz and J. D. Reiss, “Deep learning for automatic mixing: Challenges and next steps.” <https://static1.squarespace.com/static/5554d97de4b0ee3b50a3ad52/t/618e84cf83e6fc75816e12bc/1636730122670/>

MDX\_Workshop\_at\_ISMIR\_Deep\_learning\_for\_automatic\_mixing.pdf, 2021.  
Workshop presented at ISMIR 2021, MDX Workshop.

- [55] M. Martinez Ramirez, D. Stoller, and D. Moffat, “A deep learning approach to intelligent drum mixing with the wave-u-net,” in *Placeholder Journal*, Audio Engineering Society, 2021.
- [56] S. I. Mimalakis, K. Drossos, T. Virtanen, and G. Schuller, “Deep neural networks for dynamic range compression in mastering applications,” in *Audio Engineering Society Convention 140*, Audio Engineering Society, 2016.
- [57] S. H. Hawley, B. Colburn, and S. I. Mimalakis, “Signaltrain: Profiling audio compressors with deep neural networks,” *arXiv preprint arXiv:1905.11928*, 2019.
- [58] M. A. M. Ramírez and J. D. Reiss, “Modeling nonlinear audio effects with end-to-end deep neural networks,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 171–175, IEEE, 2019.
- [59] M. Martinez Ramirez, J. Reiss, *et al.*, “End-to-end equalization with convolutional neural networks,” *Placeholder Journal*, 2018.
- [60] J. T. Colonel and J. Reiss, “Reverse engineering of a recording mix with differentiable digital signal processing,” *The Journal of the Acoustical Society of America*, vol. 150, no. 1, pp. 608–619, 2021.
- [61] D. Stoller, S. Ewert, and S. Dixon, “Wave-u-net: A multi-scale neural network for end-to-end audio source separation,” *arXiv preprint arXiv:1806.03185*, 2018.
- [62] C. J. Steinmetz, J. Pons, S. Pascual, and J. Serra, “Automatic multitrack mixing with a differentiable mixing console of neural audio effects,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 71–75, IEEE, 2021.
- [63] M. A. Martínez-Ramírez, W.-H. Liao, G. Fabbro, S. Uhlich, C. Nagashima, and Y. Mitsufuji, “Automatic music mixing with deep learning and out-of-domain data,” *arXiv preprint arXiv:2208.11428*, 2022.
- [64] S. S. Vanka, C. Steinmetz, J.-B. Rolland, J. Reiss, and G. Fazekas, “Diff-mst: Differentiable mixing style transfer,” *arXiv preprint arXiv:2407.08889*, 2024.
- [65] L. Gabrielli, S. Tomassetti, S. Squartini, C. Zinato, *et al.*, “Introducing deep machine learning for parameter estimation in physical modelling,” in *Proceedings of the 20th international conference on digital audio effects*, 2017.
- [66] G. S. D. K. C. D. Jacob Larson, James S. Denford, “The evolution of artificial intelligence (ai) spending by the u.s. government,” <https://www.brookings.edu/articles/the-evolution-of-artificial-intelligence-ai-spending-by-the-u-s-government/>, 2024.

- [67] E. Griffith, “Investors pour 27.1 billion into ai startups, defying a downturn,” <https://www.nytimes.com/2024/07/03/technology/ai-startups.html>
- [68] K. Houser, “Llms are a dead end to ai, says françois chollet,” *Freethink*, 2024.
- [69] M. McArdle, “Ai is coming for the professional class. expect outrage — and fear.” <https://www.washingtonpost.com/opinions/2024/04/29/ai-professional-class-low-skill-jobs/>, 2024.
- [70] T. Corporation, “Input and output circuits for that preamplifier ics,” <https://thatcorp.com/datasheets/dn140.pdf>, 2024.
- [71] A. Devices, “Self-contained audio preamplifier ssm2019,” <https://www.analog.com/media/en/technical-documentation/data-sheets/SSM2019.pdf>, 2011.
- [72] T. Instruments, “Tps82140 17-v input 2-a step-down converter microsip™ module with integrated inductor,” <https://www.ti.com/lit/ds/symlink/tps82140.pdf>, 2017.
- [73] T. Instruments, “Ti webench power designer,” <https://webench.ti.com/power-designer/>, 2024.
- [74] JLCPCB, “Understanding analog and digital ground in pcb design,” <https://jlcpcb.com/blog/understanding-analog-and-digital-ground-in-pcb-design>, 2024.

## Chapter B

## Copyright

When using copyrighted content, we sought direct approval from the companies to follow fair use laws. As such, proof for used figures is shown below.  
 Approval for Figure 3.4:

Aug 22, 2015 at 5:49 history asked

Fraisse

CC BY-SA 3.0

### You are free to:

**Share** — copy and redistribute the material in any medium or format for any purpose, even commercially.

**Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

### Under the following terms:



**Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.



**ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.

**No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.

### Approval for Figures 6.6, 6.7, 3.10 and 6.5:

⌚ Your attachment has been added successfully

Hi Michael,

Good Day.

Yes, you may cite the requested document for your project we only ask that you include "Courtesy of Texas Instruments Inc." under the image.

I hope this solution I have provided helps you. If any clarification particular to this request is needed, please reply to this email within the next 5 days and let me know before the case is closed by the system.

To close this case, just click the accept link together with this update or inform us that this case can be closed.

Best Regards,

Ray Vincent Aranzaso

Texas Instruments | Customer Support Center

## Approval for Figures 3.5 and 3.8:

Hi Michael,

My name is Alyssa Scarpulla, and I am an attorney from ADI's Intellectual Property department.

Thank you for reaching out and asking for permission to use the two figures in your paper. This email serves as permission to use the figures in your paper on ADC technology, please include the following disclaimer under each image "Analog Devices, Inc. ("ADI"), © 2024. All Rights Reserved. These images are reproduced with permission by ADI."

Please let me know if you have any additional questions.

Best,

**Alyssa Scarpulla**

Associate Counsel, Intellectual Property

Email [alyssa.scarpulla@analog.com](mailto:alyssa.scarpulla@analog.com)



[analog.com](http://analog.com) |

## Approval for Figures 3.9 and 3.11:

Hello Michael,

Here is the response I got from a member of our legal team.

Michael can use those two graphics in his research paper as long as he includes the following attribution on each page that contains the material: "Reproduced with Permission. © Renesas Electronics Corporation or one of its subsidiaries. All Rights Reserved."

Best regards,  
Cindy



10/11/2024 09:15 PM

## Approval for Figure 3.3:



Phil Nass <phil\_nass@sweetwater.com>

To: Michael Meyers



...

Tue 10/8/2024 5:56 PM

i Retention: UCF Delete after 10 Years (10 years) Expires: Fri 10/6/2034 5:56 PM

Oh I gotcha. Yeah that's not a problem. I would just make sure you cite where it came from, but that's no worries. Its just a standard power diagram, so no worries.

...

# Chapter C

## LLM Citations

Prompt and Output for figure 5.1. Output from ChatGPT model 4o.

Can you generate an image of a schematic of a preamp circuit with the SSM2019 IC?

The image contains two parts: a photograph of a circuit board and a schematic diagram. The photograph shows a green printed circuit board with two black integrated circuit packages labeled 'SSM2019'. Various resistors, capacitors, and other components are visible. The schematic diagram on the right shows a detailed circuit layout for a preamp using the SSM2019 IC. It includes multiple input paths (IN-2, IN-3, IN-5) and output paths (Output -5V, Output -15V, Output coprét, Output copalt). Power supplies are labeled -Pin V, -Pin 5, and -Pin 15v. The schematic is annotated with component values like 3G, SC, and 1n-15v.

Here is the schematic of the preamp circuit using the SSM2019 IC as described. Let me know if you need any further details or adjustments!

◀ ▶ 🔍